Theory Track
Neural & Cognitive Sciences Track
of the

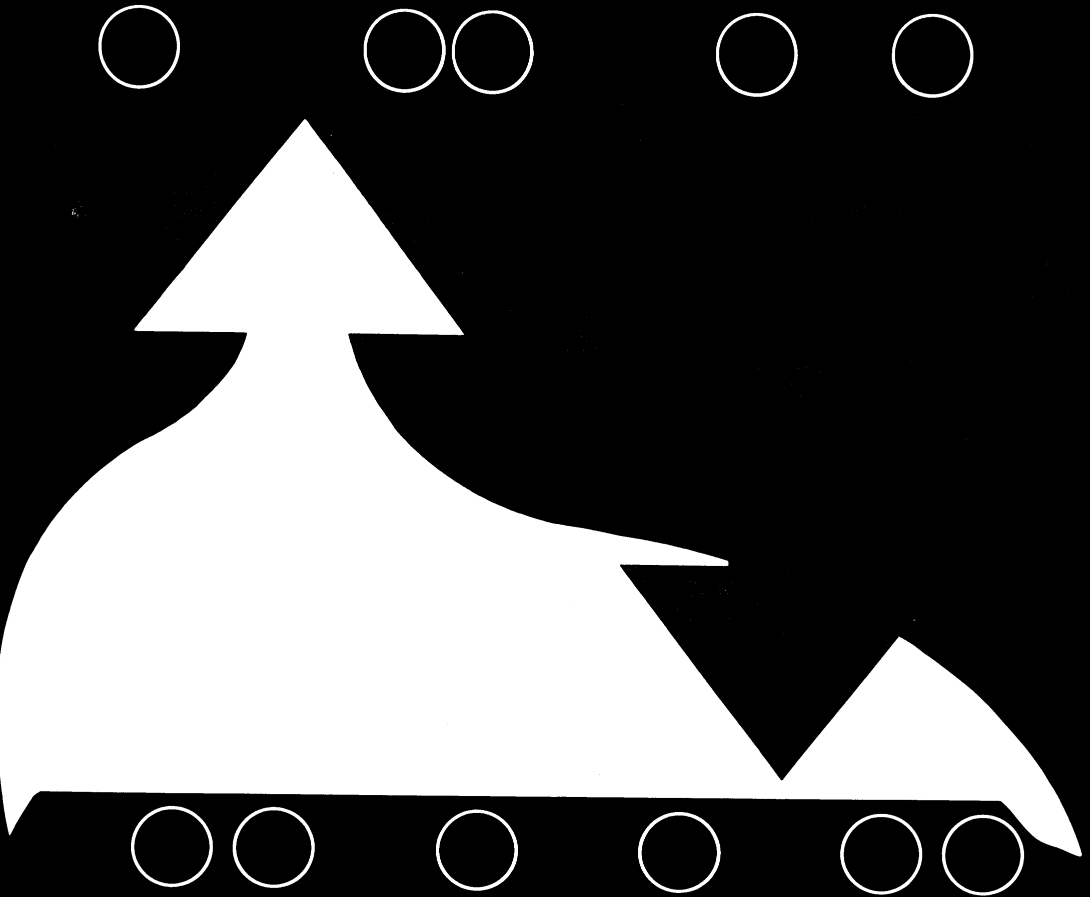
Proceedings of the International Joint Conference on Neural Networks

January, 1990

VOLUME
1

IJCNN-90-WASH-DC

Washington, D.C.
January 15-19, 1990



INNS
INTERNATIONAL
NEURAL NETWORK
SOCIETY

THE INSTITUTE OF
ELECTRICAL AND
ELECTRONICS
ENGINEERS, INC.

Editor
Maureen Caudill

IJCNN-90-WASH DC

International Joint Conference on Neural Networks

January 15-19, 1990
Omni Shoreham Hotel
Washington, DC

Volume I
Theory Track
Neural and Cognitive Sciences Track

---

co-sponsored by the
International Neural Network Society
and the
Institute of Electrical and Electronics Engineers, Inc.

LEA LAWRENCE ERLBAUM ASSOCIATES, PUBLISHERS
Hillsdale, New Jersey Hove and London

Lawrence Erlbaum Associates, Inc., Publishers
365 Broadway
Hillsdale, New Jersey 07642

ISBN 0-8058-0775-6 (Vol. 1)
ISBN 0-8058-0776-4 (Vol. 2)
ISBN 0-8058-0754-3 (Set)

PRINTED IN THE UNITED STATES OF AMERICA
10 9 8 7 6 5 4 3 2 1

Organizing Committee

A Special Note of Appreciation

The INNS would like to extend special thanks to

Harriet Caudill

who provided extraordinary support for this meeting,
and invaluable assistance in preparing these proceedings.

Technical Program Committee

Co-Chairs:

James Anderson *Brown University* & **Andras Pellionisz** *New York University Medical Center*

Committee Members:

Shun-ichi Amari *Tokyo University*
Pierre Baldi *Jet Propulsion Laboratory*
Jacob Barhen *Jet Propulsion Laboratory*
Mark Bear *Center for Neural Sciences*
Tom Brown *Yale University Medical School*
Daniel Bullock *Boston University*
Heinrich Bulthoff *Brown University*
Charles Butler *Physical Sciences Inc.*
Gail Carpenter *Boston University*
John Caulfield *University of Alabama in Huntsville*
Michael Cohen *Boston University*
Leon Cooper *Brown University*
Claude Cruz *Plexus Systems*
John Daugman *Harvard University*
Rolf Eckmiller *Heinrich-Heine-Universitat Dusseldorf*
Gerald Edelman *The Rockefeller University*
Federico Faggin *Synaptics*
Walter Freeman *University of California, Berkeley*
Kunihiko Fukushima *Osaka Univeristy*
Apostolos Georgopoulos *Johns Hopkins University Medical. School*
Lee Giles *NEC Laboratories*
Jefim Goldberg *Baylor College of Medicine*
Hans Graf *AT&T Bell Labs*
Stephen Grossberg *Boston University*
Dan Hammerstrom *Oregon Graduate Center*
Robert Hecht-Nielsen *Hecht-Nielsen Neurocomputer Corp.*
Geoffrey Hinton *University of Toronto*
Morris Hirsch *University of California, Berkeley*
James Houk *Northwestern University Medical School*
David Hubel *Harvard University. Medical School*
Harry Jerison *University of California, Los Angeles*
Ken Johnson *Hughes Aircraft Company*
Candace Kamm *Bellcore*
Teuvo Kohonen *Helsinki Unversity of Technology*
Tim Kraft *Science Applications International Corporation*
Tom Landauer *Bell Commications. Research*
Daniel Levine *University of Texas at Arlington*
Ralph Linsker *IBM Watson Research Center*
James McClelland *Carnegie-Mellon University*
Carver Mead *California Institute of Technology*
Ennio Mingolla *Boston University*
Sei Miyake *ATR Auditory & Visual Perception Labs*
Paul Mueller *University of Pennsylvania*
Mussa-Ivaldi *Massachusetts Institute of Technology*
Yoh-han Pao *Case-Western Reserve University*
Andrew Penz *Texas Instruments*
Barry Peterson *Northwestern University Medical School*
Demetri Psaltis *California Institute of Technology*
Doug Reilly *Nestor, Inc.*
David Rumelhart *Stanford University*
Tariq Samad *Honeywell*
Eric Schwartz *New York University Medical Center*
Allen Selverston *University of California, San Diego*
Bernard Soffer *Hughes Research Laboratories*
George Sperling *New York University*
David Stork *Stanford University*
Harold Szu *Naval Research Laboratory*
David van Essen *California Institute of Technology*
Christoph von der Malsburg *University of Southern California*
Alex Weibel *ATR*
Fred Weingard *Booz-Allen & Hamilton*
Bernard Widrow *Stanford University*
George Works *Science Applications International Corporation*

Conference Sponsors

The International Neural Network Society and the IEEE Technical Activities Board Neural Network Committee are the co-sponsors of the International Joint Conference on Neural Networks, 1990, Washington, DC.

International Neural Network Society (INNS)

VOLUME I

Table of Contents, Volume I

Table of Contents, Volume I

Table of Contents, Volume I

Table of Contents, Volume I

Table of Contents, Volume I

Table of Contents, Volume I

Table of Contents, Volume II

Table of Contents, Volume II

Table of Contents, Volume II

Table of Contents, Volume II

Table of Contents, Volume II

Table of Contents, Volume II

Table of Contents, Volume II

Neural and Cognitive Sciences

Multidirectional Associative Memory

MASAFUMI HAGIWARA

Dept. of Elec. Eng.
Facul. of Sci. and Tech.
Keio University,
3-14-1 Hiyoshi, Kohoku-ku, Yokohama
223 Japan

Abstract Multidirectional Associative Memory (MAM) is proposed and simulated.It enables multiple associations such as $(A_i,B_i,C_i,\cdots)$. Its structure is very simple, and is devised to satisfy many facts based on psychology. The proposed MAM has the following features.
1) By recollecting one data, many other data can be recalled (multiple association).
2) When an interference from other layer exists, it can be suppressed by recollecting other data (noise suppression effect).
3) Recollection becomes correct by memorization with other data (robust memory).

1.Introduction

It is widely accepted that human memory is characterized by its associative, distributed, and parallel processing capabilities. With this notion, many kinds of memory models have been proposed [1]-[6]. Among them, a bidirectional associative memory (BAM) proposed by B.Kosko has attracted much attention because of its simple structure and its similarity to brain (for example, there exists reverbelation in the BAM) [5]. The BAM agrees with the fact that a neural projection from one region to another is usually accompanied by a backward projection [6]. The BAM is the minimal two-layer nonlinear feedback network and produces two-way associative search for stored stimulus-response associations (A_i,B_i). However, it is similar to the conventional ones in respect of the number of vectors stored simultaneously: multiple associations such as $(A_i,B_i,C_i,\cdots)$ are impossible.

It is well known that the following strategies are effective in order to ensure memorization.
1) Deep understanding of the meaning of the data to be stored.
2) Understanding of the mutual relation of the data to be stored.
In addition, it is noticeable that the more the data for retrieval exist, the more correctly the recollection becomes.

In this paper, a new memory model called a Multidirectional Associative Memory (MAM) is proposed and simulated. It enables multiple associations such as $(A_i,B_i,C_i,\cdots)$. The structure of the MAM is very simple, and is devised to satisfy above mentioned facts based on psychology. Therefore, the proposed MAM has the following features.
1) By recollecting one data, many other data can be recalled (multiple association).
2) When an interference from other layer exists, it can be suppressed by recollecting other data. (noise suppression effect).
3) Recollection becomes correct by memorization with other data (robust memory).

Following this introduction, the proposed MAM is explained in Sec.2. In Sec.3, a demonstration is shown by computer simulation.

2. Structure of MAM

In this section, structure of the multidirectional associative memory (MAM) is explained using a three-layer type MAM as an example.

2.1 Behavior of MAM

Since the MAM can be considered as a multilayer type bidirectional associative memory (BAM), it has many features in common with the BAM [5].

The MAM is the minimal multi-layer nonlinear feedback network. Information pass forward from one layer to others by passing through the connection matrices P,Q,R,···. Information pass backward through these matrices transpose P^T, Q^T, R^T,···.

When the neurons are activated, the network quickly evolves to a stable state of multi-pattern reverberation, or resonance. The stable reverberation corresponds to a system energy local minimum.

Fig.1 shows a structure of a multidirectional associative memory (MAM) when the number of the layers is three. The behavior of a MAM is explained using Fig.1 (three-layer MAM). It produces a sequence of grouped approximations to the stored group (A_i,B_i,C_i): (A,B,C), (A',B',C'), (A'',B'',C''), (A''',B''',C'''),···. Ideally, this sequence will quickly converge to some fixed group (A_f,B_f,C_f), and this fixed group will be (A_i,B_i,C_i) or nearly so.

A MAM behaves as a heteroassociative content addressable memory (CAM) if it is represented by a recollection such as shown in Fig.2.

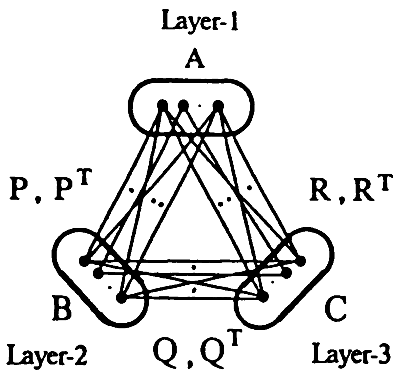


Fig.1 Structure of multidirectional associative memory (three-layer type).

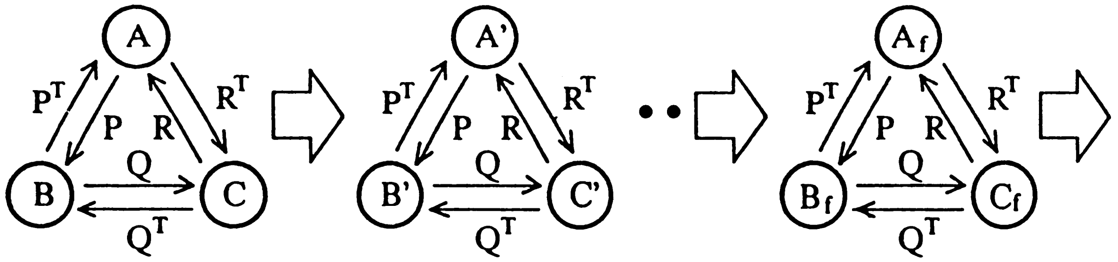


Fig.2. Recollection example of 3-layer MAM.

2.2 MAM encoding

Here, the encoding of a MAM is explained using the three-layer MAM.

1) Let (A_1,B_1,C_1),···(A_m,B_m,C_m) be the bipolar date groups to be stored.
2) To make the matrix from the layer A to the layer B, convert bipolar pairs into bipolar correlation matrices $A_i^T B_i$.
3) Add up the bipolar correlation matrices,

$$P = \sum_i A_i^T B_i \tag{1}$$

Matrix P stores the m associations: (A_1,B_1),···(A_m,B_m).

4) To make the matrix backward direction of Eq.(1) (from layer-1 to layer-2), a matrix transpose is made

$$P^T = \sum_i B_i^T A_i \tag{2}$$

5) In the same way of 2)-4), all the matrices Q(layer-2→layer-3), Q^T(layer-3→layer-2), R(layer-3→layer-1), and R^T (layer-1→layer-3) are made.

2.3 MAM decoding

The MAM recall procedure is a nonlinear feedback procedure. Each neuron a_i, b_i, and c_i independently and synchronously examines its input sum from the neurons of other layers (see Fig.2), then changes state or not. The input sum to a_i, b_i, and c_i are

$$B(P^T)^i + CR^i = \sum_j (b_j p_{ij} + c_j r_{ji}) \quad (3)$$

$$C(Q^T)^i + AP^i = \sum_j (c_j q_{ij} + a_j p_{ji}) \quad (4)$$

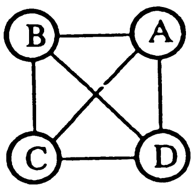
$$A(R^T)^i + BQ^i = \sum_j (a_j r_{ij} + b_j q_{ji}) \quad (5)$$

where X^i means the *i*-th column of the matrix X, and x_{ij} is an element of X. We take 0 as the threshold for all neurons. In summary, the threshold functions for a_i, b_i, and c_i are

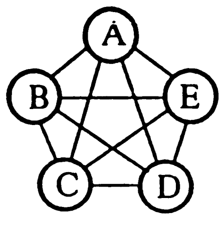
$$a_i = \begin{cases} 1: & \text{if } B(P^T)^i + CR^i \geq 0 \\ -1: & \text{if } B(P^T)^i + CR^i < 0 \end{cases} \quad (6)$$

$$b_i = \begin{cases} 1: & \text{if } C(Q^T)^i + AP^i \geq 0 \\ -1: & \text{if } C(Q^T)^i + AP^i < 0 \end{cases} \quad (7)$$

$$c_i = \begin{cases} 1: & \text{if } A(R^T)^i + BQ^i \geq 0 \\ -1: & \text{if } A(R^T)^i + BQ^i < 0 \end{cases} \quad (8)$$



(a) 4-layer MAM.



(b) 5-layer MAM.

Fig.3. Various kinds of MAMs.

The neurons continue their synchronous state changes until a multidirectionally stable state (A_f, B_f, C_f) is reached.

So far the three-layer MAM is explained. It should be state that there exist many kinds of MAMs such as shown in Fig.3.

3. Simulation results

In this section, we show the computer simulation results to demonstrate the effectiveness of the proposed MAM.

The following simulation conditions are used.

1) The spatial character associations (A,a,1) (B,b,2), and (C,c,3) are stored for the MAM, and (A,a) (B,b), and (C,c) are stored for the BAM.
2) The layer-1 contains 6×8=48 neurons (MAM and BAM), the layer-2 contains 5×7=35 neurons (MAM and BAM), and the layer-3 contains 4×6=24 neurons (MAM).

Table 1 Simulation result.

			BAM		MAM			
Layer-1	Number of reversed bits	0 : 19	○	○	○	○	○	○
		20 : 27	×	×	×	○	○	○
		28 : 48	△	△	△	○	○	○
Layer-2			—	Noise free input	—	Noise free input	—	Noise free input
Layer-3					—	—	Noise free input	Noise free input

○: correct recollection ×: false recollection △: (reversed) correct recollection

Table 1 shows the simulation result. The pattern for the digit "A" to the layer-1 was corrupted by randomly reversing each bit from 0 to 48 (all bits are reversed). As for the BAM, when the number of reversed bits exceeds 19, correct recollection can not be done even if a noise free correct initial input is added to another layer. On the other hand, as for the proposed MAM, when one noise free correct initial input is added to another layer, recollection becomes perfect irrespective of the corruption of the layer-1.

Fig.4 displays snapshots of synchronous MAM and BAM recall. In this figure, the steady state comes only at the second steps. The pattern for the digit "A" to the layer-1 was corrupted by randomly reversing each bit with a probability of 44%. When the input is restricted to the layer-1 only, the BAM cannot recall correctly. However, with additional input to the layer-3, the MAM can recall perfectly because of multiple association effect of the MAM.

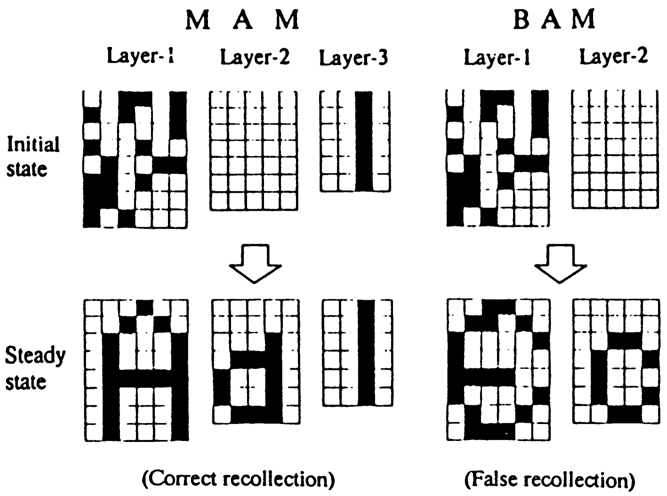


Fig.4. Synchronous MAM and BAM recall. (The digit "A" was corrupted by randomly reversing with a probability 0.44.)

Therefore, it can be said that the multiple association enables a supreme noise suppression and robustness of memory.

4. Conclusions

Multidirectional Associative Memory (MAM) has been proposed and simulated. It enables multiple associations such as $(A_i,B_i,C_i,\cdots)$. Its structure is very simple, and is devised to satisfy many facts based on psychology. The following features of the proposed MAM have been confirmed.

1) By recollecting one data, many other data can be recalled (multiple association).
2) When an interference from other layer exists, it can be suppressed by recollecting other data (noise suppression effect).
3) Recollection becomes correct by memorization with many data (robust memory).

ACKNOWLEDGEMENT

The author would like to thank Prof. Masao Nakagawa of Keio university.

REFERENCES

[1] K.Nakano :"Associatron - a model of associative memory", IEEE Trans. Syst. Man. Cybern., vol.SMC-2, pp.380-388, 1972.
[2] Y.Hirai: "A model of human associative processor (HASP)", IEEE Trans. Syst. Man. Cybern., vol.SMC-13, no.5, pp.851-857, Sept./Oct. 1983.
[3] Y.Hirai: "Mutually linked HASP's; a solution for constraint-satisfaction problems by associative processing", IEEE Trans. Syst. Man. Cybern., vol.SMC-15, no.3, pp.432-442, May/June 1985.
[4] T.Kohonen, Self-organization and associative memory, Berlin: Springer-Verlag, 1984.
[5] B.Kosko: "Bidirectional associative memories", IEEE Trans. Syst. Man. Cybern., vol.SMC-18, no.1, pp.49-60, Jan./Feb. 1988.
[6] K.Okajima, S.Tanaka and S.Fujiwara: "A heteroassociative memory network with feedback connection", Proc. IEEE Int. Conf. Neural Networks, II 711-718, 1987.

Maximum Entropy Prediction in Neural Networks

William B Levy, Ph.D.
Department of Neurological Surgery
University of Virginia School of Medicine
Charlottesville, Virginia 22908

Neural networks can generate predictive representations. Particularly interesting are the networks that produce a predictive representation that is a vector of probabilities associated with a future representation. These predictive representations are interesting because, among other reasons, such vectors imply the predictive representation of maximum likelihood.

Such a vector of probabilities, called a type II predictive representation to distinguish it from a maximum likelihood predictive representation, consists of elements each of which is the probability of one particular neuron firing. More specifically, each neuron produces the probability of its own future state. In our investigations each neuron generates this probability by a local computation that uses maximum entropy (M.E.) inference, stored averages, and Bayes's equation.

The motivation for such investigations[8] comes from the importance of prediction in the life of an animal[3,17], from various observations that point out how synaptic modification can lead to the encoding of a statistic[1,4,14], and from the existence of a unique, optimal procedure to produce probabilities based on statistics, i.e., M.E. inference [6,15].

In order for a neural network to produce this type II predictive representation, there is a small set of absolute axioms and requirements: a definition of the prediction problem, the requirements of M.E. inference, and complexity considerations. In turn the implementation of these requirements leads to a set of network characteristics as natural outcomes of the usual, classic characteristics of neurons and synapses. The purpose of this communication is to point out the implications these requirements have for neural networks which mediate predictive representations.

THE PREDICTION PROBLEM

It is first necessary to define the type of prediction we are studying. We want a network to generate predictions which are useful (i.e. usable by another network or the organism itself); moreover, we want a network to base such predictions on appropriate correlational information which is adaptively encoded at neurons and synapses.

There are three requirements of a predictive representation. (To emphasize that a prediction is also a representation, we call it a predictive representation as distinct from a "standard representation.") The first two requirements stem from our interest only in predictions which will be usable.

(1) Timeliness: A predictive representation must precede in time the standard representation being predicted (because the whole point of creating a prediction is to improve some outcome in the future).

(2) Meaningfulness: A predictive representation space must map into the standard representation space being predicted about. This requirement is necessary if a prediction is to be used for the benefit (survival, propagation) of the organism containing this network. A specific example might make the motivation for this requirement clearer. Consider a prediction generating network which is embedded in a larger network that alters W in the external world to suit some homeostatic purpose. This larger network needs to relate, or map, a predictive representation onto the standard representation of W in order to use the prediction to control W in a sensible way that anticipates homeostatic needs. Without this map, the homeostatic part of the network would be unable to take advantage of the predictions being generated.

The third requirement stems from our desire that the network encode and use appropriate associative (correlational) information.

(3) Aptness: Appropriate correlations must be used to generate predictions. Among the many constraints on a neural network there is the local principle which sensibly limits the information available to a neuron; e.g., of all the synaptic weights and neuronal activities in a network, only those which are inputs to a neuron are local to that neuron. Then suppose, because of the local principle and extant circuitry, a neuron can only learn an association between a representation in the space A and another representation in the space B where any one A representation precedes any one B representation. Then it is appropriate for this neuron to use an A representation to predict about B. However, it is not appropriate for the neuron to use a B representation to predict about A in the future or to use an A representation to predict about some nonlocal event C (except in

the sense that C can be represented as B). Furthermore, even though the local principle requires convergence of an input to a cell such convergence is not enough for aptness. Just because the inputs to a neuron are, by definition, local to that neuron, does not imply that it is appropriate for such a neuron to associate just any set of these inputs. Specifically, a neuron would be living in a fantasy world of inappropriate correlational encodings if it were to use its own predictive representation rather than a standard representation for synaptic reinforcement.

NETWORK CHARACTERISTICS DERIVATIVE OF THIS DEFINITION

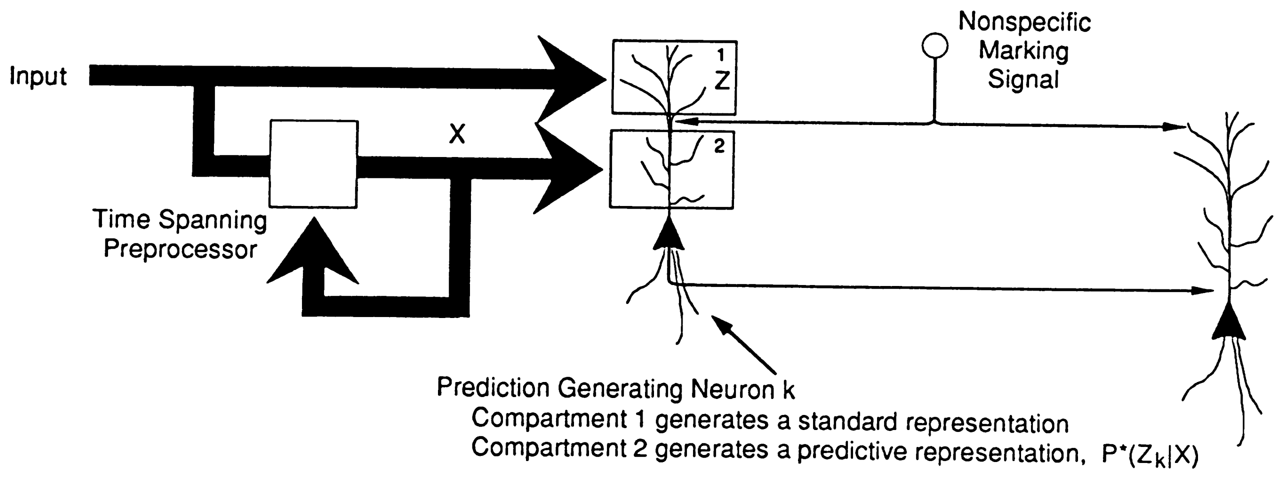
The requirement for a timely prediction is, in part, implemented by the temporal characteristics of associative modification rules[5, 12]. These characteristics allow time spanning associations between two standard representations. However, the allowable time span of these rules is rather short, certainly less than 125 ms. This span is hardly large enough for most practical purposes requiring prediction. An arbitrarily large time gap, however, can be spanned by using feedback circuitry for preprocessing the representation that will be used as the conditioning variable in prediction generation[8].

Requirements (2) and (3) produce the need for a bidirectional mapping between the standard representation space and the corresponding predictive representation space. The obvious implementation of this bidirectional mapping is that the same neuron mediates both a standard representation and its own predictive representation. There are advantages to such a mapping including simplicity, immediate interpretability, and constancy over changes in circuitry due to synaptic modification. However, although this mapping prevents confusion as to the applicable space of a prediction, it does not prevent, rather it can cause, confusion between the two types of representations themselves.

Such a dual representational implementation within a single neuron leads to the possibility that the two different types of representations will be confused. Such confusion could happen in two places. One place is at the associatively modifiable synapses of the prediction generating neuron. Associative synaptic modification could be reinforced by the unconfirmed state of a predictive representation instead of the reality of a standard representation. The other place of possible confusion is at the postsynaptic cells receiving inputs from such a prediction generating neuron. These postsynaptic neurons must distinguish which input signals are predictive representations and which are standard representations.

The distinction which solves the first problem can be accomplished by a nonspecific, low dimension, marking signal, a two-compartment neuron, and non-simultaneity of the two types of representations. The second problem needs only a marking signal and the non-simultaneity of the two representations to produce the necessary distinction. Such marking signals would alter the interaction of postsynaptic cells according to the marking signal's temporal relationship with one of the two types of representations.

Combined with the network characteristics just noted, aptness is satisfied by an associative modification rule in which reinforcement of synaptic modification is a rectified affair. Consider the event being predicted as the voltage of compartment 1, Z, in the figure. Then Z in compartment 1 can reinforce compartment 2 synaptic modification for a state of each input X_i in compartment 2 that precedes the state Z but cannot reinforce modification of states which occur after state Z in time. Furthermore, the prediction generating compartment (2 in the figure) cannot reinforce synapses in compartment 1.



Input
X
Time Spanning Preprocessor
1
Z
2
Nonspecific Marking Signal
Prediction Generating Neuron k
Compartment 1 generates a standard representation
Compartment 2 generates a predictive representation, $P^*(Z_k|X)$

Although the definitional requirements constrain the characteristics of suitable networks, our implementations are probably not unique in satisfying these requirements because so many different types of neurons exist. Still, the suggested network characteristics seem rather natural, straightforward constructions, particularly when considered in the context provided by the hippocampus of the mammalian brain [8].

Let us now consider computational constraints which limit the characteristics of a neural network. For simplicity of exposition, we consider a standard representation space in which each neuron takes on a state in the set {0,1} and a predictive representation space in which each neuron takes on some monotonic function of a probability; for expository purposes, let it take on values in [0,1] as the probability itself.

COMPUTATIONAL CONSIDERATIONS

In an idealized situation each prediction generating neuron, k, would, at time t, generate $P(Z_k(t+n)=1 \mid X(t)=x)$, the conditional probability that k will be in the one state at some n steps into the future given that its inputs, the vector variable X, is in state x now at time t.

Because the dimension of X is quite large, there will be many configurations X that have never been experienced before the prediction generation time t. Moreover, even with sufficient sampling, it is impossible to store the exponentially many statistics, e.g. the expectation $E[Z_k = 1 \mid X = x]$, that might be needed. The solution to this problem is to use Bayes's equation and M.E. inference on low-order moments. In accord with Bayes's equation and leaving implicit the time notations mentioned above, each neuron would compute

$$P^*(Z_k=1 \mid X=x) = P^*(X=x \mid Z_k=1) \cdot P^*(Z_k=1) / P^*(X=x) \qquad (1)$$

where, on the right hand side, $P^*(\,)$ is a M.E. inferred probability distribution computed from sample averages that have nearly converged to the population based expectation. A M.E. inferred probability based on the lowest order moments of interest is:

$$P^*(X=x \mid Z_k=1) = \prod_i P^*(X_i \mid Z_k=1) = \prod_i \bar{P}(X_i=1 \mid Z_k=1)^{x_i} \cdot (1-\bar{P}(X_i=1 \mid Z_k=1))^{(1-x_i)} \qquad (2)$$

$$= \exp\{\sum_i x_i \cdot \log \bar{P}(X_i=1 \mid Z_k=1) + (1-x_i) \cdot \log(1-\bar{P}(X_i=1 \mid Z_k=1))\} \qquad (3)$$

where the $\bar{P}(\,)$ are sample based averages.

As in many neural networks, synapses would store statistics, e.g. $(\bar{P}(X_i \mid Z_k=1))$, and the readout of the appropriate statistics is just a natural result of the signal flow, x_i. The "learning" of such statistics could result from synaptic modification rules similar to those known to exist in the brain[7,9,10,11].

The existence of synaptic modification rules that encode averages provided much of the impetus for the approach described here. Equation (1) and computations like equation (2) imply the need for three different types of averages: $\bar{P}(X_i=1 \mid Z_k=1)$, $\bar{P}(Z_k=1)$, and $\bar{P}(X_i=1 \mid Z_k=0)$. Note that $\bar{P}(Z_k)$ requires a neuron to encode its own average activity or to guarantee a preset value. Note also that a synaptic encoding of $\bar{P}(X_i=1 \mid Z_k=1)/\bar{P}(X_i=1)$ might substitute for the pair $(\bar{P}(X_i=1 \mid Z_k=0), \bar{P}(X_i=1 \mid Z_k=1))$. In either case a neuron has the information to calculate the denominator of equation (1). Interestingly there is recent evidence for the existence of a synaptic modification rule that would encode $\bar{P}(X_i \mid Z_k=0)$ or the alternate statistic[9,16].

Regardless of the statistics in the constraint set (in our case the low order statistical correlations), M.E. inference will always use what is essentially a multiplicative form[2]. This requirement leads to an obvious implementation in a neural network: add logarithms and exponentiate. Because synaptic currents vary with the logarithm of their conductance and because depolarization translates into cell firing in a nonlinear way, such hypothetical characteristics are plausible. (Of course a maximum likelihood predictive representation does not even require exponentiation before its formation from a monotonic function of a type II predictive representation formed with logarithms.)

ANOTHER COMPLEXITY CONSTRAINT

M.E. inference can make use of moment constraints, which are essentially correlations, of any order. For a postsynaptic cell k and inputs X_i, X_j, X_h, a lowest order constraint of interest would be $\bar{P}(X_i=1 \mid Z_k=1)$ and examples of higher order constraints are $\bar{P}(X_i \cdot X_h=1 \mid Z_k=1)$ and $\bar{P}(X_i \cdot X_j \cdot X_h=1 \mid Z_k=1)$.

Unfortunately computational complexity issues often make it impossible, in practice, to compute a M.E. derived probability from a constraint set of arbitrary moments of arbitrary order. A computationally intractable problem can arise if the constraint set does not consist solely of lowest order correlates because there might be overlap among two constraints in the set. When such overlap exists, there can be a need for an exponential number of variables to calculate the M.E. inferred probability distribution. This exponential requirement renders the M.E. method computationally intractable in such cases.

Example of a set of nonoverlapping moment constraints:

$\{\bar{P}(X_1=1 \mid Z_k=1), \bar{P}(X_2=1 \mid Z_k=1), \bar{P}(X_3 \cdot X_4=1 \mid Z_k=1)\}$.

Example of a set with overlapping constraints:

$\{\bar{P}(X_1=1 \mid Z_k=1), \bar{P}(X_1 \cdot X_4=1 \mid Z_k=1)\}$.

On the other hand, intractability will always be avoided if the constraints in a set do not overlap. That is, if a conditioned variable X_i (i.e. the activity of input line i) appears in no more than one moment constraint in a set of constraints, then there will be no overlap, and this particular complexity problem is avoided.

We view this complexity problem and its solution as a constraint affecting the characteristic computation of a prediction generating network. More exactly the affect is on the preprocessor computation that produces the X inputs of the prediction generating neurons. It would be useful for the prediction generating neurons to receive their inputs from a network preprocessor that moves information out of higher-order moments into lower-order moments and that avoids overlapping constraints with high probability. Moreover, it seems possible that this preprocessor and the time spanning preprocessor are identical[8].

Full connectivity from the X space to a prediction generating neuron is not a requirement because M.E. inference remains consistent even with missing moment constraints.

Thus the requirements stemming from this particular computation, equations (1) and (2) or (3), imply several network characteristics. These requirements include: a multiplicative combination of probabilities; computation of M.E. inference as if working with lowest order moments; and a requirement for averages over three different kinds of distributions. The combination of these computational requirements and the definitional requirements create an important set of restrictions on the class of acceptable neural networks that create predictions.

This research and WBL are supported in part by the NIH (NS14588)and by an NIMH RSDA (MH00622). The help of D. Adelsberger-Mangan, C. M. Colbert, and N. L Desmond is greatly appreciated.

References

1. Amari, S.-I. (1977) Biol. Cybern., 26, 175-185.
2. Csiszä r, I. (1975) Ann. Prob. 3, 146-158.
3. Dawkins, R. (1976) The selfish gene. New York: Oxford University Press.
4. Geman, S. (1981) SIAM AMS Proc., 13, 91-105.
5. Gustaffson, B., Wigströ m, H., Abraham, W. C., and Huang, Y.-Y. (1985) J. Neurosci. 7, 774-780.
6. Jaynes, E. T. (1978) In R. D. Levine & M. Tribus, Eds., The maximum entropy formalism. 15-118. Cambridge: MIT Press.
7. Levy, W. B (1982) Proc. Fourth Annual Conference of Cognitive Science Society, 135-136.
8. Levy, W. B (in press) In: R. D. Hawkins and G. H. Bower, Eds., Computational models of learning in simple neuronal systems. New York: Academic Press.
9. Levy, W. B, Colbert, C. M. and Desmond, N. L (1989) In: M. A. Gluck and D. E. Rumelhart, Eds., Neuroscience and connectionist models. Hillsdale, NJ: Lawrence Erlbaum Assoc., Inc.
10. Levy, W. B and Desmond, N. L (1985) In G. Buzsaki and C. H. Vanderwolf, Eds., Electrical activity of the archicortex. Budapest, Hungary: Akademiai Kiado, 359-373.
11. Levy, W. B and Steward, O. (1979) Brain Res. 175, 233-245.
12. Levy, W. B and Steward, O. (1983) Neurosci. 8, 791-797.
13. Lorente de Nó, R. (1938) J. Neurophysiol. 1, 207-244.
14. Rosenblatt, F. (1962) Principles of neurodynamics. Washington, DC: Spartan Books.
15. Shore, J. E., & Johnson, R. W. (1980) IEEE Trans. Information Theory, IT-26, 26-37.
16. Stanton, P. K. and Sejnowski, T. J. (1989) Nature 339, 215-218.
17. Young, J. Z., Ed. (1970) The life of mammals. Oxford: Clarendon Press.

NEURAL DYNAMICS OF MOTION SEGMENTATION: DIRECTION FIELDS, APERTURES, AND RESONANT GROUPING

Stephen Grossberg and Ennio Mingolla
Center for Adaptive Systems, Boston University
111 Cummington Street, Boston, MA 02215

A neural network model of motion segmentation by visual cortex is described. The model clarifies how preprocessing of motion signals by a motion OC Filter is joined to long-range cooperative motion mechanisms in a motion CC Loop to control phenomena such as induced motion, motion capture, and motion after effects. The total model system is a motion Boundary Contour System (BCS) that is computed in parallel with the static BCS of Grossberg and Mingolla before both systems cooperate to generate a boundary representation for 3-D visual form perception. The present investigations clarify how the BCS used in static segmentation problems can be modified for use in motion segmentation problems, notably for analysing how ambiguous local movements (the aperture problem) on a complex moving shape are suppressed and actively reorganized into a coherent global motion signal. Unlike many previous approaches, we analyse how a coherent motion signal is imparted to all regions of a moving figure ("motion capture"), not only regions at which unambiguous motion signals exist.

Why Are Static and Motion Boundary Contour Systems Needed?

Some regions of visual cortex are specialized for motion processing, notably region MT (Albright, Desimone, & Gross, 1984; Maunsell & van Essen, 1983; Newsome, Gizzi, & Movshon, 1983; Zeki, 1974a, 1974b). However, even the earliest stages of visual cortex processing, such as simple cells in V1, require stimuli that change through time for their maximal activation and are direction-sensitive (DeValois, Albrecht, & Thorell, 1982; Heggelund, 1981; Hubel & Wiesel, 1962, 1968, 1977; Tanaka, Lee, & Creutzfeldt, 1983). Why has evolution generated regions such as MT, when even V1 is change-sensitive and direction-sensitive? What computational properties are achieved by MT that are not already available in V1?

The monocular Boundary Contour System, or BCS, theory of Grossberg and Mingolla (1985a, 1985b), and its binocular generalization (Grossberg, 1987b; Grossberg & Marshall, 1989), has successfully modelled many boundary segmentation properties of V1 and its prestriate projections. (See Grossberg (1987c, 1988a) for collections of these and related articles.) The BCS was there used to analyse data generated in response to static visual images. Henceforth we therefore call such a BCS a static BCS model. Nonetheless, its model cells can easily be gated by cells sensitive to image transients, such as Y cells (Enroth-Cugell & Robson, 1966; Hoffmann, 1973; Sekuler, 1975; Stone, 1972; Stone & Dreher, 1973; Tolhurst, 1973), to generate receptive fields sensitive to image transients. How does a motion BCS differ from a static BCS whose cells are sensitive to image transients?

Joining Sensitivity to Direction-of-Motion with Insensitivity to Direction-of-Contrast

The static BCS consists of two major subdivisions: an oriented contrast-sensitive filter, called the OC Filter, and a cooperative-competitive feedback network, called the CC Loop. The OC Filter models the simple cells and complex cells of V1. Its projections to hypercomplex cells form the interface of the OC Filter with the CC Loop. The hypercomplex cells project, in turn, to a cell type called cooperative bipole cells by Grossberg and Mingolla. The bipole cells interact with the hypercomplex cells via the CC Loop.

This research was supported in part by the Air Force Office of Scientific Research (AFOSR F49620-86-C-0037 and AFOSR F49620-87-C-0018), the Army Research Office (ARO DAAL03-88-K-0088).

The OC Filter is a nonlinear filter that multiplexes several different types of image information into a spatially organized representation, or map, across the network of hypercomplex cells. Such a map functions as a compressed code that is capable of reacting selectively to prescribed combinations of image features.

The CC Loop reacts to this multiplexed spatial map by transforming and amplifying those spatial combinations of cell activations whose coded information is mutually consistent, and actively suppressing the rest. The result combines information about image edges, texture, shading, depth, and spatial scale into a resonant boundary representation. Eckhorn *et al.* (1988), Gray *et al.* (1989) and Peterhans and von der Heydt (1989) have recently reported neurophysiological data that support predicted cooperative and resonant properties of the CC Loop.

Although the simple cells of the BCS are sensitive to direction-of-contrast, or contrast polarity, the complex cells of the BCS are rendered insensitive to direction-of-contrast by receiving inputs from pairs of simple cells with opposite direction-of-contrast. Such a property is also true of the simple cells and complex cells in area V1 (DeValois, Albrecht, & Thorell, 1982; Poggio, Motter, Squatrito, & Trotter, 1985; Thorell, DeValois, & Albrecht, 1984).

This property is useful for extracting boundary structure that is independent of illumination fluctuations, such as shadows. As a result, the output of the static OC Filter is unable to differentiate direction-of-motion. A key property of the motion BCS model presented here is that it possesses a modified OC Filter that multiplexes the property of insensitivity to direction-of-contrast, which is equally useful for the processing of static and moving forms, with sensitivity to direction-of-motion (Grossberg, 1987a). The properties of this motion OC Filter (Figure 1) clarify many properties of motion perception. Grossberg and Rudd (1989a, 1989b) have, for example, used the motion OC Filter to explain properties of apparent motion. When the motion OC Filter is connected to the CC Loop, a much larger body of data, including coherent global motion percepts such as induced motion and motion capture, can also be analysed.

For example, a consideration of the action of Level 4 cells in the OC Filter indicates how ambiguous information about object motion direction can be combined into a motion direction signal near image corners. Consider the lower right corner of a light rectangle, moving diagonally up and to the right, on a dark background. Over time, triplets of level 2 cells of horizontal orientation, when gated by a "darkening" Level 3 cell, detect roughly vertical motion (along the bottom horizontal edge). Simultaneously, vertically oriented Level 2 cells, when gated by a (different) "brightening" Level 3 cells, detect roughly horizontal motion (along the leading vertical edge). These two motion signals (upward, rightward) are derived from direction-of-contrast- sensitive cells, but combine to form a total direction signal that is independent of contrast direction. This is accomplished by the Level 4 to Level 5 Gaussian filter, which computes a direction field from the oriented motion direction vectors over relatively large spatial distances and over a range of orientations.

A similar process occurs at the top-right corner of the same rectangle (though there the process is aided by the local filters at 45 degrees). Interaction of this direction field with a motion CC Loop can select the preferred directions near the figural corners and use these directions to complete the direction field along the entire leading edge of the figure. Such resonant completion uses cooperative-competitive feedback between motion hypercomplex cells fed by the motion OC Filter and the motion bipole cells to which they project.

Multiplexing of Motion Direction and Motion Depth

In the static BCS it has been shown how cells become binocular at the complex cell level (Grossberg, 1987b; Grossberg & Marshall, 1989). A similar hypothesis is made about the motion BCS; namely, that another role of the Gaussian filter is to combine motion signals from both eyes at the complex cells of Level 5 (Figure 1). As noted above, the Gaussian filter also provides an additional degree of freedom whereby cells at Level 5 can become sensitive to direction-of-motion over a wider range of stimulus orientations than cells at Level 2, whose preferred direction-of-motion is perpendicular to their preferred orientation. Likewise, many cells in MT are sensitive to direction-of-motion over a range of stimulus orientations, whereas cells in V1 typically are sensitive to the direction-of-motion perpendicular to their orientational preference (Albright, 1984; Albright, Desimone, & Gross, 1984; Maunsell & van Essen, 1983).

MOTION OC FILTER

Insensitive to Direction-of-Contrast

Sensitive to Direction-of-Motion

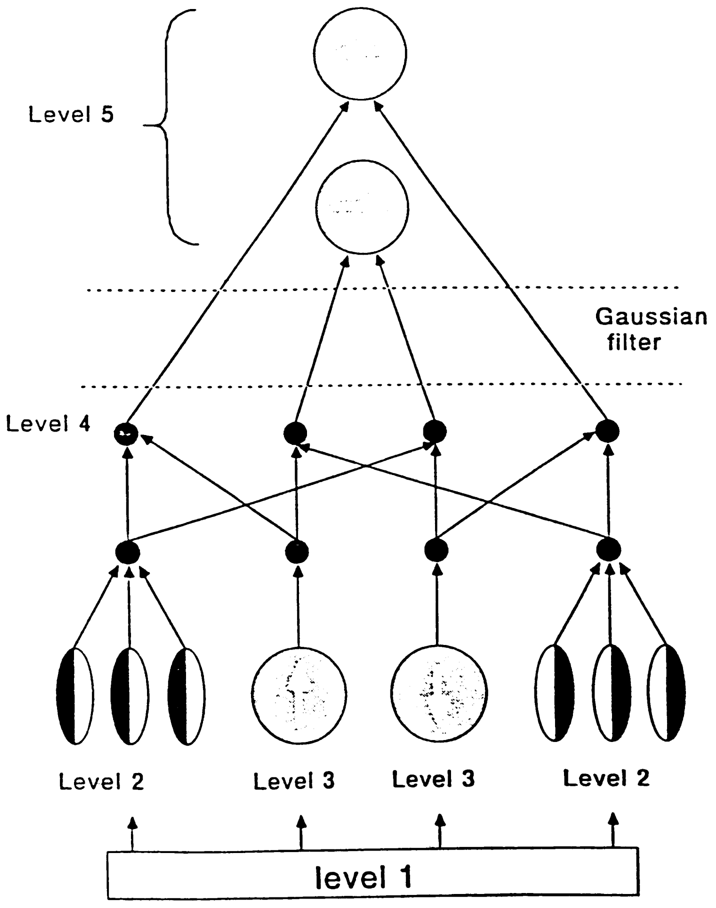


Figure 1. The motion OC Filter: Level 1 registers the input pattern. Level 2 consists of rectified and time-averaged signals from sustained response cells with oriented receptive fields that are sensitive to direction-of-contrast. Level 3 consists of rectified and time averaged signals from transient response cells with unoriented receptive fields that are sensitive to direction-of-change in the total cell input. Level 4 cells gate pairwise the sustained cell and transient cell signals to become sensitive to direction-of-motion and sensitive to direction-of-contrast. A long-range Gaussian filter combines outputs from Level 4 at Level 5. Level 5 cells employ signal-contrast enhancing competition and become sensitive to direction-of-motion and insensitive to direction-of-contrast.

References

Albright, T.D. (1984). *Journal of Neurophysiology*. **52**. 1106–1130.

Albright, T.D., Desimone, R., and Gross, C.G. (1984). *Journal of Neurophysiology*. **51**, 16–31.

DeValois, R.L., Albrecht, D.G., and Thorell, L.G. (1982). *Vision Research*, **22**, 545–559.

Eckhorn, R., Bauer, R., Jordan, W., Brosch, M., Kruse, W., Munk, M., and Reitboeck, H.J. (1988). *Biological Cybernetics*, **60**, 121-130.

Enroth-Cugell, C. and Robson, J.G. (1966). *Journal of Physiology*, **187**, 517–552.

Gray, C.M., Konig, P., Engel, A.K., and Singer, W. (1989). *Nature*, **338**, 334–337.

Grossberg, S. (1987a). *Perception and Psychophysics*, **41**, 87–116.

Grossberg, S. (1987b). *Perception and Psychophysics*, **41**, 117–158.

Grossberg, S. (Ed.) (1987c). **The Adaptive Brain, Vol. II: Vision, Speech, Language, and Motor Control.** Amsterdam: North-Holland.

Grossberg, S. (Ed.) (1988a). **Neural Networks and Natural Intelligence**. Cambridge, MA: MIT Press.

Grossberg, S. and Marshall, J. (1989). *Neural Networks*, **2**, 29–51.

Grossberg, S. and Mingolla, E. (1985a). *Psychological Review*, **92**, 173–211.

Grossberg, S. and Mingolla, E. (1985b). *Perception and Psychophysics*, **38**, 141–171.

Grossberg, S. and Rudd, M.E. (1989a). *Investigative Ophthalmology* Supplement, **30**, 73.

Grossberg, S. and Rudd, M.E. (1989b). *Proceedings of the International Joint Conference on Neural Networks*, June 19, 1989, Washington, DC.

Heggelund, P. (1981). *Experimental Brain Research*, **42**, 89–98.

Hoffman, K.-P. (1973). *Journal of Neurophysiology*. **36**, 409–424.

Hubel, D.H. and Wiesel, T.N. (1962). *Journal of Physiology*. **160**, 106–154.

Hubel, D.H. and Wiesel, T.N. (1968). *Journal of Physiology*, **195**. 215–243.

Hubel, D.H. and Wiesel, T.N. (1977). *Proceedings of the Royal Society of London (B)*, **198**, 1–59.

Maunsell, J.H.R. and van Essen, D.C. (1983). *Journal of Neurophysiology*. **49**. 1127–1147.

Newsome, W.T., Gizzi, M.S., and Movshon, J.A. (1983). *Investigative Ophthalmology and Visual Science*, **24**, 106.

Peterhans, E. and von der Heydt, R. (1989). *Journal of Neuroscience*. **9**. 1749–1763.

Poggio, G.F., Motter, B.C., Squatrito, S., and Trotter, Y. (1985). *Vision Research*, **25**, 397–406.

Sekuler, R. (1975). In E.C. Carterette and M.P. Friedman (Eds.). **Handbook of Perception, Volume V: Seeing.** New York: Academic Press.

Stone, J. (1972). *Investigative Ophthalmology*, **11**, 338–344.

Stone, J. and Dreher, B. (1973). *Journal of Neurophysiology*. **36**, 551–567.

Tanaka, M. Lee, B.B., and Creutzfeldt, O.D. (1983). In J.D. Mollon and L.T. Sharpe (Eds.), **Colour Vision.** New York: Academic Press. 1983.

Thorell, L.G., DeValois, R.L., and Albrecht, D.G. (1984). *Vision Research*. **24**, 751–769.

Tolhurst, D.J. (1973). *Journal of Physiology*. **231**, 385–402.

Zeki, S.M. (1974a). *Journal of Physiology (London)*. **236**. 549–573.

Zeki, S.M. (1974b). *Journal of Physiology (London)*. **242**. 827–841.

About the Geometry Intrinsic to Neural Nets

A.J. Pellionisz
Dept. of Physiology and Biophysics
New York Univ. Med. Ctr.
550 First Ave, New York, NY. 10010

"Neural net" field will succeed, or fail, depending on meeting two crucial challenges. One is to discern, from the living brain, the mathematics that is inherent in CNS function. The next is to develop the electronic or optical technology that can suitably implement it (see Fig.1). This dual evolution is similar to previous natural science-based branches of technology; electrical- and nuclear engineering. In all cases, the basic science has to be established first (theory of electricity and nuclear physics in the past and brain theory in the future). Based on the "basic science", a novel technology might be developed, eventually leading to a new industry. A crucial problem is, that as known from the history of natural sciences, the emergence new disciplines often necessitates a creative process of establishing the mathematics that is intrinsic to the novel scientific problem. Theory of complex numbers and functions (in electricity) and of quantum mechanics (in nuclear physics) are recent examples, but it is worth remembering that even classical (Newtonian) mechanics had to be mathematically inventive (leading to the development of calculus).

There are apparent exceptions. Computer science evolved along a different path as it is not part of natural sciences. Its mathematics (Boolean algebra of mathematical logics, see [19]) was in place even before the suitable electronics to implement it was developed. This is probably one reason why "neural nets", the R&D of "brain like machines", often implies that the basic research including the identification of intrinsic mathematics is already done and, it seems, all one needs is the development of suitable technology. The classical viewpoint even equated Boolean algebra with the mathematical language of neural nets, in which neurons were simple binary elements [10]. Lately, since it is generally agreed that the brain is a massively parallel (array) processor, arrays as vectors and matrices of real numbers (representing points and transformations in the regular vectorspace with Euclidean geometry) seemed sufficient [20] as the mathematical tools of brain theory. But is it really true that mathematical brain theory is all there to base a new industry upon? An honest answer that neuroscience can give is "to the same extent that nuclear physics was ready in the early twentieth century".

Brain theory, as a mathematical theory of the only true neurocomputer (biological neural nets), is only in its infancy. Still, it is already evident that the mathematics inherent to CNS function is as non-trivial as it is non-orthodox. First, about thirty years after von Neumann pointed out the fundamental differences between the computer and the brain [11], it is becoming increasingly widespread to believe that the mathematics underlying brain function is not algebra [10], not even calculus [7] but geometry [8],[2],[14]. Second, it is becoming apparent that "neural geometry" as the basis of "neural neurocomputing" transcends the restrictions of a simple or even simplistic Euclidean structure. This presentation will demonstrate this fact with examples taken from neuroscience applications.

As shown in Fig.2., the classical Cartesian orthogonal coordinate systems [1] are demonstrably not the ones that nature actually uses in living brains. It is elaborated in detail elsewhere [16],[12] that coordinate systems intrinsic to sensorimotor mechanisms such as the vestibulo-collic apparatus use generalized vectors (tensors, c.f. [3]) that are typically non-orthogonal and overcomplete [5]. Based on tensor theory, network models of existing neural nets, such as the Vestibulo-Ocular Reflex (VOR) were constructed [16],[15] accounting to both the distributed nature of this reflex [12], and specifically elaborated the learning mechanism of such networks and in addition provided a structuro-functional model of the cerebellum in such sensorimotor transformations [17]. In contrast to the decade-old tensor approach, the generation-old classical system-

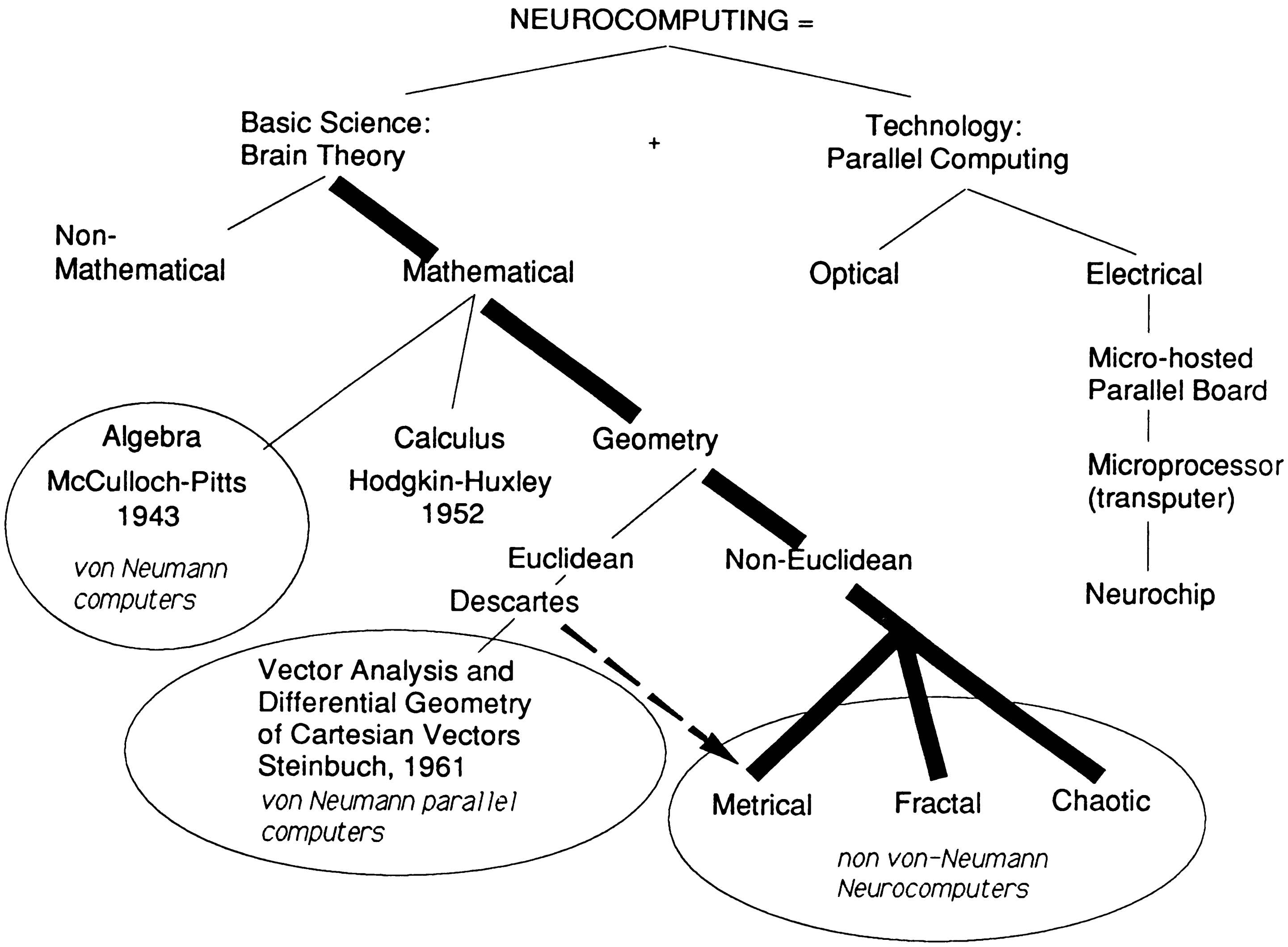


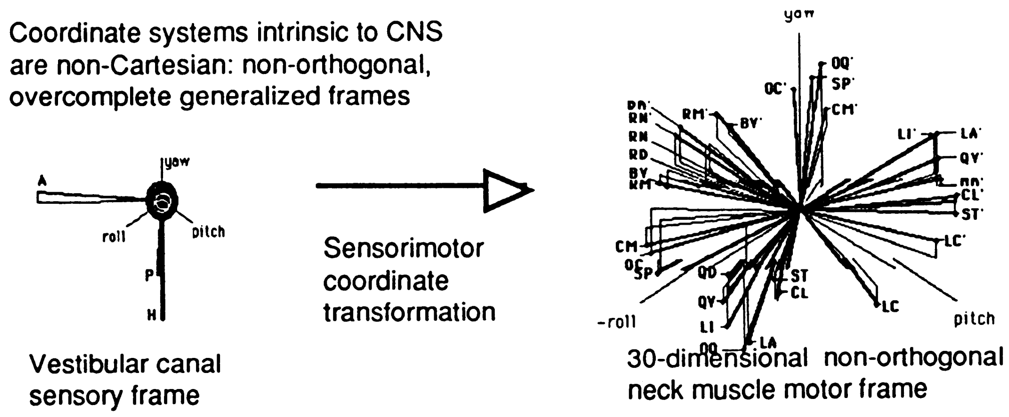
Fig.1. Neurocomputing is a new technology that has to be based on the intrinsic mathematics of biological brains

theory approach in gaze research could only produce, to date, single transformation (lumped) models, totally glossing over the underlying biological neural networks. In a quantitative model [18] of the Vestibulo-Collic Reflex (VCR) the non-Euclidean nature of e.g. the neck-motor metric is plainly evident in Fig.2.C. The tensor model of such non-Euclidean geometries has received experimental supporting evidence [6], [18]. The implications of the mathematical formalism of generalized (non-orthogonal) vectors (tensors) on the theory of associations forming the metric of the multidimensional intrinsic geometry of CNS will be elaborated elsewhere [13].

In addition to an obvious need of pulling away from extrinsic Cartesian frames of reference and the Euclidean geometry of the orthodox vectorspace, it is becoming evident that grossly non-metrical geometries are also manifest in structuro-functional properties of biological neural nets. Deterministic chaos [4] is one striking evidence. Recently, in an attempt of meeting the beautiful challenge from the originator of a rapidly emerging mathematical discipline of fractals [9], it has been demonstrated [14] that the structure of neurons reflects a fractal geometry (see Fig.3).

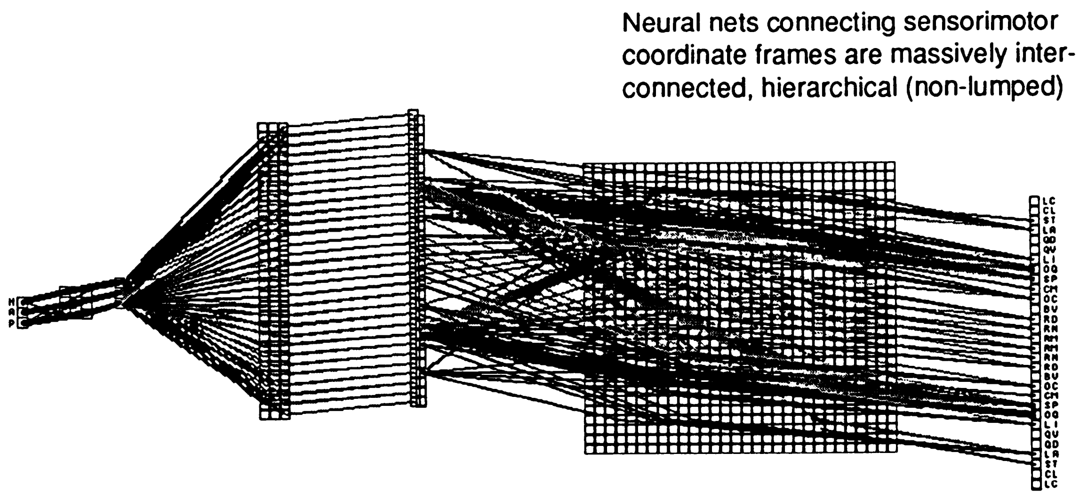
The empires of electrical- and nuclear technology were built on a foundation of basic research – not shying away from the horrendous primary task of discerning the appropriate intrinsic mathematics from nature herself. With enough investments at stake, the field of "neural nets" will do likewise.

A



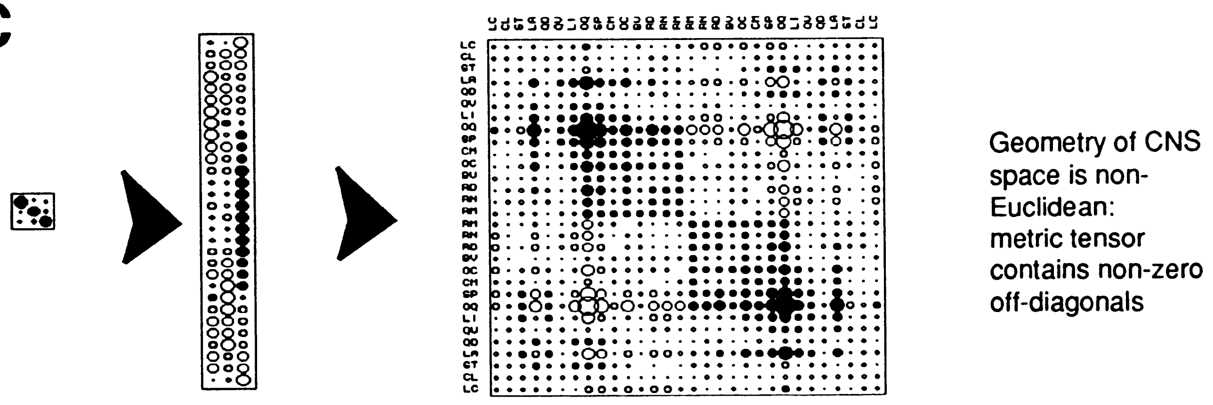
Coordinate systems intrinsic to CNS are non-Cartesian: non-orthogonal, overcomplete generalized frames
yaw
A
roll
pitch
P
H
Vestibular canal sensory frame
Sensorimotor coordinate transformation
-roll
30-dimensional non-orthogonal neck muscle motor frame
OQ
SP
CM
OC
RM
BY
LI
LA
QV
CL
ST
LC
RD
RN
QD

B



Neural nets connecting sensorimotor coordinate frames are massively inter-connected, hierarchical (non-lumped)
H
A
P

C



Geometry of CNS space is non-Euclidean: metric tensor contains non-zero off-diagonals

D

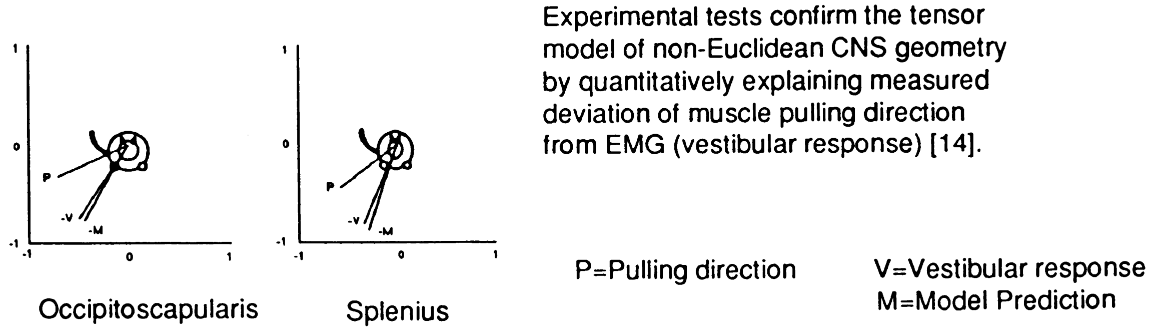


Fig.2. Demonstration of non-Euclidean metrical geometry intrinsic to CNS (for details, see [14])

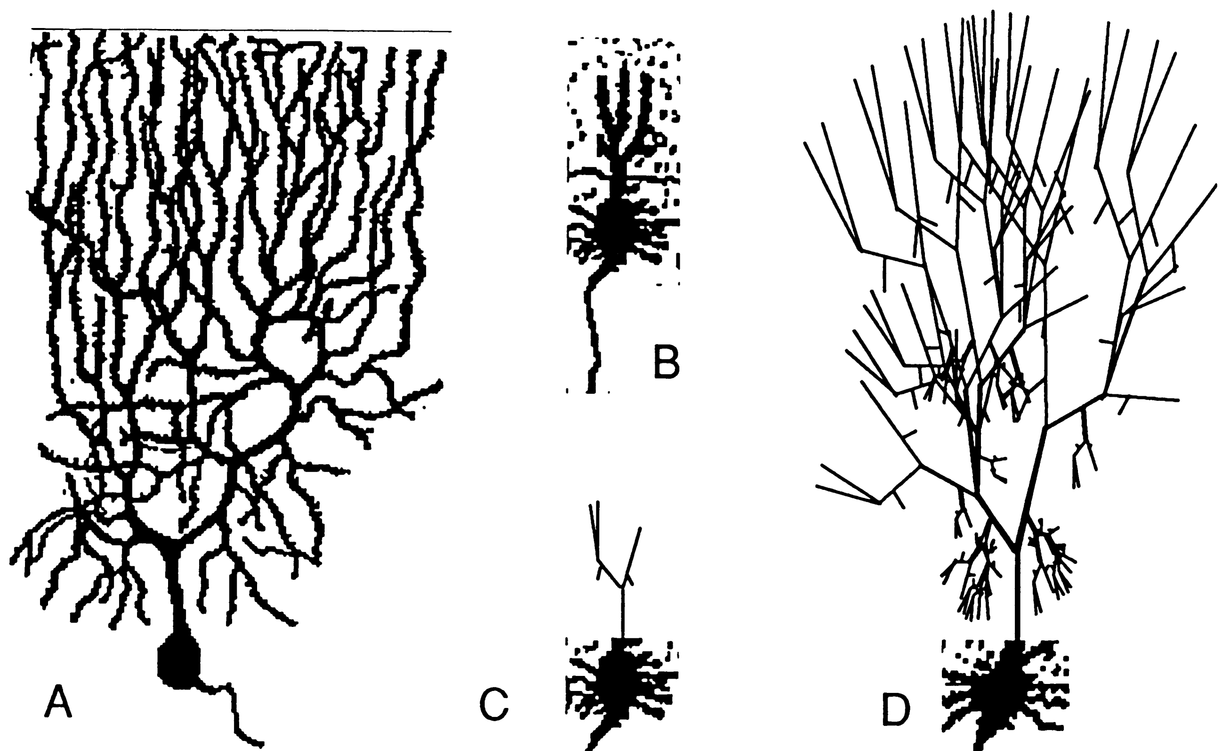


Fig.3. Demonstration of non-Euclidean, non-metrical (fractal) geometry of CNS. A,B: fully developed and embryonic Purkinje cells, C,D: fractal "embryonic" template and fully developed fractal dendritic tree (for details, see [14]).'

References

1. Descartes, R. "Treatise of Man." 1972 Harvard Univ. Press, Cambridge, MA. (Original in 1629 in Latin and French).
2. Eckmiller, R. and C. von Malsburg. Neural Computers. Proc of the NATO Adv. Research Workshop, Springer. 1988.
3. Einstein, A. (., 1916) In: Sommerfeld A (ed) The principle of relativity. (1952) Dover, New York. p 111-164. The foundation of the general theory of relativity: 1916.
4. Freeman, W. Simulation of chaotic EEG patterns with a dynamic model of the olfactory system. Biol. Cybern. **55**: 1987.
5. Georgopoulos, A., S. AB and K. RE. Neuronal population coding of movement direction. Science. **233**: 1416-1419, 1986.
6. Gielen, C. C. A. M. and E. J. van Zuylen. Coordination of arm muscles during flexion and supination: Application of the tensor analysis approach. Neuroscience. **17**: 527-539, 1986.
7. Hodgkin, A. and H. AF. A quantitative description of membrane current and its application to conduction and excitation in nerve. J. Physiol. (London). : 1176:500-544, 1952.
8. Koenderink, J. J. Geometrical structures determined by the functional order in nervous nets. Biol. Cybern. **50, 43-50**: 1984.
9. Mandelbrot, B. "The fractal geometry of nature." 1977 W.H. Freeman. New York.
10. McCulloch, W. S. and W. Pitts. A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys. : 5, 115-133., 1943.
11. Neumann, J. v. "The computer and the brain." 1958 Yale Univ. Press, New Haven, CT.
12. Pellionisz, A. "Tensorial aspects of the multidimensional approach to the vestibulo-oculomotor reflex and gaze." Reviews of Oculomotor Research. I. Adaptive Mechanisms in Gaze Control, pp. 181-296. Berthoz and Melvill-Jones ed. 1985 Elsevier. Amsterdam.
13. Pellionisz, A. "The geometry of brain function: Tensor network theory." 1989 Cambridge Univ. Press (In Press).
14. Pellionisz, A. "Neural geometry: Towards a fractal model of neurons." Models in Brain Function. Cotterill ed. 1989 Cambridge Univ. Press.
15. Pellionisz, A. and W. Graf. Tensor network model of the "three-neuron vestibulo-ocular reflex-arc" in cat. J. Theoretical Neurobiology. **5**: 127-151, 1987.
16. Pellionisz, A. and R. Llinás. Tensorial approach to the geometry of brain function: Cerebellar coordination via metric tensor. **5**: 1125-1136, 1980.
17. Pellionisz, A. and R. Llinás. Tensor Network Theory of the metaorganization of functional geometries in the CNS. Neurosci. **16**: 245-274, 1985.
18. Peterson, B. W., A. J. Pellionisz, J. A. Baker and E. A. Keshner. Functional morphology and neural control of neck muscles in mammals. Am. Zoology. **29**: 139-149, 1989.
19. Shannon, C. A mathematical theory of communication. Bell System Technology J. : 27, 3-4., 1948.
20. Steinbuch, K. Die Lernmatrix. Kybernetik. : 1, 36-45., 1961.

Optimal Preprocessing Networks and a Data Processing Theorem

Donald St. P. Richards[†] and William B Levy[‡]
[†]Department of Mathematics and [‡]Department of Neurosurgery
University of Virginia, Charlottesville, Virginia 22903

1. Introduction

In this paper we continue (cf., Levy, 1989b) to treat the problem of constructing a neural-like network that performs optimal predictions for very general environments. We are particularly interested in the computations that could be implemented in a neural network to generate optimal predictions. Here we start by defining and studying the information in X, a vector of inputs, that might be made available to the prediction-generating neurons. (Each prediction-generating neuron generates a conditional probability of its own future state.) The vector X can be used as the conditioning variable for making predictions. However, it might be useful to process X before it reaches the prediction-generating neurons. We consider how best to preserve the predictive information in X while modifying its form. That is, this paper seeks to discover good forms for preprocessing the vector of inputs X, the conditioning variable of the prediction. In moving towards this goal we obtain an analog of information theory's data processing theorem for a preprocessor in a prediction network and discuss the implications of this theorem for constructing networks that solve an optimal preprocessing problem. We identify conditions under which this theorem allows only one type of optimal preprocessing. Because our prediction generating neurons benefit from statistically independent X inputs,the transformation we seek also minimizes statistical dependence. Finally, we sketch a computationally tractable algorithm to perform the desired preprocessing.

2. The data processing theorem

The data processing theorem (cf. Csiszar and Körner, 1981, p.55; Shannon and Weaver, 1949; p. 57) states that if sequential encoding and decoding procedures are applied to data represented by a stochastic process, *e.g.*, a communication channel, the amount of information contained in the data cannot be increased because of the processing. We want to formulate for neural-like networks an analog of the data processing theorem. This theorem will be an analog if, whenever a representation X is transformed into a representation $Y = f(X)$ by a transformation f, then predictive information cannot be increased by this transformation.

For simplicity we shall assume that all random variables are discrete. In formulating an analog of the data processing theorem for the type of neural-like networks considered in this paper, we need to measure the amount of predictive information contained in the conditional distribution $P_{Z|Y}$. To measure the difference between the conditional distribution $P_{Z|Y=y}$ and the prior distribution P_Z, we use the divergence (or relative entropy) measure:

$$D(P_{Z|Y=y}||P_Z) = \sum_z P(Z=z|Y=y)\log\frac{P(Z=z|Y=y)}{P(Z=z)}.$$

This divergence represents the information in Y about Z. The greater this quantity, the greater the difference between the distributions P_Z and $P_{Z|Y=y}$, and hence the greater the potential gain in our knowledge about Z as a consequence of knowing y. Therefore, $D(P_{Z|Y=y}||P_Z)$ is an appropriate measure of the predictive information about Z given that $Y = y$.

Averaging over all possible values of Y produces a measure

$$D(P_{Z|Y}||P_Z) = \sum_y P(Y=y)D(P_{Z|Y=y}||P_Z)$$

of the predictive information about Z present in the random variable Y.

We can now state our analog of the data processing theorem. This result holds when the network is unable to affect or control the environment generating its inputs.

Theorem. Let X, Y and Z be finite dimensional random vectors with discrete components and with $Y = f(X)$ for some transformation f. Then predictive information about Z cannot be increased by transforming X to Y; that is,

$$D(P_{Z|Y}||P_Z) \le D(P_{Z|X}||P_Z).$$

Proof. Let $g(t) = t \log t$, $0 \le t \le 1$, and observe that $g(t)$ is a convex function. Next recall that

$$P(Y = y) = P(f(X) = y) = \sum_{\{x:f(x)=y\}} P(X = x).$$

Now

$$P(Z = z|Y = y) = \frac{P(Z = z, Y = y)}{P(Y = y)} = \sum_{\{x:f(x)=y\}} \frac{P(Z = z, X = x)}{P(Y = y)}$$

$$= \sum_{\{x:f(x)=y\}} \frac{P(X = x)}{P(Y = y)} P(Z = z|X = x),$$

which is a convex combination of the values $P(Z = z|X = x)$ because

$$\sum_{\{x:f(x)=y\}} \frac{P(X = x)}{P(Y = y)} = 1.$$

From the convexity of the function $g(t)$ it follows that

$$P(Z = z|Y = y) log P(Z = z|Y = y) \le \sum_{\{x:f(x)=y\}} \frac{P(X = x)}{P(Y = y)} P(Z = z|X = x)$$

$$\times \, log P(Z = z|X = x).$$

If we denote Shannon's entropy for the random variable Z by

$$H(Z) = -\sum_z P(Z = z) log P(Z = z),$$

then we obtain that

$$\begin{aligned} D(P_{Z|Y}||P_Z) &= H(Z) + \sum_{y,z} P(Y = y) P(Z = z|Y = y) log P(Z = z|Y = y) \\ &\le H(Z) + \sum_{\{y,z;x:f(x)=y\}} P(X = x) P(Z = z|X = x) log P(Z = z|X = x) \\ &= H(Z) + \sum_{x,z} P(Z = z, X = x) log P(Z = z|X = x) \\ &= D(P_{Z|X}||P_Z). \end{aligned}$$

This result leads to a powerful observation. Note that the above inequality does not depend on the underlying environment probabilities. This implies that the attempts to reconstruct (or estimate) the environment probabilities or even just its distributional form, as compared to the network's own representation probabilities, will be of no help in making predictions. Further if we were to guess incorrectly those environment probabilities, and this would happen in the "real world" whenever an assumption of distributional form is made about the environment, then the preprocessor could lose predictive information. This implication follows from the fact that such reconstructions (or estimations) are transformations. Therefore, the environment probabilities should be ignored in the search for suitable transformations f. On the other hand, the probabilities of network states should not be ignored.

Note that if a network can control its environment, then we could create a situation in which $H(X|Y) = 0$ and $H(Z|X) = 0$ but Y drives the environment to produce Z so that $D(P_{Z|Y}||P_Z) > 0$ while $D(P_{Z|X}||P_Z) = 0$. Even so, an extension of the theorem to such a network can cover many interesting situations in which the network is allowed to act on the evironment. Specifically, if we consider only network actions on the environment that are produced slowly and in a stepwise manner, then at any one such step the theorem holds conditional upon the state into which the network has driven the environment.

The second important consequence of the theorem is the following result.

Corollary. Suppose that the random vectors X and Y take values in $\{0,1\}^n$ and $\{0,1\}^m$, respectively, where $Y = f(X)$ and $P(X = x) > 0$ for all $x \in \{0,1\}^n$. If f is one-to-one then
(a) $D(P_{Z|X}||P_Z) = D(P_{Z|Y}||P_Z)$;
(b) $n \leq m$.

In particular, m can never be smaller than n without risking a loss of predictive information. If (a) holds and $n = m$, then f is a permutation on $\{0,1\}^n$.

Proof. The proof of (a) is obtained by going through the proof of the above theorem and using standard results for the situation when equality holds in the inequalities of the theorem. The proof of (b) follows obviously from the assumption that f is one-to-one.

If in contradiction to the assumptions of the Corollary, $P(X = x) = 0$ for some $x \in \{0,1\}^n$, then Y can have a dimension $m < n$, without loss of predictive information. When $m < n$ the transformation will not be a permutation. Here is an example which illustrates this situation.

Example. Let the random vector $X = (X_1, X_2)$ take values in the space $\{0,1\}^2$ such that the distribution of X is as follows: $P(0,0) = .6$, $P(1,0) = .4$, and $P(0,1) = P(1,1) = 0$. Define $Y = f(X)$ where $f : \{0,1\}^2 \rightarrow \{0,1\}$ such that $f(0,0) = f(0,1) = 0$ and $f(1,0) = f(1,1) = 1$. Then computations based on Y are as good as those based on X, the dimension of the space has been lowered, and no information has been lost. Note that this example does not contradict the Corollary because here we have $P(X = x) = 0$ for some $x \in \{0,1\}^2$. It is not difficult to similar construct examples for Z such that $D(P_{Z|X}||P_Z) = D(P_{Z|Y}||P_Z)$.

In practice, a reduction of the dimension is very likely to result in the loss of predictive information. Even for moderately large n, many configurations may never be sampled in realistic time spans, even if $P(X = x) > 0$ for all x. Thus, the network will not be able to determine if $P(X = x) = 0$ for any x. Therefore the only transformations that will minimize the dimension of Y and that are guaranteed to avoid loss of information are permutations. Because our networks are large, we will search for transformations which are "approximately" permutations.

3. Implications for our networks

Based on the results above, it may seem that no transformation should be applied to X before using it to predict Z. However, this is not true because the mere existence of information in X about Z does not imply that the prediction-generating neurons are able to utilize this information. That is, the computations that are available to a prediction-generating neuron will be constrained to some very small set of all possible computations while the computations needed to use all the information, $D(P_{Z|X}||P_Z)$, will often lie outside this set. In addition other constraints, e.g., computational complexity, can prevent a network from using information that may be present in the data. Therefore it is sometimes helpful to transform the representations to use this information.

This computational limitation motivates transforming X so that each prediction generating neuron makes better use of the predictive information that might be present. In particular the prediction-generating networks that we are studying seem to produce more accurate predictions when there is a reduction in the statistical dependence between the coordinates $X_1, \ldots, X_n$ of the conditioning variable X.

Example. Let the random vector $X = (X_1, X_2)$ take values in the space $\{0,1\}^2$ such that the distribution of X is as follows: $P(0,0) = .18$, $P(1,0) = .12$, and $P(0,1) = .28$, $P(1,1) = .42$. Define $Y = f(X)$ where $f : \{0,1\}^2 \rightarrow \{0,1\}^2$ such that the distribution of Y is as follows: $P(0,0) = .18$, $P(1,0) = .12$, and $P(0,1) = .42$, $P(1,1) = .28$. In this case, the components of X are *not* independent while the components of Y *are* independent.

In general it will not always be possible to achieve independence of the components with a permutation transformation, but it often happens that the statistical dependence between the components can be reduced. (This type of manipulation was recognized by Barlow (1959).) In the most general setting, statistical dependence is measured by Watanabe's (1969) extension of the measure of mutual information as defined in Shannon and Weaver (1949). Specifically, this means that we measure dependence between the components $X_1, \ldots, X_n$ of X by the quantity

$$I(X) = E_X \log \left[\frac{P(X)}{\prod_{i=1}^{n} P(X_i)} \right].$$

Therefore the smaller the value of $I(X)$ the lesser the statistical dependence between $X_1, \ldots, X_n$, with $I(X) = 0$ indicating complete independence.

In high-dimensional spaces, methods for identifying permutations of $\{0,1\}^n$ that reduce statistical dependence are usually difficult to implement. Indeed, the number of such permutations is $2^n!/n!2^n$, a number which grows faster than exponentially with n. To prove these results, let us count the number of distinct permutations which might have to be examined for $X \in \{0,1\}^n$. There are 2^n possible values for X, hence there are $2^n!$ permutations of these points. However, some of these orderings will not change the value of the measure of statistical dependence $I(X)$. In particular, the $n!$ permutations which are generated by relabeling the n axes do not change statistical dependence. Also, because entropy is symmetric about 1/2, the 2^n permutations which are generated by complementing axes also do not change statistical dependence. Therefore the number of permutations which we need to examine is $2^n!/n!2^n$. If we use the notation $a_n \sim b_n$ whenever $a_n/b_n \rightarrow 1$ as $n \rightarrow \infty$, then by Stirling's formula, $n! \sim (2\pi)^{1/2} n^{n+1/2} e^{-n}$. Therefore as $n \rightarrow \infty$,

$$\frac{2^n!}{n!2^n} \sim \frac{2^{n(2^n-1/2)} e^{n-2^n}}{n^{n+1/2}} \sim 2^{2^n n},$$

implying that the number of good permutations increases faster than 2^n.

This calculation implies that any deterministic search for the best transformation is out of the question for solving problems in the real world. However, it may be possible to construct such a transformation sequentially, i.e., sample by sample, under the relaxed requirement of finding a very close relative of the optimal transformation with high probability.

4. The tight packing algorithm

We have proposed (Levy, 1989a) an algorithm for constructing a transformation $f : X \rightarrow Y$ which minimizes $\sum_j H(Y_j)$ while preserving the information $D(P_{Z|Y} || P_Z)$. This algorithm, called "tight packing," uses the property that the sample space is much larger than the number of possible samples which will ever be received (*e.g.*, $2^{100,000}$ vs. 2^{20}). By transforming the samples in X to points in $Y \in \{0,1\}^n$ so that the points are distributed on the vertices of the smallest possible hypercube(*i.e.*,, the hypercube with minimal Hamming distance between all vertices), then independence can be achieved whenever the number of samples received is a power of 2. This procedure, however, does require a dynamically developing transformation that progressively enlarges the size of the space into which the samples are mapped.

Acknowledgements

The version given here of the proof of the theorem is due to D. Costa; we thank him also for extensive discussions of the ideas here. D.R. was supported in part by NSF-8802929 and WBL was supported in part by NIMH RSDA MH 00622.

References

Barlow, H. B. (1959). Sensory mechanisms, the reduction of redundancy, and intelligence. In: *Mechanisation of Thought Processes*, Vol. II (pp. 537-559). London: Her Majesty's Stationery Office.

Csiszár, I. and Körner, J. (1981). *Information Theory: Coding Theorems for Discrete Memoryless Systems.* Academic Press, New York, 452 pp.

Levy, W. B. (1989a). A computational approach to the hippocampal function. In: *Computational Models of Learning in Simple Neural Systems.* (Hawkins, R. D. and Bower, G. H., Eds.) Academic Press, New York.

Levy, W. B. (1989b) Maximum entropy prediction in neural networks; submitted to this Conference.

Shannon, C. E. and Weaver, W. (1949). *The Mathematical Theory of Communication.* University of Illinois Press, Urbana, IL, 125 pp.

Watanabe, S. (1969). *Knowing and Guessing. A Quantitative Study of Inference and Information.* New York: John Wiley.

Learning “Semantotopic Maps” from Context

Helge Ritter
Beckman Institute and Department of Physics
University of Illinois at Urbana-Champaign
Urbana IL 61801
USA

Teuvo Kohonen
Helsinki University of Technology
Laboratory of Computer and Information Science
Rakentajanaukio 2 C, SF-02150 Espoo
Finland

Abstract: Self-organized formation of topographic maps for abstract data, such as words, is demonstrated in this work. Semantic relationships inherent in the data are reflected by their relative distances in the map. This is made possible by assuming that the logic similarity relationships between words are defined by the contexts in which the words appear, and which are again reflected in a self-organizing topological feature map. In the demonstration, the context is defined by simple sentences, consisting of structured sequences of nouns, verbs, and adverbs. Such phrases already involve many of the abstractions that appear in thinking, namely, the most common categories, into which the words are then automatically grouped in our simulations.

1. Introduction

One challenge to neural network research is to understand to what extent a simple adaptive system can detect useful abstractions and generalizations from raw data. The self-organizing feature maps earlier introduced by one of the authors ([2-6]) were adressing this issue and exhibited very interesting capabilities by forming nonlinear, two-dimensional maps of low-level sensory signals, such as e.g. phoneme spectra.

This approach was inspired by the occurrence of similar maps in the primary sensory brain areas, such as the tonotopic map in the auditory cortex, or the somatotopic and retinotopic maps in the sensory cortices. Common to both of these brain maps as well as certain artificial sensory feature maps earlier demonstrated, is that they almost directly display *physical properties* of observations, such as e.g. intensity, direction, or frequency. It seems that the subsequent, higher processing stages continue to build representations that are further organized according to the *logical properties* of their input items ([1,9]).

In this work we report on an intriguing new result, namely, that artificial higher level maps of, e.g., words, topographically organized according to semantic meaning, can be obtained by the same basic self-organizing process used already for the simpler maps in [2-6,10], *provided that the words are offered together with the context in which they occur.* These resulting *“semantotopic maps”* raise the intriguing possibility that at successively higher stages of processing the same self-organizing process might be capable of gradually building representations of increasing levels of abstraction.

We have to mention a couple of recent works in which the back propagation algorithm was used to create internal semantic representations ([7,8]). In contrast to these approaches, the present self-organizing maps work in an *unsupervised* mode and thus demonstrate the autonomous capabilities of a neural network to create “higher intelligence” ([11]).

2. The Model Equations

The model assumes a sheet of laterally interacting adaptive neurons, connected to a common bundle of input fibers. Any activity pattern on the input fibers gives rise to excitation of some local group of neurons. After learning, the spatial positions of the excited groups specify a mapping of the input signals onto the two-dimensional sheet, having the property of a topographic map, i.e. it represents distance relations of the high-dimensional space of the input signals approximately as distance relationships on the two-dimensional neural sheet. This remarkable property follows from the assumed lateral interactions and a very simple, biologically justifiable adaptation law given below (for details cf.[6,11]).

The activity pattern at the input is described by an n-dimensional real vector $\mathbf{x}$, normalized to $||\mathbf{x}|| = 1$, where n is the number of input lines. The responsiveness of neuron r is specified by an n-dimensional vector $\mathbf{w}_r$ of “synaptic efficacies”, and is measured by the dot product $\mathbf{x} \cdot \mathbf{w}_r$. Each neuron is labeled by its two-dimensional position r in the sheet. The group of excited neurons is taken to be centered at the neuron s for which $\mathbf{x} \cdot \mathbf{w}_s$ is maximal. Its extent and shape are described by a function h_{rs}, whose value is the excitation of neuron r, if the group center is at s. A rather realistic modeling choice for h_{rs} is the Gaussian

$$h_{rs} = \exp\left(-\frac{1}{\sigma^2}||r - s||^2\right). \qquad (1)$$

During learning, each presentation of an input $\mathbf{x}$ will cause an adjustment in the weights $\mathbf{w}_r$ given by

$$\mathbf{w}_r^{(new)} = \mathbf{w}_r^{(old)} + \epsilon \cdot h_{rs} \cdot (\mathbf{x} - \mathbf{w}_r^{(old)}). \tag{2}$$

Eq. (2) can be justified by assuming the traditional Hebbian law for synaptic modification, and an additional nonlinear, "active" forgetting process for the synaptic strengths ([6]).

3. Formation of Semantic Maps from Context

Much of higher-level processing, in particular *language* and *reasoning*, seems to be based on *discrete symbols* and their *logical meaning*. One might think that applying the neural adaptation laws (2) to a symbol set (coded by using vectors) might create a topographic map that displays the *"logical distances"* between the symbols. However, there occurs a problem associated with the discrete nature of symbols. Unlike continuous data, symbolic items usually admit no meaningful metric, which could be derived from their encoding in any simple way. Hence logical relatedness between different symbols, such as e.g. words, will in general not be directly detectable from any metric relations between their encodings, even when the symbols represent similar items. How could it then be possible to map them topographically? The answer is that the symbol, at least in the learning process, must be presented *in due context*, i.e. in conjunction with all or part of the attribute values of the item it encodes, or with other, correlating symbols.

The simplest system model for symbol maps assumes each data vector $\mathbf{x}$ as a concatenation of two (or more) fields, one specifying the symbol code, denoted by $\mathbf{x}_s$, and the other the attribute set, denoted $\mathbf{x}_a$, respectively.

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_s \\ \mathbf{x}_a \end{bmatrix} = \begin{bmatrix} \mathbf{x}_s \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \mathbf{x}_a \end{bmatrix}. \tag{3}$$

Eq. (3) illustrates in vector notation that the encodings of the symbol part and the attribute part can form a vector sum of two orthogonal components. The core idea underlying symbol maps is that the two parts are weighted properly such that *the norm of the attribute part predominates over that of the symbol part during the self-organizing process*; the topographical mapping then mainly reflects metric relationships of the attribute sets. Since the inputs for symbolic signals, however, are also active all the time, memory traces from them are formed to the corresponding inputs of those cells of the map that have been selected (or actually forced) by the attribute part. *If then, during recognition of input information, the attribute signals are missing or are weaker, the (same) map units are selected on the basis of the symbol part solely. In this way the symbols become encoded into a spatial order reflecting their logic (or semantic) similarities.*

In the following, we shall demonstrate this idea when the symbols correspond to words forming a simple text. Then perhaps the most straightforward (and very literal!) way to define the context of a word is to take all those words (together with their serial order) that occur in a certain "window" around the selected word. For simplicity, we shall imagine that the content of each "window" can somehow be presented to the input ports of the neural system. We are not interested in any particular means for the conversion of, say temporal signal patterns into parallel ones (for this task on could employ paths with different delays, eigenstates that depend on sequences, or any other mechanisms implemented in the short-term memory).

A sequence of randomly generated three-word sentences provided the input "text". The vocabulary is listed in Fig. 1a and comprises nouns, verbs, and adverbs. Each class has further subdivisions, such as names of persons, animals, and inanimate objects in the category of nouns. These distinctions are in part of a grammatical, in part of a semantic nature. Generally, they cannot be inferred from any patterns used for the encoding of the individual words, but only from the context, in which the words occur. In natural languages, such a context might comprise a rich variety of sensory experiences. In this very limited demonstration, however, we will only take into account the context provided by the immediately adjacent textual environment of each word occurrence. It will turn out that even this extremely restricted context will suffice to convey some interesting semantic structures. Of course this requires that each sentence be not totally random, but obey at least some rudimentary rules of grammar and semantic correctness. This is ensured by *restricting the sentences to a set of* 39 *"legal" sentence patterns only* (Fig. 1b). Each sentence was constructed by substituting the numbers in a randomly selected triple from Fig. 1b by words with compatible numbering in Fig. 1a, making a total of 498 three-word sentences possible, a few of which are given in Fig. 1c.

In the first demonstration, the context of a word was restricted to the pair formed by its immediate predecessor and successor (ignoring any sentence borders). To prevent any semantic or grammatical information from being conveyed by patterns in the encoding of the individual words, we encoded each word by a seven-dimensional random vector of unit length. For each word, this vector was chosen independently from an isotropic probability distribution at the outset of the simulation. Each predecessor/successor-pair was represented by concatenating the code vectors of the two words to a 14-dimensional code vector.

Bob/Jim/Mary	1
horse/dog/cat	2
beer/water	3
meat/bread	4
runs/walks	5
works/speaks	6
visits/phones	7
buys/sells	8
likes/hates	9
drinks/eats	10
much/little	11
fast/slowly	12
often/seldom	13
well/poorly	14

Fig.1a

Sentence Patterns:		
1-5-12	1-9-2	2-5-14
1-5-13	1-9-3	2-9-1
1-5-14	1-9-4	2-9-2
1-6-12	1-10-3	2-9-3
1-6-13	1-11-4	2-9-4
1-6-14	1-10-12	2-10-3
1-6-15	1-10-13	2-10-12
1-7-14	1-10-14	2-10-13
1-8-12	1-11-12	2-10-14
1-8-2	1-11-13	1-11-4
1-8-3	1-11-14	1-11-12
1-8-4	2-5-12	2-11-13
1-9-1	2-5-13	2-11-14

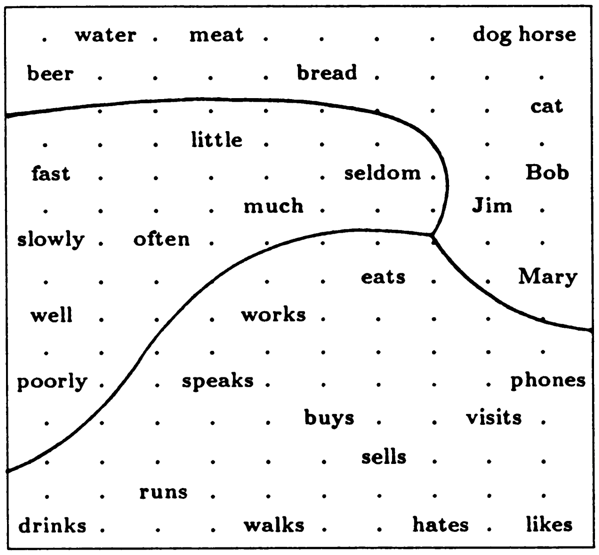
Fig.1b

Mary likes meat
Jim speaks well
Mary likes Jim
Jim eats often
Mary buys meat
dog drinks fast
horse hates meat
Jim eats seldom
Bob buys meat
cat walks slowly
Jim eats bread
cat hates Jim
Bob sells beer
(etc.)

Fig.1c

It turned out in all of our computer experiments that instead of paying attention to each phrase separately, a much more efficient learning strategy was to consider each word in its *average context* over a set of possible clauses, before presenting it to the learning algorithm. The (mean) context of a word was thus first defined as *the average over 10,000 sentences of all code vectors of predecessor/successor-pairs surrounding that word.* The resulting thirty 14-dimensional "average word contexts", normalized to unit length, assumed the role of the "attribute fields" $\mathbf{x}_a$ in equation (3). Each "attribute field" was combined with a 7-dimensional "symbol field" $\mathbf{x}_s$, consisting of the code vector for the word itself, but scaled to length a. The use of the random code vectors guaranteed that the symbol fields $\mathbf{x}_s$ did not convey any systematic information about similarity relationships between the words. The parameter a determined the relative influence of the symbol part $\mathbf{x}_s$ in comparison to the context part $\mathbf{x}_a$ and was set to $a = 0.2$.

For the simulation, a planar lattice of 10 × 15 formal neurons was used. Initially each neuron made only weak random connections to the $n = 21$ input lines of the system, so that no initial order was present. The learning step size was $\epsilon = 0.8$ and the radius $\sigma(t)$ of the adjustment zone was gradually decreased from an initial value $\sigma_i = 4$ to a final value $\sigma_f = 0.5$ according to the law $\sigma(t) = \sigma_i \cdot (\sigma_f/\sigma_i)^{t/t_{max}}$. Here t counts the number of adaptation steps.



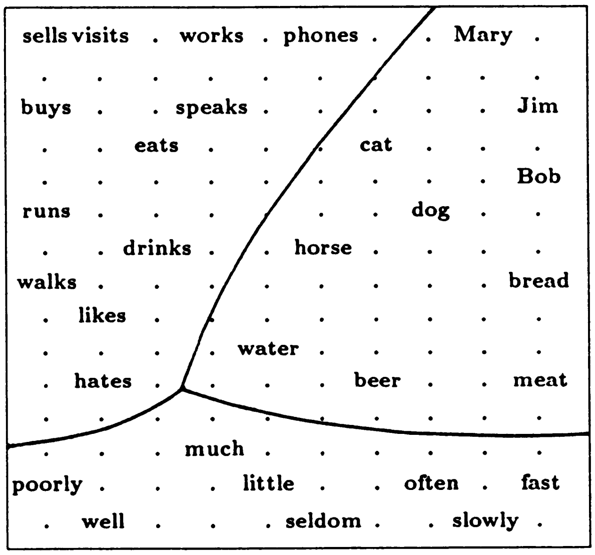


Fig.2: "Semantic map" obtained after 2000 presentations of word-context-pairs derived from 10,000 random sentences of the kind shown in Fig.1c. Nouns, verbs and adverbs are segregated into different domains. Within each domain a further grouping according to aspects of meaning is discernible.

Fig.3: This map has been obtained by the same procedure as the map in Fig.2, but with a more restricted context that included only the immediate predecessor of each word.

After $t_{max} = 2000$ input presentations the responses of the neurons to presentation of the symbol parts alone were tested. In Fig. 2, the symbolic label was written to that site at which the symbol signal $\mathbf{x} = [\mathbf{x}_s, 0]^T$ gave the maximum response. We clearly see that *the contexts have "channeled" the word items to memory positions whose arrangement reflects both grammatical and semantic relationships.* Words of same type, i.e. nouns, verbs, and adverbs, have segregated into separate, large domains. Each of these domains is further subdivided according to similarities on the semantic level. For instance, names of persons and animals tend to be clustered in separate subdomains of a common "noun-domain", reflecting different co-occurrence with, e.g., verbs such as "run" and "phone". Adverbs with opposite meaning tend to be particularly close together, as their opposite meaning ensures them maximum common usage. The grouping of the verbs indicates differences in how they can co-occur with adverbs, persons, animals, and non-animate objects such as e.g. food.

Figure 3 shows the result of a further computer experiment, based on the same vocabulary and the same sentence patterns as before. However, in this simulation the context of a word was restricted to its immediate predecessor only (i.e. the context now consists of a 7-dimensional vector). Even this very limited context proved sufficient to produce a map with roughly similar properties as in Fig. 2. This shows that the displayed regularities are fairly robust to changes in the details of the encoding as long as the context captures a sufficient amount from the underlying logical structure.

One might argue that the structure resulting in the map has artificially been created by a preplanned choice of the sentence patterns allowed for the input. However, it is easy to check that the patterns in Fig. 1b almost completely exhaust the possibilities for combining the words in Fig. 1a into semantically well-formed three-word sentences. This may make it clear that all the selected sentence patterns were really determined by the constraints inherent in the semantically correct usage of the words, and not vice versa. Moreover, a significant percentage of the word neighborhoods extended across borders of the randomly concatenated sentences. As this concatenation was unrestricted, this constituted a kind of "noise" remarkably well tolerated by the ordering process. Fig. 2 and Fig. 3 also show that the maps generated by the model are not unique and that there are different, almost equivalent ways, in which a set of similarity relationships can be displayed in a map. If uniqueness is desired, further constraints, such as e.g. boundary conditions or some coarse initial ordering, might be used to initially "polarize" the system suitably.

4. Discussion

The results of this work show that the principle of self-organizing maps can be extended to higher levels of processing, where the relationships between items are more subtle and less apparent from their intrinsic features, a property that is characteristic of ***symbolic expressions.*** Symbols, in general, do not contain metrically relatable components. Consequently, meaningful topographic maps of symbols must no longer display the intrinsic features, but instead the ***logical similarities*** of their inputs. It turns out that organized mappings of symbolic data may still ensue from the same basic adaptation laws, provided that the symbolic input data are presented together ***with a sufficient amount of context, that then defines the similarity relationships between them.*** If the symbolic descriptions leave memory traces on the same neurons at which the contextual signals converge, too, the same neurons then also become sensitized to the symbolic signals in a spatial order that reflects their logical similarity. In the case of words, the strong correlation between local context and word meaning approximates a ***semantic ordering*** as it is met in natural languages.

Considering the richness of language, one might suspect that the "semantic space" ought to be extremely high-dimensional, and that only little can be gained by mapping semantic meaning onto manifolds as low as two-dimensional. A simple argument indicates that perhaps surprisingly few dimensions may be sufficient to represent "semantic space". Assume that each word is used to denote one of three "grade values" (e.g. "weak", "intermediate", "strong") along each of a set of independent "semantic dimensions". Then a 12-dimensional space can accommodate already $3^{12} = 531\,441$ words, significantly more than needed by natural languages. If each word denotes more than three values per axis, even fewer dimensions will suffice. This makes it very likely that a limited number of low-dimensional maps along different (possibly nonlinearly curved) directions may be able to represent a significant amount of semantic structure.

References:

[1] Caramazza A. (1988) Some Aspects of Language Processing revealed through the Analysis of Aquired Aphasia: The Lexical System. Ann. Rev. Neurosci. pp. 395-421

[2] Kohonen T. (1982a) Self-organized Formation of Topologically Correct Feature Maps. Biol. Cybern. 43:59-69.

[3] Kohonen T. (1982b) Analysis of a Simple Self-organizing Process. Biol. Cybern. 44:135-140.

[4] Kohonen T. (1982c) Clustering, Taxonomy and Topological Maps of Patterns. Proc. Sixth Int. Conf. Pattern Recognition (IEEE Computer Society Press, Silver Spring) pp. 114-128.

[5] Kohonen T., Mäkisara K., Saramäki T. (1984) Phonotopic Maps - Insightful representation of Phonological Features for Speech Recognition. Proc. IEEE Seventh Int. Conf. Pattern Recognition, (IEEE Computer Society, Montreal, Canada) pp. 182-185.

[6] Kohonen T. (1984) Self-Organization and Associative Memory. Springer Series in Information Sciences 8, Heidelberg.

[7] Miikkulainen R., Dyer M.G. (1988a) Forming Global Representations with Extended Backpropagation. Proc. IEEE ICNN 88, San Diego, Vol.I, 285-292 (IEEE Computer Society Press).

[8] Miikkulainen R., Dyer M.G. (1988b) Encoding Input/Output Representations in Connectionist Cognitive Systems. In Touretzky, Hinton and Sejnowski (Eds.) Proceedings of the 1988 Connectionist Models Summer School, CMU (Morgan Kaufmann Publishers Inc.).

[9] Ojemann G.A., (1983) Brain organization for language from the perspective of electrical stimulation mapping. Behav. Brain Sci. 189-230.

[10] Ritter H., Schulten K. (1988) Kohonen's Self-Organizing Maps: Exploring their Computational Capabilities. Proc. IEEE ICNN 88, San Diego, Vol.I, pp. 109-116 (IEEE Computer Society Press)

[11] Ritter H., Kohonen T. (1989) Self-Organizing Semantic Maps. Biol. Cybern., in press

ANALYSIS OF EEG CHANGES BETWEEN FRONTAL AND OCCIPITAL AREA IN SPEAKING PROCESS

Gang Wang, * Morikuni Takigawa, Tomoyuki Miyazaki, Taisuke Takeishi
Department of Electronic Engineering, Kagoshima University
* Health Science Center, Kagoshima University
Mailing Address: No.65, Sengoku Building
6-11 Nishisengoku Cho, Kagoshima Shi, 892
Japan

There are various techniques to study the EEG activity over the scalp. Particularly, the methods of correlation and spectral analysis have been more and more widely used in EEG analysis to investigate functions and structures of the brain which generates multivariate time series signals. But the weakness of these methods is that the direction of the correlating influences between two areas can not be shown. The 'entropy analysis' method was developed for revealing the correlation not only in time but also in space. In our study, this method had been used for analyzing the inter-hemisphere activity between frontal and occipital area in speaking process.

METHODS

1. Entropy Analysis

For two time series,

$$X = X_{k-n}, \ldots, X_k, X_{k+1}, \ldots, X_{k+m}$$

$$Y = Y_{k-n}, \ldots, X_k, X_{k+1}, \ldots, X_{k+m}$$

where, X_{k-i}, Y_{k-i}, X_k, Y_k and X_{k+i}, Y_{k+i} denote the value of series X and Y in past, present and future respectively. With these notation, the generating model of time series X and Y shown in figure 1 had been established. If the series are approximated by Gaussian process, the information flow from X to Y can be expressed by the covariance matrix of the series. Thus,

$$I(W_{X_k} \rightarrow Y_{k+m}) = \frac{1}{2} \log \frac{|R(X^n Y^n Y_k Y_{k+m})|\ |R(X^n Y^n X_k Y_k)|}{|R(X^n Y^n Y_k)|\ |R(X^n Y^n X_k Y_k Y_{k+m})|}$$

$$= \frac{1}{2} \log \left[1 + \frac{hyx^{2}_{k+m,k}}{\sum_{i=0}^{m-1} \left(hyx^{2}_{k+m,k+m-i} + hyy^{2}_{k+m,k+m-i} \right)} \right]$$

where R(.) denotes covariance matrix and hyx denotes the impulse response from X to Y for lag m and time origin k. hyy is from Y to Y. The time series X, Y were fitted by two-dimensional AR model. It is known that the impulse response can be derived from the coefficient of the AR model directly.

2. EEG Recording and Computer Analysis

Eleven right-handed normal male subjects of age range 18-21 were studied. Electrodes were placed on F3, F4, 01 and 02 at standard 10-20 system, each referred to linked ear reference. The EEGs when the subjects saying a non-meaningful word and, in contrast with its, resting EEG were recorded. Via A/D converter, the EEG signals were digitized at 6 ms interval, and fitted by two-dimensional autoregressive (AR) model for each pair (F3 - 01 and F4 - 02). The order of model wasdetermined by Akaike's AIC criterion. Based on this model, the information flow in different direction were calculated by using eq.(1).

RESULTS

Figures 2 illustrates the information flow curve of one subject in different conditions. Means and standard deviations of information flow for 11 subjects are shown in table I.

In table I, the information flow in two opposite directions are

	F3 → 01	01 → F3	F4 → 02	02 → F4
RESTING	0.71±0.19	0.89±0.27	1.20±0.42	1.01±0.34
BEFORE SPEAKING	1.02±0.48	0.71±0.21	0.19±0.08	1.75±0.46
IN SPEAKING	0.14±0.05	1.82±0.67	0.42±0.19	1.61±0.53

Table I. Means and standard deviations of information flow (bit/sec)

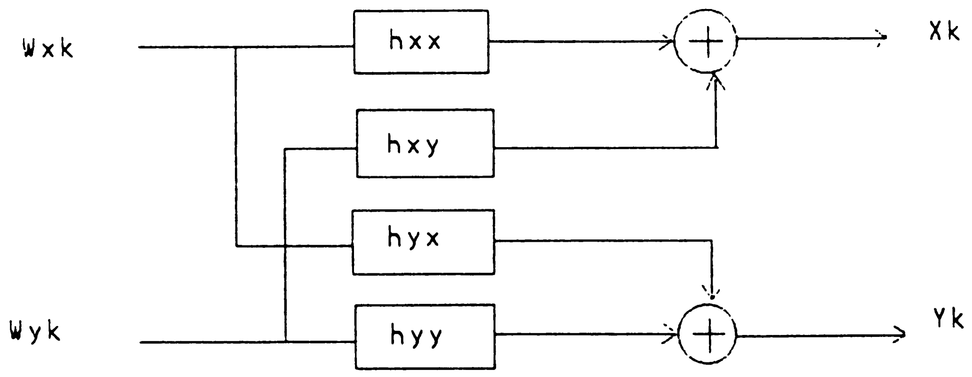


Fig.1 Generating model of time series X and Y. Wx, Wy denote the soure of series X and Y.

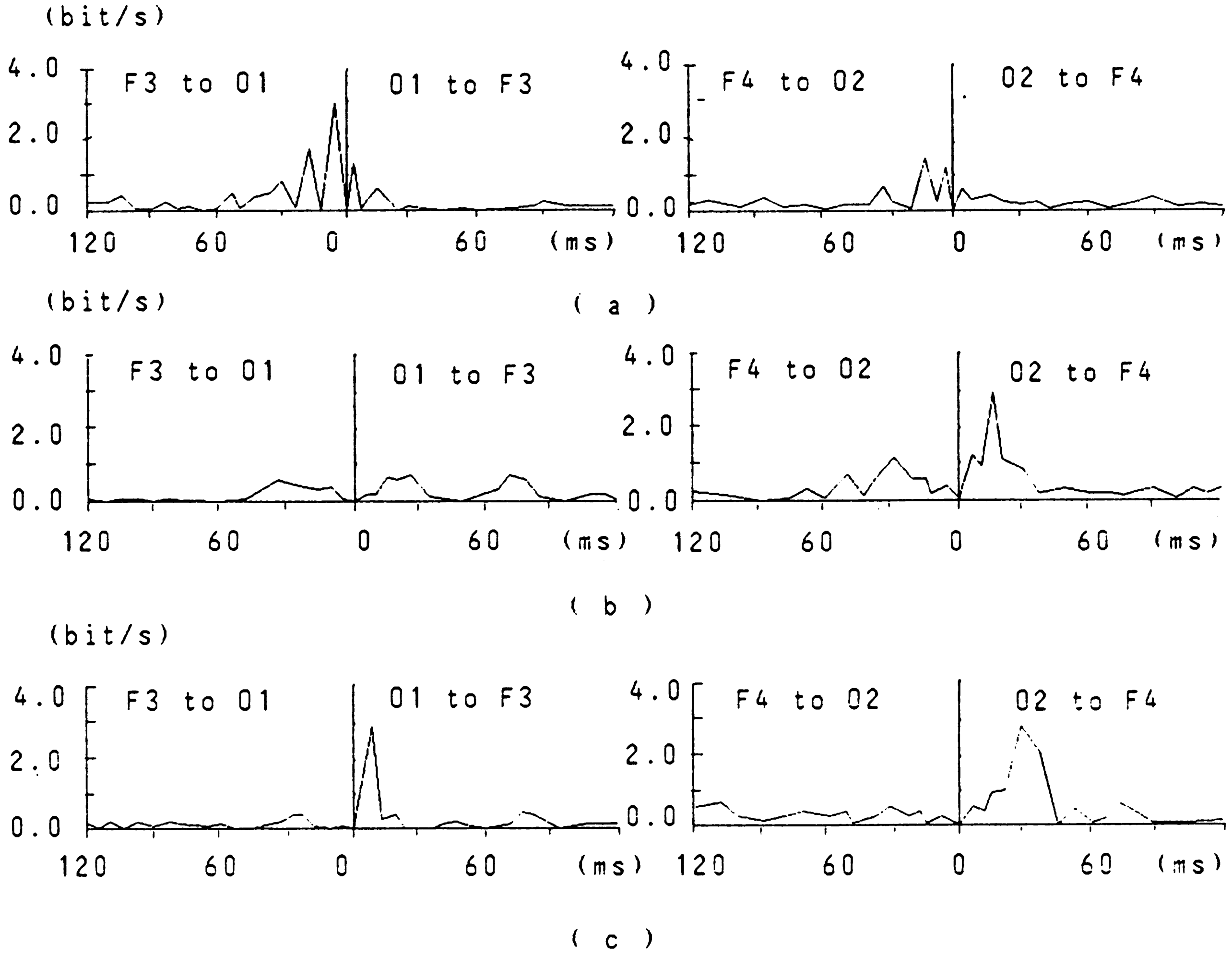


Fig. 2 Information flow curves in opposite directions between F3,01 and F4, 02. (a) resting (b) before speaking (c) in speaking

presented. To explicate the relations of its, mean differences were evaluated by unrelated t test. In resting, the information flow between F3 and O1, F4 and O2 had no significant differences. Before speaking, the dominant direction of the information flow was found in O2 $\rightarrow$ F4 direction ($P<0.02$), but not found between F3 and O1. In speaking, the dominant direction of the information flow was in posterior-anterior both left ($P<0.02$) and right hemisphere ($P<0.05$).

DISCUSSION

The dominant information flow was found in posterior-anterior direction in right hemisphere before speaking, but had not been found in left hemisphere. The information flow changes in right hemisphere were followed by the changes in left hemisphere between frontal and occipital area.

For right-handed normal subjects, it's usually the case that the left hemisphere is the 'talker' and 'doer', while the right hemisphere plays an important cognitive role in putting the left hemisphere's activities within their proper contexts. From the results of the experiment, the changes between occipital and frontal area in right hemisphere were prior to left hemisphere. In other words, when thesubjects want to say a word but the voice had not been out, there aresignificant information flow in occipital-frontal direction in right hemisphere. In temporal, the changes in left hemisphere is followed.

The physiological implications of the spread direction have not yet been established. However, it may suggest the existence of electro-physiological activities before and in speaking.

REFERENCES

1. Cook,N.D. The brain code. Methuen, London and New York, 1986
2. Inouye, T., Yagasaki, A., Takahashi, H. and Shinosaki, K. The dominant direction of interhemispheric EEG changes in linquistic process. Electroenceph. Clin. Neurophysiol., 1981,51:265-275
3. Saito, Y. and Harashima,H. Tracking of information within multichannel EEG record. In: N. Yamaguchi (Eds.), Recent Advances in EEG and EMG Data Processing. Elsever, Amsterdam, 1981:133-146

High-Order Bidirectional Associative Memory and Its Application to Frequency Classification

Chwan-Hwa Wu[*], Heng-Ming Tai[**], Chia-Jiu Wang[***] and Tai-Lang Jong[****]

[*] Department of Electrical Engineering, Auburn University, AL 36849
[**] Department of Electrical Engineering, University of Tulsa, Tulsa, OK 74104
[***] Department of Electrical and computer Engineering, University of Colorado at Colorado Springs, Colorado Springs, CO 80933-7150
[****] Department of Electrical Engineering, Texas Tech University, Lubbock, TX 79409

Abstract

A novel encoding scheme of a bidirectional associative memory (BAM) incorporating the high-order nonlinearity is proposed. This method significantly improves the storage capacity and error-correcting capability of the BAM. We discuss the memory storage capacity and the stability of the high-order bidirectional associative memory (HOBAM) in this paper. The application of the HOBAM to a frequency classifier is also presented.

I. Introduction

The bidirectional associative memory (BAM) introduced by Kosko [1,2] allows the associative search for stored stimulus-response pairs (X,Y). This type of neural network is a two-layer nonlinear feedback hierarchy of interconnected neurons and behaves as a heteroassociative memory. It has been pointed out that the BAM serves as a vital element in intelligent systems [3] and spectral signature recognition applications [4]. The storage capacity for perfect recalls of the BAM, however, is limited by the number of neurons in the network. Moreover, the condition of the continuity of association between pattern sets of the BAM must be satisfied in order to achieve reliable recalls. This requires that if the stored stimulus patterns are close, then the stored response patterns are close. Although there are techniques [5] available to improve its performance, the multiple training method requires interactive tuning.

It has been reported that neural networks with high-order nonlinearity show dramatic improvement in memory capacity and error-correcting capability in comparison with linear memory models [6-8]. In this paper, we propose a neural network for the BAM using high-order correlations. This high-order bidirectional associative memory (HOBAM) not only possesses merits of the BAM and the advantage of high-order strategy, but also relaxes the continuity assumption for reliable recalls. The storage capacity and stability of HOBAM are discussed in section II. An application of the HOBAM to the frequency classifications is reported in section III. The last section is the concluding remarks.

II. High-Order Bidirectional Associative Memory

For the high-order bidirectional associative memory, one cycle of the recall process of the kth-order model is represented by

$$Y(t+1) = \mathrm{sign}\left(\sum_{s=1}^{M} Y^s [X^{s\prime} X(t)]^k \right) \tag{1}$$

$$X(t+1) = \mathrm{sign}\left(\sum_{s=1}^{M} X^s [Y^{s\prime} Y(t+1)]^k \right) \tag{2}$$

where ' represents the transpose of a matrix. Assuming that X^s and Y^s are sets of randomly generated vectors, then signal to noise ratio of the X to Y mapping can be derived as [7],

$$\mathrm{SNR}\,\big|_{XY} = N_X \,/\, \{ (M-1)\,(2k)!\, N_X^k \,/\, (k!\, 2^k)\}^{1/2} \tag{3}$$

The estimated capacity M_X and M_Y are

$$M_X \approx \frac{N_X^k\, k!\, 2^k}{(2k)!\, (SNR\,|_{XY})^2} \text{ and } M_Y \approx \frac{N_Y^k\, k!\, 2^k}{(2k)!\, (SNR\,|_{YX})^2} \qquad (4)$$

To guarantee a successful evolution of the system, the storage capacity can be chosen as

$$M = MIN(\, M_X, M_Y) \qquad (5)$$

which possesses the capacity $O(\, MIN(N_X^k, N_Y^k))$.

Extensive computer simulations on the performance of the BAM and HOBAM have been carried out for $N_X = N_Y = 40$. The results are shown in Figs. 1-2. Each network is presented with a number of sets of randomly chosen association pairs and each set has M stored pairs. The successful recall probabilities vs. the number of stored pairs are recorded in Fig. 1, which indicates that the storage capacity of the HOBAM is higher. Fig. 2 illustrates the comparison of the error-correction capability. In the simulation, each point corresponds to 100 sets of 10 stored pairs. Each pair is degraded by the specified Hamming distance and used as the input pattern to the network. It is evident that HOBAM has a better error-correction capability.

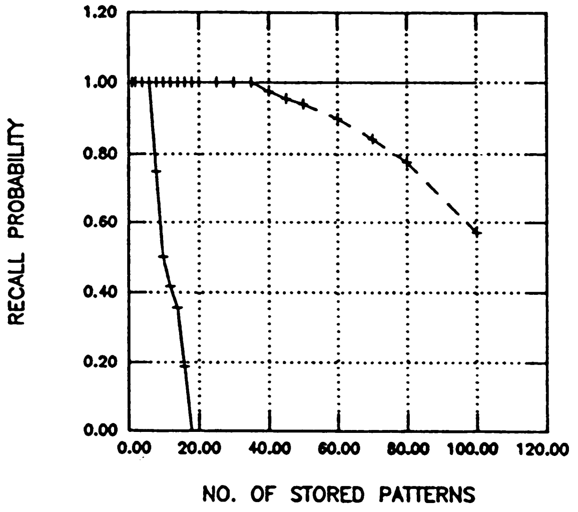


Fig. 1 Plot of the storage capacity of neural net models ($N_X = N_Y = 40$). The solid line with "-", dashed line with "+" and solid line without symbol correspond to BAM, 2nd-order and 3rd-order HOBAM, respectively.

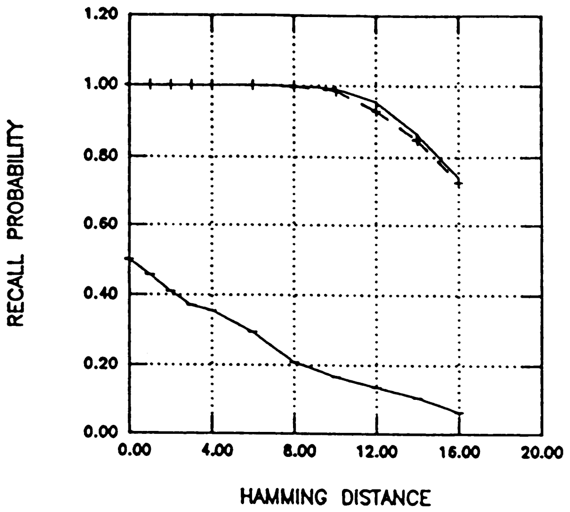


Fig.2 Improvement of the error-correction capability with the increase of the order.

Plot of correct recall probability vs. the amount of Hamming distance between the input pattern and stored pairs. Symbols are the same as those in Fig. 1.

We discuss the evolution of the system from an incomplete pair (X,Y) next. For a more clear description, we redefine the dynamical equations of HOBAM as

$$y_i(t+1) = \text{sign}\, (\sum_s y_i^s\, [\, \sum_j u_j^s\, u_j(t)]^k) \qquad (6)$$

$$x_i(t+1) = \text{sign}\, (\sum_s x_i^s\, [\, \sum_j v_j^s\, v_j(t+1)]^k) \qquad (7)$$

where $x_j = 2\, u_j - 1$ and $y_j = 2\, v_j - 1$. Let us define that

$$\alpha^s(t) = \sum_j u_j^s\, u_j(t) \qquad (8a)$$

$$\beta^s(t) = \sum_j v_j^s\, v_j(t) \qquad (8b)$$

The condition to ensure that the system to reach a stable equilibrium state is that there exists an α^a such that

$$\alpha^a > \alpha^s \text{ for all s which is not equal to a.} \tag{9}$$

We can always find a k such that

$$(\alpha^a)^k > \sum_{s \neq a} (\alpha^s)^k \tag{10}$$

As a result from Eq.(7), $y_i = y_i^a$. Therefore, $x_i = x_i^a$ form Eq.(8) due to the fact that $\beta^a = N_Y$.

The energy function of the system can be defined as

$$E = - \sum_i y_i \,\text{sign}(\sum_s y_i^s (\alpha^s)^k) - \sum_i x_i \,\text{sign}(\sum_s x_i^s (\beta^s)^k) \tag{11}$$

After a recall from X(0) → Y(1) → X(1),

$$E = - (N_X + N_Y). \tag{12}$$

The system converges to the global minimum energy state.

If there is no maximum α^a, i.e. two or more stored patterns have the same Hamming distance to the input pair (X,Y). This is the situation which human beings have to make a guess. The stability of the system for this situation is under investigation.

IV. Application: Frequency Classifier

A robust frequency classifier is desirable for the radar signal processing in that the signals are usually noise-corrupted. The HOBAM described above can be employed to classify the frequency of the signals mixed with noise. The inputs of the network are the digitized sinusoidal waves with Gaussian noise,

$$s(n) = \sin(2\pi fn/N) + G(n)$$

where G(n) is the Gaussian noise. We have chosen N = 16 in the experiment. The outputs, S(f), of the system are the Discrete Fourier Transforms of the inputs. The inputs and outputs are normalized in the range [-1,1]. Because the BAM is a binary neural network, the preprocessing of signals is required. The normalized signal is quantized into 2P levels. N-point digitized signals are transformed into 2P by N cells of a bipolar representation (each cell is +1 or -1) as the following procedure. If the nth signal is positive, we assign 1's to the cells in the nth-column between cell P (including cell P) and the cell where the signal is located, and -1 to the rest of the cells in the n-th column. Whereas the nth signal is negative, 1's are assigned to the cells between cell P + 1 (including cell P + 1) and the cell contains the signal; -1's are assigned to the rest of the cells in the same column. The stored patterns for X and Y are the bipolar representations of the sinusodial waves without noise and their Fourier Transforms, respectively, for f = 1,2,...7.

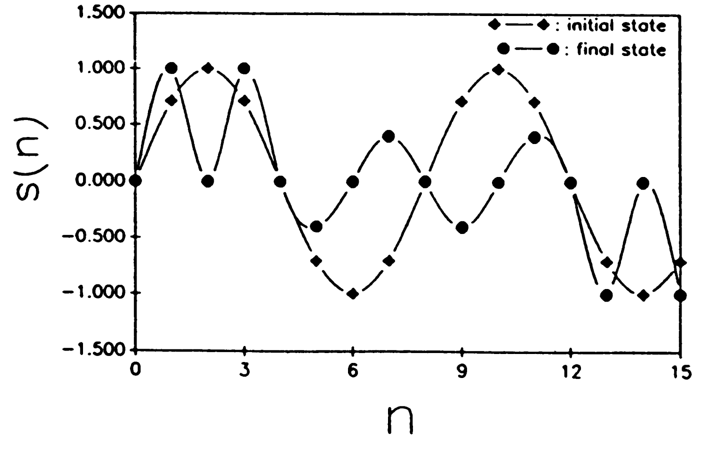


Fig. 3(a) The initial and final state, s(n), of BAM.

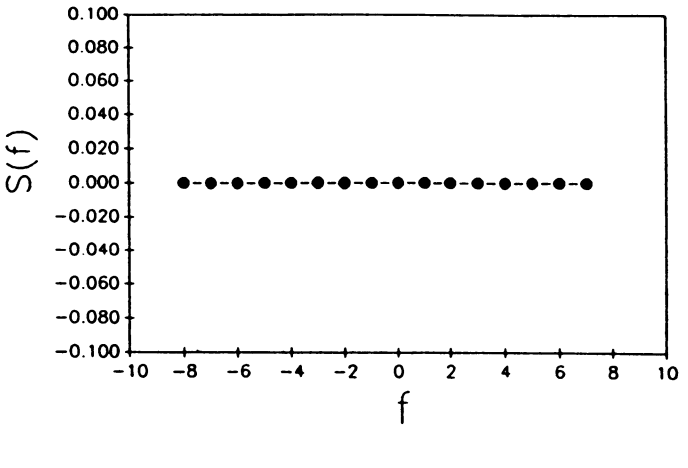


Fig. 3(b) The final state, S(f), of BAM.

We have experimented the BAM by assigning the sinusoidal waves without noise to X and leave Y blank (-1 in every cell). The BAM can not recall any pair of the patterns when we train more than two patterns to the network. As shown in Fig. 3, the input to X is the sinusoidal wave (f = 2) and the system converges to a spurious state. For the same kind of test, the HOBAM has demonstrated the capability to recall any pair of patterns without noise when there are seven pairs of patterns stored. For the situation that the Gaussian noise presents, we have experimented the HOBAM for the SNR up to 1. Every X to Y mapping is successful for 5-th order or higher HOBAM's. The results of f = 4 and SNR = 1 are plotted in Fig. 4 for an 8-th order HOBAM. The robust classification indicates that the HOBAM is capable of making an accurate decision even when the noise energy is significantly large compared to that of the signal.

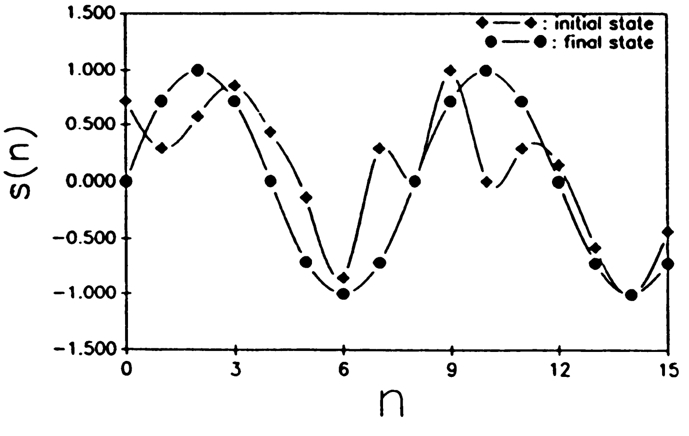


Fig. 4(a) The initial and final state, s(n), of HOBAM.

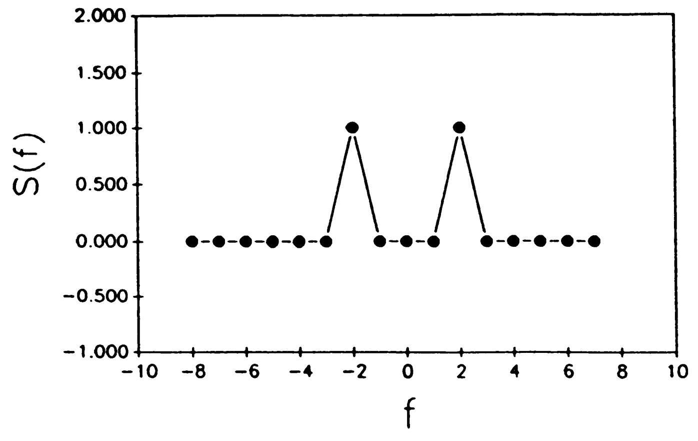


Fig 4(b) The final state, S(f), of HOBAM.

IV. Concluding Remarks

In this paper, we present the high-order bidirectional associative memory which significantly enhances the BAM performance on both forward and backward information flows. Simulation results demonstrate that the proposed neural networks increase the memory capacity and improve the error-correction capability. In addition, the condition that the system can reach the global minimum energy state is discussed. The application of a frequency classifier has shown promising results to recognize signals even when the noise energy is considerably large.

References

1. B. Kosko, "Adaptive bidirectional associative memory," *Appl. opt.* **26**, 1987, pp. 4947-4960.
2. B. Kosko, "Bidirectional associative memory," *IEEE Trans. SMC* , **18**, 1988, pp. 49-60.
3. J. B. Cruz, and A. R. Stubberud, "Intelligent control of variably configured systems using neural networks," Proc. *IEEE TENCON 87*, 1987, pp. 893-898.
4. G. Mathai and B. R. Upadhyaya, "Performance analysis and application of the bidirectional associative memory to industrial spectral signatures," *Proc. IJCNN*, 1989, pp. I-33-I-37.
5. Y. F. Wang, J. B. Cruz and J. H. Mulligan Jr., "An enhanced bidirectional memory," *ibid*, pp. I-105-I-110.
6. H. H. Chen et al., "High order correlation model for associative memory," *Proc. AIP conf. Neural networks for computing,"* 1986, pp. 86-99.
7. H. M. Tai and T. L. Jong, "Neural networks with high-order nonlinearity," *Electron. Lett.* **24**, 1988, pp. 1225-1226.
8. D. Psaltis et al., "High order associative memories and their optical implementations," *Neural Networks* **1**, 1988, pp. 149-163.

A Neural Net Editor with Biological Applications

Vahe Bedian
Department of Biology
James F. Lynch and Fengman Zhang
Department of Mathematics and Computer Science
Clarkson University
Potsdam, N.Y. 13676

July 22, 1989

Constructing an accurate model of the nervous system of an organism, or a portion of it, is a difficult task. Single neurons are complex and show a variety of behaviors, and networks of such neurons become almost intractable. The experimental data on which the model rests - connections between neurons, types of synapses, thresholds, and so on - is often imprecise or incomplete. A computer simulation of the neural net can be a very useful tool. It can help to identify essential parameters by comparing the behavior of the simulation with the observed behavior of the organism. In the same way, it can also help to fill in missing information.

For these reasons, we developed a simulator to model relatively simple, well understood nervous systems and behaviors of organisms such as the molluscs *Aplysia*, *Hermissenda*, and *Pleurobranchaea*. In [1], we applied the simulator to a model of the rhythmic feeding and choice behavior observed in *Pleurobranchaea* [5]. The network simulator is constructed around a biophysically realistic neuromime capable of representing major electrical neuronal properties, such as threshold, time constant, absolute and relative refractory periods, and accommodation.

We designed the simulation program to be as independent as possible from the neural net that we were modeling. The specifications of the network were contained in tables in an external text file that was read by the program. A text editor was used to make modifications to the network. Nevertheless, even with a table-driven simulator and an interactive text editor, the procedure was tedious. Also, there was no systematic way to do sensitivity analysis by performing a search through the parameter space. Thus, we devoted considerable effort to the design and implementation of a special-purpose editor that allows rapid modification and testing of networks.

The editor is a hierarchical, menu-driven system (see Figure 1). At every step, the user responds to a prompt from the editor. All the responses are brief: single keystrokes followed by a RETURN for menu options and yes/no queries, or short character strings for neuron numbers and names, experiment labels, and file names. Most of the prompts are self-explanatory, but there is an extensive HELP function available. In a number of situations, the user will want to issue a command repeatedly, for instance, the ADD NEURON command. The editor uses the convention that such a command is repeated until the user enters a null line, i.e. a line with RETURN as the first character. Then the editor returns to the menu which invoked the command.

Each user response activates a new menu or performs some operation on a network file. The logical structure of a network file is similar to the structure of the files manipulated by combinatorial graph editing systems such as CABRI [3] and TYGES [4]. The two main categories of data in those systems are node parameters and edge parameters. In our system, the analogous categories are neuron parameters and connection parameters.

The neuron parameters are:

Threshold Voltage
Peak Spike Voltage
Accommodation Rate
Recovery from Accommodation
Tonic Input
Time Constant
Threshold for Opening Calcium Channels

The connection parameters are:

Synaptic Delay
Chemical Synapse Strength
Non-rectifying Electrotonic Coupling
Rectifying Electrotonic Coupling

The network file also contains a description of the experiments to be run. Each experiment applies input stimuli to selected neurons for a specified length of time.

All neuron parameters have default values, which are saved in a file. All connection parameters have default value 0. When the CREATE NETWORK command is invoked, the user has the option of overriding any of the neuron parameter default values. The changes may then be saved by writing them to the default file, or they may take effect only for the duration of the command. The user then specifies the number of neurons in the network, their names, and the number of experiments. The new network consists of a homogeneous collection of neurons, all with the current default values and no connections. The EDIT menu is then entered to allow the user to change neuron parameters and add connections via the MODIFY NEURON and MODIFY CONNECTION commands. Alternatively, the user may create a network with 0 neurons, and then repeatedly use the ADD NEURON command to generate the neurons. ADD NEURON also allows the user to copy parameters from the default file or from some existing neuron.

When the user is satisfied with the network, the simulator may be invoked by the RUN SIMULATOR command. Various options are allowed. A sequence of experiments may be run, where the input stimuli to the neurons are varied. Systematic search of parameter space is possible by specifying a parameter to be varied, its initial and final values, and its increment value. With this option, the simulator runs repeatedly, varying the selected parameter after each run. Output choices are also given. Different neurons can be selected for plotting their voltage levels as a function of time. Firing frequencies can be plotted as a function of a parameter.

The effort expended on development of the editor has amply repaid itself. Using the editor, we performed a more extensive and systematic simulation of oscillator networks. Some of the graphs for one and two neuron systems are shown in Figure 2. The complete results for these and larger networks are reported in [2]. We anticipate further use of the editor. Modeling networks of other reasonably well-understood organisms is an obvious possibility. We also intend to study models of learning in such organisms. This would require relatively minor additions to the editor and simulator.

References

[1] V. Bedian, F. Zhang, J. F. Lynch, and M. H. Roberts, A Neural Network Model with Applications to *Pleurobranchaea*, *Soc. Neurosci. Abstr.*, vol 14 (1988), 259.

[2] ____________, A Neuronal Model with Network Simulation of *Pleurobranchaea* Feeding and Choice Behavior, submitted to *J. Theor. Biol.*

[3] CABRI, an Interactive System for Graph Manipulation, in *Graph-Theoretic Concepts in Computer Science*, G. Tinhofer and G. Schmidt (eds), *Lecture Notes in Computer Science*, vol. 246, Springer-Verlag (1987), 58-67.

[4] J. Hynd and P. D. Eades, The Typed Graph Editing System - TYGES, *Proc. 3rd Australasian Conf. on Computer Graphics*, Brisbane, Australia (1985), 15-19.

[5] W. J. Davis, Neural Mechanisms of Behavioral Plasticity in an Invertebrate Model System, in *Model Neural Networks and Behavior*, A. I. Selverston (ed.), Plenum Press (1985), 263-282.

NEURAL NET EDITOR

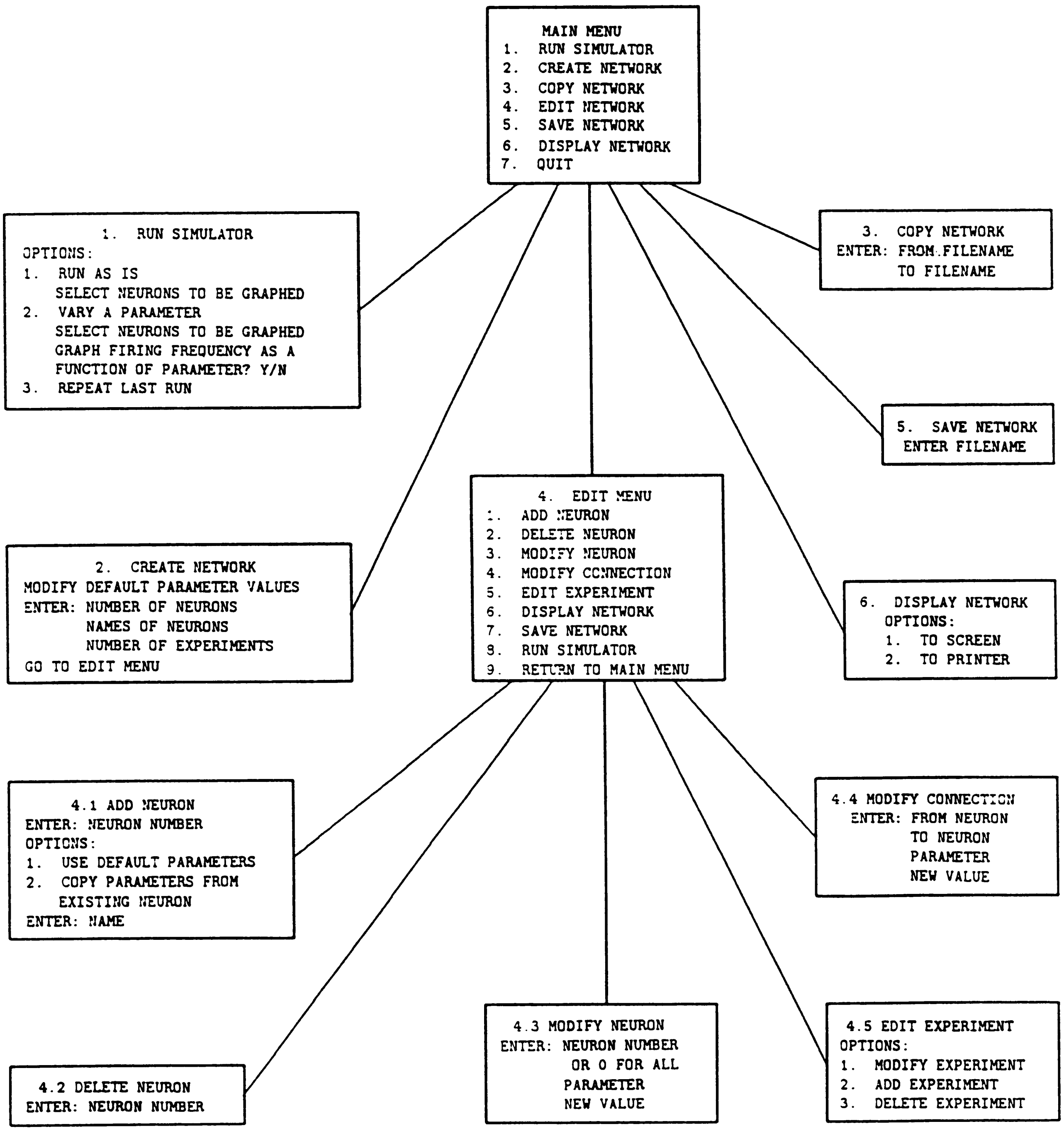
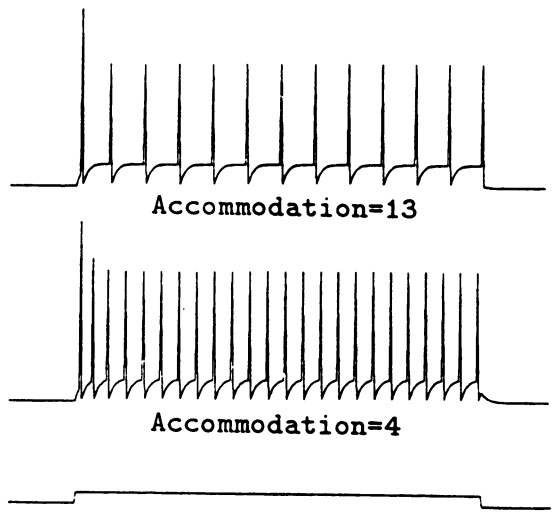
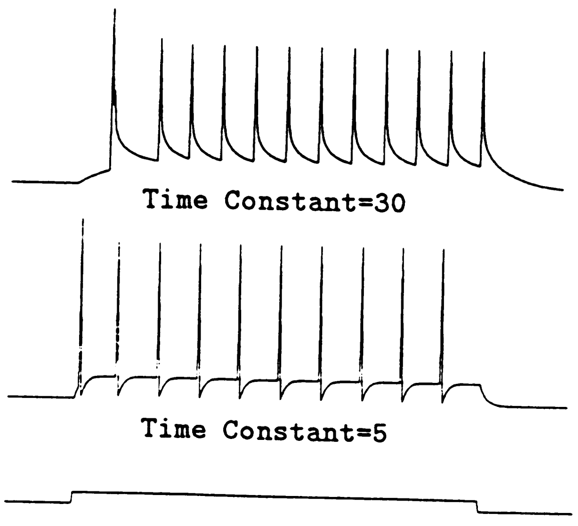


FIGURE 1.



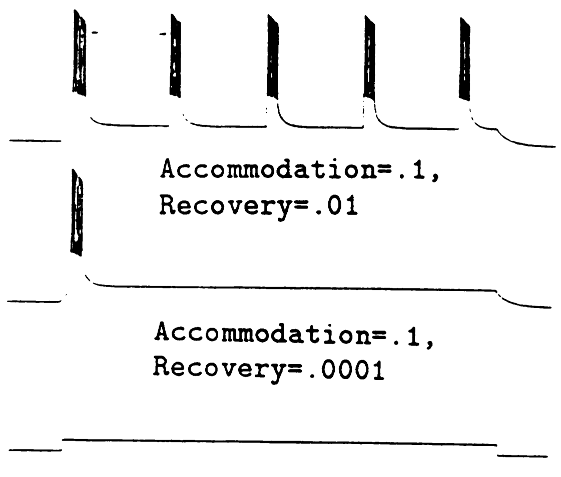
Input

1. Single Neuron, Tonic Input
Effect of Change in
Accommodation



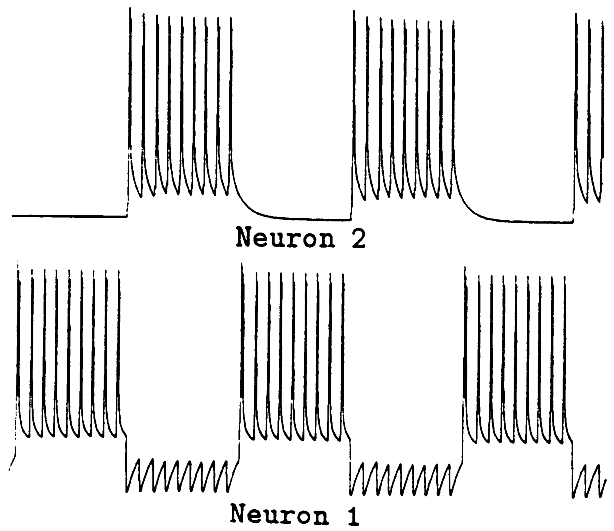
Input

2. Single Neuron, Tonic Input
Effect of Change in
Time Constant



Input

3. Single Neuron, Tonic Input
On Response and Bursting
Due to Low Accommodation
Rate and Recovery



4. Two Neuron Oscillator:

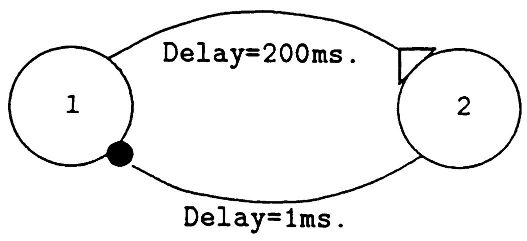


FIGURE 2.

Using Classifier Systems to Implement Distributed Representations

Lashon B. Booker

Navy Center for Applied Research in AI
Naval Research Laboratory, Code 5510
Washington, D.C. 20375

1. Introduction

Genetic algorithms and classifier systems have proven to be useful tools for implementing flexible problem solving behavior, learning and search in symbolic systems (Goldberg and Holland, 1988). Because of the robust statistical nature of these techniques, it is not surprising that they have also been applied to problems involving neural network architectures. One type of application uses genetic algorithms to efficiently optimize the weights in a neural network. Montana and Davis (1989), for example, show how a genetic algorithm outperforms backpropagation when training a feedforward network on data from a sonar image classification problem. Another type of application uses genetic algorithms to evolve neural network architectures whose design is especially suited for a given task (Harp, Samad, and Guha, 1989; Miller, Todd, and Hegde, 1989).

In addition to work showing how genetic algorithms can be applied to neural networks, there has also been research exploring the relationships between neural networks and classifier systems. Much of this research tries to establish some kind of functional equivalence between layered, feedforward neural networks and specialized variants of classifier systems (Belew and Gherrity, 1989; Davis, 1988, 1989). The value of this work is that it highlights opportunities to transfer techniques that have proven useful in one framework into systems based on the other framework.

This paper shows how the distributed representation techniques used in neural networks and other connectionist systems have a natural counterpart in classifier systems. Eventually, this representational correspondence may have important practical implications for the parsimonious design of hybrid systems having both subsymbolic and symbolic capabilities.

2. Distributed Representations in Neural Networks

A distributed representation is one in which many representational units participate in the representation of a single entity, and every unit helps to represent many different entities (Hinton, McClelland, and Rumelhart, 1986). This kind of representation has been closely identified with neural networks and connectionism because it endows networks of simple computational elements with some important information processing capabilities. Among these capabilities are:

- Content addressable memory that allows for a "best fit" retrieval given only a partial description of the desired item.
- Automatic achievement of certain generalizations, since modifications to the constituent units of one representation automatically affect all similar representations that share those units.
- Incremental storage of new concepts by generating new combinations of existing representational units, thereby avoiding the need for any new units.
- Efficient use of network processing capacity through the judicious use of *coarse coding* techniques.

While distributed representations have many useful properties, they also have certain disadvantages (Feldman, 1988). Most of these shortcomings are related to the difficulties of representing non-arbitrary structure within and among distributed representations. Nevertheless, distributed representations have important implications for the symbolic/connectionist debate about the nature of mental structures and processes (Smolensky, 1988).

Neither connectionism nor traditional artificial intelligence has provided a completely satisfactory account of intelligent behavior. Symbolic systems are excellent at representing and logically reasoning about complex knowledge structures. They have been correspondingly weak at capturing the flexible aspects of cognition such as robust learning, recognizing natural categories, and formulating non-brittle problem solving strategies. Neural networks and connectionist architectures, on the other hand, excel at learning and pattern recognition while they do not yet offer much in the way of structured representations and inferences. There is a clear need for hybrid systems that offer the advantages of both the symbolic and connectionist paradigms (Belew and Forrest, 1988).

One approach to designing such a hybrid is to start with the subsymbolic capabilities of distributed representations, and find ways to implement these capabilities using representational units that facilitate symbolic descriptions. The essence of the distributed representation idea in neural networks is the use of numeric weight vectors as epistemic primitives, along with methodically designed input encodings, network organizations, and statistical techniques to extract patterns and map inputs into outputs (Smolensky, 1988). High-level symbolic properties are viewed as an emergent phenomenon of the dynamics of the low level representational primitives. The classifier system framework takes a similar view of how high level capabilities emerge from low level constructs, but with some important differences. In the next section we indicate how classifier systems can be used to implement distributed representations.

3. Distributed Representations in Classifier Systems

Classifier systems offer a framework for problem solving and learning designed to provide flexible representational capabilities at the basic level of concepts, relations, and the way they are organized (Holland, 1986; Booker, Goldberg, and Holland, in press). The flexibility of classifier systems is achieved by using several mechanisms, including

(1) A simple syntax for rules or *classifiers* that facilitates learning without a crippling loss of descriptive power.

(2) Pattern-directed inferences that generalize over the variations in specific input configurations.

(3) Parallel activation of several classifiers on every cycle, treating activated classifier as a set of competing hypotheses whose credibility can be evaluated using a local credit assignment scheme (the *bucket brigade*).

(4) A statistical induction scheme (the *genetic algorithm*) that proposes new rules to improve upon current capabilities to categorize inputs and solve problems.

Perhaps the most important factor contributing to the flexibility of classifier systems is the use of parallelism and recombination as intrinsic properties of all representations. All representations are constructed from a set of constituent elements or *building blocks*. Because the syntax of the representation language is simple, recombining these building blocks can be done with local syntactic manipulations that avoid the need for complex symbolic interpreters or knowledge-intensive supervision.

The inherently distributed nature of representations in classifier systems leads rather directly to the kinds of distributed representations found in neural networks once we specify the building blocks to be used as basic representational units. There are several kinds of building blocks available in classifier systems. At the lowest epistemic level are the building blocks present in the bit strings that encode all input, output, and internal messages. The semantics of these bit-level building blocks depend on the nature of the encoding, and several encoding schemes are available:

- Simple feature lists where each bit designates a single feature-value pair or unary predicate. Combinations of these building blocks denote the presence or absence of elements belonging to some subset of features.
- Simple numeric encodings where a string of bits denotes a numeric attribute or the numeric encoding of a symbolic attribute. The building blocks here designate hyperplanes in the space of legal numeric values.
- Ordinal feature manifolds (Hayes-Roth, 1976) where each bit denotes some value of an ordinal attribute, and values are coarsely encoded to allow graded comparisons. The building blocks in these strings designate continuous ranges of ordinal values.
- Complementary feature manifolds (Hayes-Roth, 1976) where each bit designates the absence of some value of a nominal attribute. These building blocks denote disjunctive sets of nominal values.

This variety of feature encodings gives considerable representational power to individual classifiers. These encodings support generalizations in the activating condition of a classifier that represent critical sets of features, intervals of ordinal values, values of tree-structured attributes, and simple disjuncts of nominal values. This set of capabilities corresponds to the primitive language constructs used in most symbolic learning paradigms. It is misleading, therefore, to characterize bit string representations as inherently weak and "low-level".

At the next level of representation, messages are building blocks for representing associations among rules and concepts. Because the message language has a simple syntax, associations among rules can be constructed and modified with local syntactic manipulations that avoid the need for complex interpreters or knowledge-intensive critics. We can extend this notion even further by assigning each classifier an address or *tag*. Tags can be given semantics in terms of distance from an input interface, and position relevant to a coordinate system that is meaningful to that interface. By identifying tags with semantic content in this way, we impose a hierarchical organization on classifier representations and we can use tags to identify relationships within and between levels in a hierarchy, and relationships between hierarchies. In principle, this makes classifier systems a very powerful framework for representing knowledge. For example, it has been shown (Forrest, 1985) that the knowledge contained in a standard semantic network description (eg. KL-ONE or NETL) can be mapped into a set of classifiers that support the same information retrieval operations. It remains to be shown that these kinds of complex knowledge structures can be learned by a classifier system as a results of its experiences in an environment.

Finally, we can view rules themselves as building blocks for representing complex concepts, constraints, and problem solving behaviors (Holland, Holyoak, Nisbett, and Thagard, 1986). Rather than construct a syntactically complex representation of a symbolic concept that would be difficult to use or modify, a classifier system uses groups of syntactically simple rules as the representation. The structure of the concept is *modeled* by the organization, variability, and credibility of the constituent rules. Because the members of a group compete to become active, the appropriate aspects of the representation are selected only when they are relevant in a given problem solving context. The modularity of the concept thereby makes it easier to use as well as easier to modify. Because rules are activated in parallel, new combinations of existing rules and rule clusters can be used to dynamically represent novel situations. Simple capabilities along these lines have been demonstrated in studies showing how classifier systems can learn an internal model of their environment and how to function in it (Booker, 1988).

4. Discussion

Both neural networks and classifier systems use a collection of basic computing elements as epistemic building blocks. Classifier systems use condition-action rules that interact by passing binary messages. Connectionist systems use simple processing units that send excitatory and inhibitory signals to each other. Concepts are represented in both systems by the distributed, simultaneous activation of several computing elements. There are important differences between classifier systems and connectionist systems, however, that stem primarily from the properties of the building blocks they use. The interactions among computing elements in a connectionist system make "best-fit" searches a primitive operation. The same capability is achieved in a classifier system using messages and tags to link related rules together. A directed spreading activation process is then required to efficiently retrieve the appropriate concept. Other differences relate to the way inductions are achieved. Modification of connection strengths is the only inductive mechanism available in most connectionist systems. Moreover, the procedures for updating strength are part of the initial system design that cannot be changed except perhaps by tuning a few parameters. Classifier systems, on the other hand, permit a broad spectrum of inductive mechanisms ranging from strength adjustments to analogies. In principle, many of these mechanisms can be controlled by or easily expressed in terms of inferential rules. These inferential rules can be evaluated, modified and used to build higher level concepts just like any other building blocks.

Classifier systems are therefore like connectionist systems in that they acknowledge the importance of microstructure, multiple constraints, and the emergence of complex phenomena from simple interactions. But because classifier systems use rules as a basic epistemic unit, they avoid having to reduce all knowledge to a set of connection strengths. This makes it possible to enjoy the advantages of distributed representations, while at the same time providing a non-trivial capability to represent structure within and between complex concepts. Classifier systems therefore have the potential to occupy an important middle ground between the symbolic and connectionist paradigms. In order to realize this potential, more work must be done to understand how classifier systems can dynamically construct and modify the kinds of distributed representations described here. The research completed to date on small systems and simple problems is a very promising first step toward that goal.

REFERENCES

Belew, R. and Forrest, S. (1988). Learning and programming in classifier systems. *Machine Learning, 3*, 193-223.

Belew, R. and Gherrity, M. (1989). Back propagation for the classifier system. *Proceedings of the Third International Conference on Genetic Algorithms*, p. 275-281, Fairfax, VA: Morgan Kaufmann.

Booker, L.B. (1988). Classifier systems that learn internal world models. *Machine Learning, 3*, 161-192.

Booker, L., Goldberg, D., and Holland, J. (in press). Classifier Systems and Genetic Algorithms. To appear in

Artificial Intelligence.

Davis, L. (1988). Mapping classifier systems into neural networks. *Proceedings of the 1988 Conference on Neural Information Processing Systems*, Denver, CO: Morgan Kaufmann.

Davis, L. (1989). Mapping neural networks into classifier systems. *Proceedings of the Third International Conference on Genetic Algorithms*, p. 375-378, Fairfax, VA: Morgan Kaufmann.

Feldman, J.A. (1988). Connectionist representation of concepts. In D. Waltz and J. Feldman (Eds), *Connectionist models and their implications: Readings from cognitive science*, Norwood, NJ: Ablex.

Forrest, S. (1985). A study of parallelism in the classifier system and its application to classification in KL-ONE semantic networks. Ph.D. dissertation, University of Michigan, Ann Arbor.

Goldberg, D. and Holland, J. (1988). Genetic algorithms and machine learning. *Machine Learning, 3*, 95-99.

Harp, S., Samad, T., and Guha, A. (1989). Towards the genetic synthesis of neural networks. *Proceedings of the Third International Conference on Genetic Algorithms*, p. 360-369, Fairfax, VA: Morgan Kaufmann.

Hayes-Roth, R. (1976). Patterns of induction and associated knowledge acquisition algorithms. In C.H. Chen (Ed.), *Pattern recognition and artificial intelligence*, New York: Academic Press.

Hinton, G., McClelland, J., and Rumelhart, D. (1986). Distributed representations. In D. Rumelhart and J. McClelland (Eds), *Parallel distributed processing: Explorations in the microstructure of cognition. Volume 1: Foundations*, Cambridge, MA: MIT Press.

Holland, J.H. (1986). Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. In R. Michalski, J. Carbonell, and T. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach* Volume 2. Los Altos, CA: Morgan Kaufmann.

Holland, J., Holyoak, K., Nisbett, R., and Thagard, P. (1986), *Induction: Processes of Inference, Learning and Discovery*, MIT Press, Cambridge, MA.

Miller, G., Todd, P., and Hegde, S. (1989). Designing neural networks using genetic algorithms. *Proceedings of the Third International Conference on Genetic Algorithms*, p. 379-384, Fairfax, VA: Morgan Kaufmann.

Montana, D. and Davis, L. (1989). Training feedforward neural networks using genetic algorithms. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI: Morgan Kaufmann.

Smolensky, P. (1988). The constituent structure of connectionist mental states: A reply to Fodor and Pylyshyn. Technical report CU-CS-394-88, Department of Computer Science, University of Colorado at Boulder, Boulder, CO.

Short-term memory capacity limitations in recurrent speech production and perception networks

Gordon D. A. Brown

Department of Psychology
University College of North Wales
Bangor LL57 2DG
Gwynedd, United Kingdom
EARN/BITNET: RØØ1@vaxa.complab.bangor.ac.uk

Introduction

Cognitive psychologists generally assume that there exists some limited-capacity speech based temporary store that is involved in a wide range of other cognitive tasks such as mental arithmetic, reasoning and counting (see Baddeley, 1986, for review). This short-term memory has been extensively investigated by psychologists. While recent neural net modelling work has often been concerned with short-term memory phenomena, this work has not always been intended to account for the same phenomena as psychologists have been interested in (but see Grossberg, 1986).

Shrater and Pfeifer (1989) show that a simple localist architecture can give rise to a psychologically realistic serial position curve with both primacy and recency portions, and that the architecture gives rise to human-like performance with variations in rate of presentation etc. In their model the primary simulation result is a separation of long-term from short-term portions of the curve; the model is not intended to reveal the internal structure of the short-term store itself. Such recency effects, involving an increased probability of recall of the last few items in a supra-span sequence of material to be remembered, have traditionally been taken by psychologists to reflect the existence of a short-term memory storage system. Schneider and Detwéiler (1987) report a detailed connectionist framework for modelling short-term memory processes, and consider at length the various different mechanisms thought to underlie short-term memory performance. The perspective we adopt in the present paper is rather narrower, in that we focus just on the "articulatory loop" subcomponent of STM. Other approaches to modelling STM phenomena are represented by, e.g., Nolfi & Parisi (1987); Grossberg (1988), and Touretzky & Hinton (1989). These contrasting approaches vary in the timescale of phenomena that they are intended to account for, but none are aimed primarily at accounting for the data concerning effects on memory capacity of, e.g., phonemic confusability, articulatory suppression, and word length.

Several models of memory rely on recurrent nets. There have been considerable recent advances in the ability of connectionist models to account for temporal phenomena in plausible ways (e.g. Elman, 1988; Norris, 1989; and see papers in Touretzky, Hinton & Sejnowski, 1989). For present purposes, the central general feature of these recurrent networks is that any new input to a net must be accompanied by some context-giving representation of the recent outputs of the net. Thus some short-term memory capacity is an intrinsic part of any recurrent net just because it has to represent (in its state or context units) some decaying function of the recent history of the network. Under certain circumstances, it is possible to reconstruct from the state units the previous outputs of the network. The model described below makes use of this fact to show how short-term auditory-verbal memory capacity arises as a necessary side-effect of the mechanisms already provided for speech production and speech perception.

The model architecture

The model depicted in Figure One incorporates both a sequence production (left side of figure) and a sequence perception (right side of figure) net. In the present version of the model, there are 20 nodes in each oval drawn in Figure One- thus for each of the perception and the production net there are 20 input/plan units, 20 context/state units, 20 hidden units and 20 output units. Each input and output unit in the present model is taken to represent a single phoneme. At the top of the diagram, the rectangular (unimplemented) boxes represent relatively peripheral process that would translate a phonemic level of representation into articulatory output or, on the perception side, interpret acoustic input categorically. (We ignore here the considerable controversy about whether and how such processing might take place.)

The model is run in two distinct phases. In the first phase, the learning phase, the perception net and the production net learn to recognize and produce the same set of sequences. At present the nets are trained with a small (32) vocabulary of items which vary in length from 4 to 8 phonemes, with the phonemes being drawn from the limited pool (20) with which the model currently operates. The items are constructed in such a way that some items have many phonemes in common with other items (are phonemically confusable). At the end of this training procedure, the two networks have "long-term memory" about the sequences of phonemes that comprise the words in

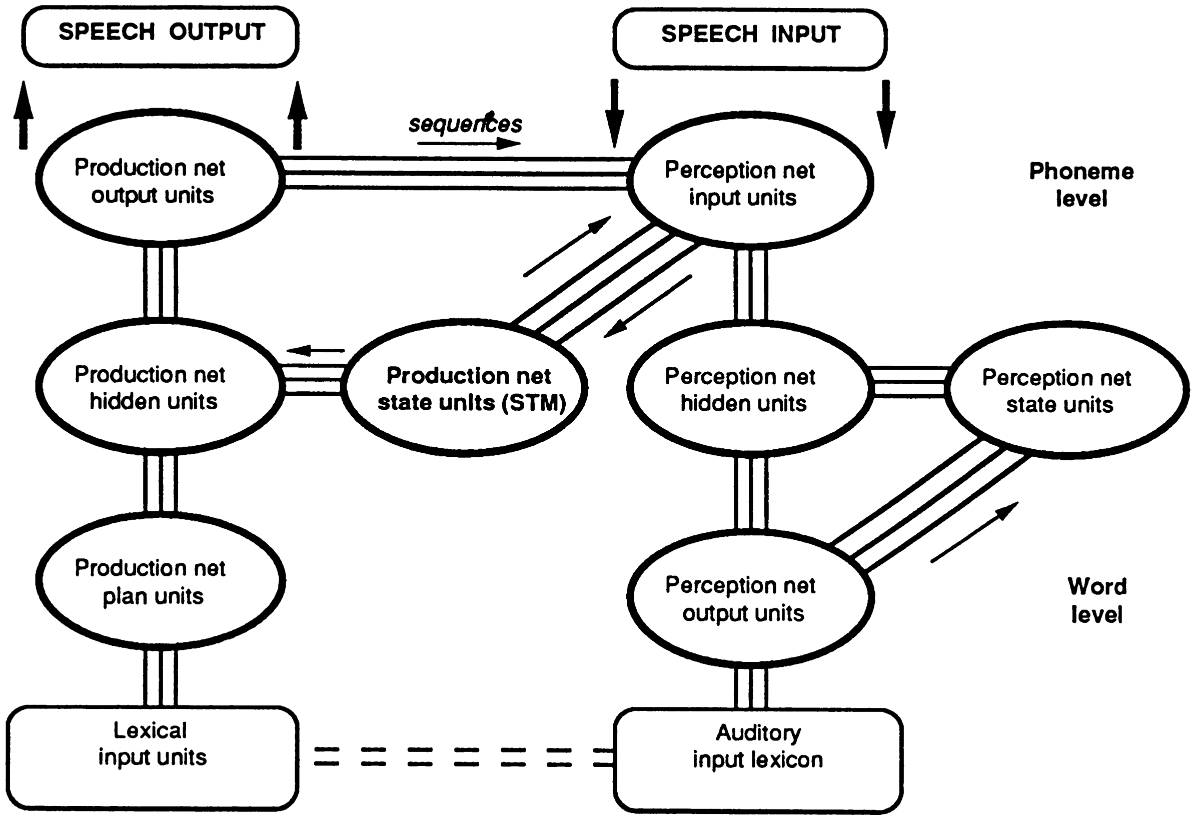

SPEECH OUTPUT
SPEECH INPUT
sequences
Production net output units
Perception net input units
Phoneme level
Production net hidden units
Production net state units (STM)
Perception net hidden units
Perception net state units
Production net plan units
Perception net output units
Word level
Lexical input units
Auditory input lexicon

Figure One

the vocabulary, and the production net can produce a sequence of outputs in response to a constant input on its plan units while the perception net can recognize that same sequential input and produce as output a whole-word representation that corresponds to the original input to the production network. This information remains unaltered during the second stage of the simulation.

Note that during the processing of each word, sequential output from the production net is copied over, a phoneme at a time, to act as potential input to the perception net. This means that the input to the production network's state units can be provided by the perception net input units instead of (as would normally be the case) by the production net output units. Whenever a new output is produced, the state units are set on the following processing cycle to be equal to that output plus (μ * the previous activation of the state units). In obtaining the simulation results reported below, μ was set at 0.7 throughout. Thus the state units for the sequence production net, which are necessary to provide a temporal context for the network, also potentially provide a short-term memory for any sequence of phonemes input to the speech perception apparatus. We note that this predicts certain patterns of interference between language production, language perception and short-term memory; at least some of these patterns of interference are in fact observed. (During the actual simulations, it proved necessary to include two sets of state units, both receiving identical input from the perception net input units: one set acted as the STM, and the other acted as normal state units. This was for the simulation-specific reasons that (a) some of the short items were so phonemically similar that they could not be distinguished without a between-word zero-ing of the state units, and (b) the simulations required experimentation with different values of μ, and for computational resource reasons it was not possible to relearn the long-term sequence production connections for each different value of μ. Other work suggests that these are implementation-specific limitations).

The second phase of the simulation involves the retention in STM of a sequence of words (each representing a sequence of phonemes). The input of the item sequence involves the clamping-on of sets of lexical input nodes in sequence; this is analogous to the presentation of a sequence of words. For the time it is clamped on, each word in the input sequence acts as input to the "production" network, emerging as a temporal output sequence, and this sequence is then input to the sequence "recognition" net and into the state units/short-term memory. For short-term recall to take place, the sequence of phonemes must be reconstructed to enable either recall or subvocal rehearsal of the material to be remembered. The reconstruction works in the following manner. To reconstruct the perceived sequence of phonemes from the final activations on the state units, it is necessary to cycle backwards through the following procedure:

(i) Assume that the most highly activated state unit represents the last-output phoneme.

(ii) Reconstruct what the levels of activation on the state units must have been on the previous time-slice if the estimate of the last-seen phoneme is correct by:

(iii) Subtracting unit activation from the last-identified phoneme, and

(iv) Dividing all state activations by μ.

(v) Repeat the procedure.

For this to work, it is necessary that output units should all be thresholded to either zero or one before being copied into the state units. This was done in all our simulations with this architecture. Provided this condition is satisfied, the method above will be able to reconstruct, from the state units, the sequence of past outputs of the production network to an extent that depends on such factors as the phonemic confusability of the relevant material (see simulation results below). It should be noted that the above series of computations is difficult to carry out in a natural way within the network outlined above, because each unit must perform calculations such as dividing its own activation by μ.

When the material is to be rehearsed subvocally, the reconstructed sequence of phonemes is simply input to the perception net, which effectively re-recognizes the item in question and hence can re-activate, or refresh, the phonological storage nodes over which the item is represented. This process is essentially the same whether rehearsal or recall is to take place. Either process takes an amount of time that depends on the spoken duration of the items in question, because a complete pass through the production and perception system is required for each time-slice of the item to be rehearsed.

Simulation results

Interesting effects of rate of item presentation can emerge when items are presented to the model sufficiently slowly for rehearsal of the items to be possible between the identification of individual items. In the simulations reported below, however, the model is presented with the items at "rehearsal speed", and its ability to recall the items, along with the errors made, is subsequently examined. For the sake of simplicity it is assumed that all possible phonemes take the same length of time to produce, so that a word containing eight phonemes will take twice as long to articulate as a word with only four phonemes. These simplifying assumptions are not critical to the operation of the model.

We have run simulations to examine the capacity of the model to store sequences of items (memory span), and assessed the effects of (i) phonemic confusability and (ii) developmental increases in rate of articulation. In all the simulations reported below, the value of μ is set at 0.7.

(i) Basic capacity limitations

In the assessment of the model's memory capacity, a sequence of phonemes is reconstructed from the state units as described above, and this sequence is input into the sequence recognition network. The output of the perception net is taken to reflect those items that are recallable. It is frequently observed that the model only recalls, e.g., six or seven out of a word's eight phonemes correctly, but that this may still be sufficient to permit re-identification of the item in question. The model shows pronounced word length effects, in that fewer long words than short words may be recalled:

Mean no. of all long words recalled:	**3.2**
Mean no. of all short words recalled:	**6.7**

(ii) Phonemic confusability effects:

The model was tested on span for a set of short confusable items, which had many phonemes in common, and a set of (equally short) non-confusable items with few phonemes in common. The results were as follows:

Mean no. of confusable words recalled:	**3.0**
Mean no. of non-confusable words recalled:	**7.0**

At the point back in the sequence where errors begin to appear, they are typically exchange errors. For example, a typical trial produced the following output (each number represents a different phoneme:

```
Target:    . . . 8 14  9  3 11  5  1  8  4  2  9 10
Output:    . . . 14  9  8  3  5 11  8  1  4  2  9 10
```

Errors of this type in the recalled string of phonemes would typically lead to words being produced that were different from, but phonemically similar to, the target. However, the vocabulary of the model was too small to permit the systematic investigation of this phenomenon.

(iii) Developmental increases in capacity

A number of authors have recently suggested that increasing rate of subvocal articulatory rehearsal is the main factor responsible for the observed developmental increases in the short-term memory capacity of children, and there is now considerable evidence to support this view (see Baddeley, 1986, for a review).

We have conducted simulations in which we mimic slow articulation by inputting "null phonemes" to the state units in between every real phoneme. Thus the state units receive new input only every two time cycles, instead of on every cycle. Although it is difficult to generalize on the basis of a small vocabulary, one result of potential interest is that phonemic similarity effects are reduced when rate of articulation is low (Table One); this appears to mirror findings in the developmental literature.

	Confusable	Non-confusable
Fast rehearsal	3.0	6.5
Slow rehearsal	1.0	2.75

Table One

Summary

This paper has explored the properties of the temporary memory storage capacity that emerges as a by-product of recurrent nets used to model the production and perception of speech and other sequential data. It was shown how such nets may be interpreted as a model of auditory-verbal short-term memory. The model displays a number of characteristics observed in the psychological literature, and in particular, shows capacity limitations that depend on the phonemic confusability and spoken duration of the material to be remembered.

References

BADDELEY, A.D. (1986) *Working memory.* Oxford: OUP.

BADDELEY, A.D. & Hitch, G.J. (1974) Working memory. In G. Bower (Ed.), *Advances in the psychology of learning and motivation 8,* New York: Academic Press.

ELMAN, J.L. (1988). *Finding structure in time.* CRL Technical Report 8801, University of California, San Diego.

GROSSBERG, S. (1986). The adaptive self-organization of serial order in behavior: Speech, language and motor control. In H. Nusbaum (Ed.), *Pattern recognition by humans and machines.* Academic Press.

GROSSBERG, S. (1988). (Ed.). *Neural networks and artificial intelligence.* Cambridge, Mass: MIT Press.

JORDAN, M.I. (1986). Attractor dynamics and parallelism in a connectionist sequential machine. *Proceedings of the Eighth Annual Conference of the Cognitive Science Society,* Hillsdale, NJ: Lawrence Erlbaum Associates.

NOLFI, S., & PARISI, D. (1987). One trial learning of stimulus sequences in a connectionist network. Abstract from INNS conference.

NORRIS, D. (1989). Dynamic net model of human speech recognition. In G.T. Altmann(Ed.), *Cognitive Models of Speech Processing: Psycholinguistic and Computational Perspectives.* Cambridge, Mass: MIT Press (in press).

SALAME, P., & BADDELEY, A.D. (1982). Disruption of short-term memory by unattended speech: Implications for the structure of working memory. *Journal of Verbal Learning and Verbal Behavior,* 21, 150-164.

SCHNEIDER, W., & DETWEILER, M. (1987). A connectionist/control architecture for working memory. In G.H. Bower (Ed.) *The psychology of learning and motivation vol 21.* New York: Academic Press.

SCHRETER, Z., & PFEIFER, R. (1989). Short-term memory/long-term memory interactions in connectionist simulations of psychological experiments on list learning. In L. Personnaz & G. Dreyfus (Eds.), *Neural Networks: From models to applications.* Paris: I.D.S.E.T.

TOURETZKY, D.S., & HINTON, G.E. (1989). A distributed connectionist production system. *Cognitive Science,* 12, 423-466.

TOURETZKY, D.S., & HINTON, G.E., & SEJNOWSKI, T. (1989). (Eds.) *Proceedings of the 1988 Connectionist Models Summer School.* San Mateo: Morgan Kaufmann.

Implications from structural evolution: semantic adaptation

Peter Cariani
37 Paul Gore St, Boston, MA 02130 tel 617-524-0781

Syntactic and semantic adaptation: summary

An adaptive classifier consists of a set of observable distinctions (primitive features), their associated means of measuring the world (sensors), a set of mathematical relations on those observables (a decision function), and a means of changing the mathematical relations so as to improve performance (training rules). In the development of neural net research, much attention has been explicitly paid to problems associated with the form of the decision function and the training process, but relatively little effort has been directed towards the problem of finding the appropriate feature primitives within which the decision function operates. This is the problem of *creating new primitive features* .

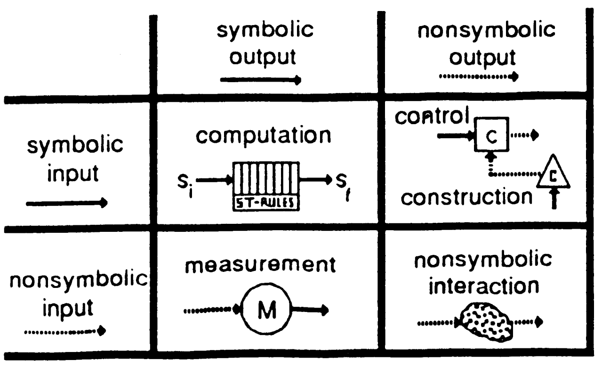
If we see the current adaptive devices as adapting the structure of their decision functions to better meet the demands of the task at hand, then the resulting process can be seen as *syntactic adaptation.*. The decision process is seen as completely syntactic, due to the rule-governed, completely symbolic nature of the decision function and its associated training rules.

Another type of adaptation is possible, however. There can be adaptation of the primitive features themselves, which necessarily involves changing the physical structure of the sensors which implement the primitive features. Because the sensors determine the relation of the symbols in the adaptive device to the world at large, they implement a semantic function. The process of making the structure of the sensors contingent upon their performance relative to a given task is one of *semantic adaptation.*

These ideas have come out of theoretical biology, from considerations involving the role of symbols in biological systems and the evolution of new structures and functions (Pattee, 1982, 1985; Cariani, 1989). We have in syntactically-adaptive neural nets rough analogues of ontogenetic learning processes. We have evolution-inspired mechanisms of "neural Darwinism" and genetic algorithms, but these all involve re-combination of pre-existing structures and functions. We have yet to develop any analogues to the evolution of qualitatively new structures and functions over phylogenetic time periods. In our tendency to reduce all cognitive functions to computations we often forget that central nervous systems co-evolved with sensory organs and effector organs, that whatever computed coordinative functions are effected must take place within whatever percepts and actions are available. We must therefore examine the processes of sensory evolution and structural evolution of effector organs. A device constructed according to these principles would be semantically-adaptive, one which constructed its own sensors and effectors, thereby determining its own feature and action primitives. As it turns out there is but one such device in the literature, the now-forgotten electrochemical device of Pask (1958, 1959).

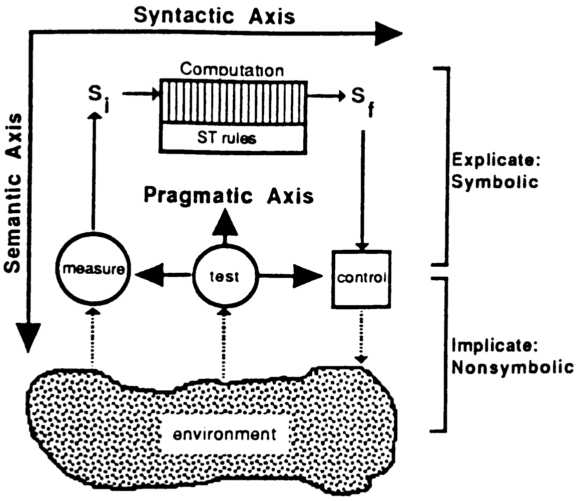
The semiotics of symbol-utilizing adaptive devices

The simplest way of conveying these distinctions is through a simple taxonomy of adaptivity. Three basic functionalities will be employed by the devices: computation, measurement, and control.Following the terminology of Charles Morris, *syntactics* will be defined as the relation of symbols to other symbols, *semantics* will be defined as the relation of symbols to the world at large, and *pragmatics* will be defined as the usefulness of symbols relative to the purposes and goals of their users. Computations are syntactic operations since they implement rule-governed relations between symbols. Measurements and controls are semantic operations, since they provide a structural linkage with the nonsymbolic world outside the device. The pragmatic aspects of a device involve the apparatus which steers the device toward optimal performance. The three

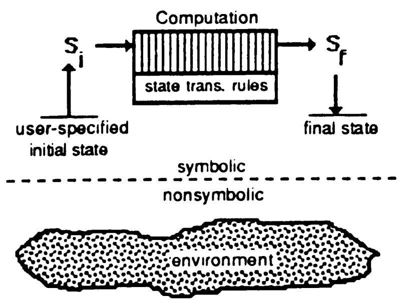


semiotic axes are all irreducibly orthogonal to each other, i.e., one cannot get semantic relations from purely syntactic, computational operations (see Cariani, 1989 for extended discussion).

The basic structure of our devices, the causal dependencies between the functionalities, will determine what parts are plastic and which parts are static. *Adaptivity* is the alteration of internal structure so as to better perform in an external environment. We can structure devices so that they improve their performance with experience by making the plastic functionalities contingent upon performance. The performance-dependent functionalities can be either in the syntactic or the semantic realm or both.

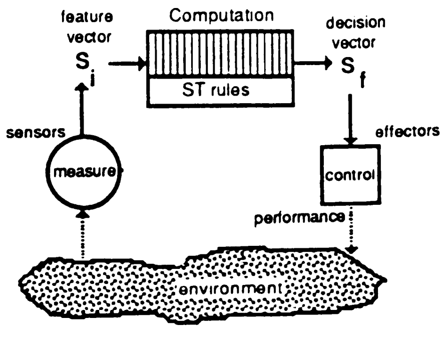


Three different types of devices can be distinguished with respect to their adaptivity: nonadaptive devices, syntactically-adaptive devices, and semantically-adaptive devices. These device types are a consequence of which structures are plastic and which are fixed. *Formal-computational devices* have only fixed syntactic parts, and no inherent semantics. *Formal-robotic* devices have both fixed syntaxes and semantics. Because of their completely static structures, both types are non-adaptive. *Adaptive devices* are syntactically-adaptive. *Evolutionary devices* are semantically-adaptive, but syntactically-nonadaptive. *General evolutionary devices* are both syntactically-adaptive and semantically-adaptive. When the organism or device has the capacity to construct its own syntactic, semantic, and pragmatic relations, as in a general evolutionary device, we have a situation of *semantic closure* (Pattee, 1982, 1985) in which the device or organism attains a degree of epistemic autonomy.



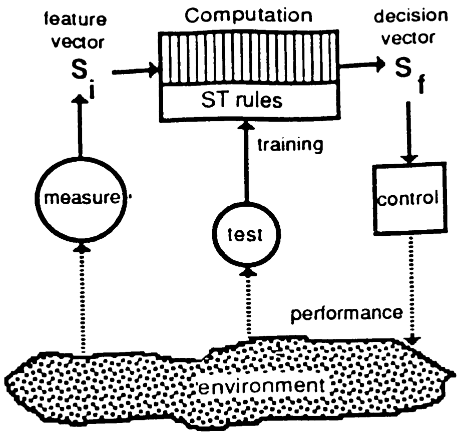
Formal-computational devices. Formal-computational devices are functionally equivalent to digital computers without sensors or effectors (i.e. completely syntactic, with no inherent external semantics). Being designed to operate as reliably as possible, independent of the external world, they are structurally nonadaptive. Because of their completely symbolic nature, formal-computational devices can only deal with problems which are already completely encoded into symbols. Because of their completely nonadaptive nature, everything that they do must in one way or another be completely specified. No new syntactic or semantic relations are created by these devices: the best that they can do is to combine pre-existing syntactic primitives into new combinations, as in the "mathematics discovering" program of Lenat.

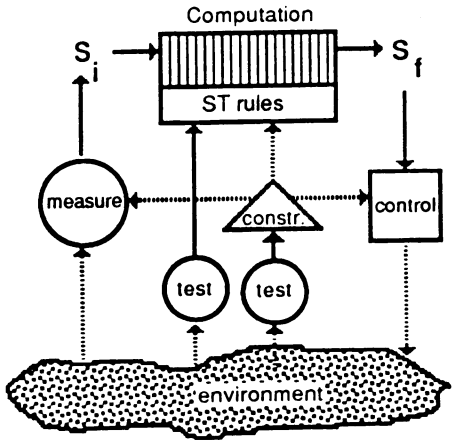
Formal-robotic devices. Formal-robotic devices are similar to formal-computational ones, except they have fixed sensors and/or effectors which connect them directly to the world, thereby giving them inherent external semantics. Robotic devices thus do not have to depend upon human beings to provide the interpretations for their symbols. Having both fixed syntaxes and semantics, formal-robotic devices can solve problems which are not already in symbolic form, (e.g. welding, walking, recognizing sights and sounds). Still, they are nonadaptive because they do not possess any structural plasticity and hence



cannot alter their internal structure to improve their performance.

Adaptive devices. Adaptive devices are a different story. Adaptive devices alter their syntactic, computational parts based upon their past experience and performance. Examples would be neural nets, trainable classifiers, Boltzmann machines, genetic algorithms, and connectionist associators. The advantages of adaptive devices lie in their flexibility and ability to adapt to unforeseen situations. To the extent that a given functionality can adapt itself to the exigencies of a specific situation, the designer is freed from having to directly foresee and specify the appropriate behavior. Here this adaptation is ultimately limited by the primitive features and actions available to the device. A neural net can only be as good as its percept and action primitives. The behavior of the device is semantically bounded: it must take place within the confines of what features and actions its sensors and effectors can implement.



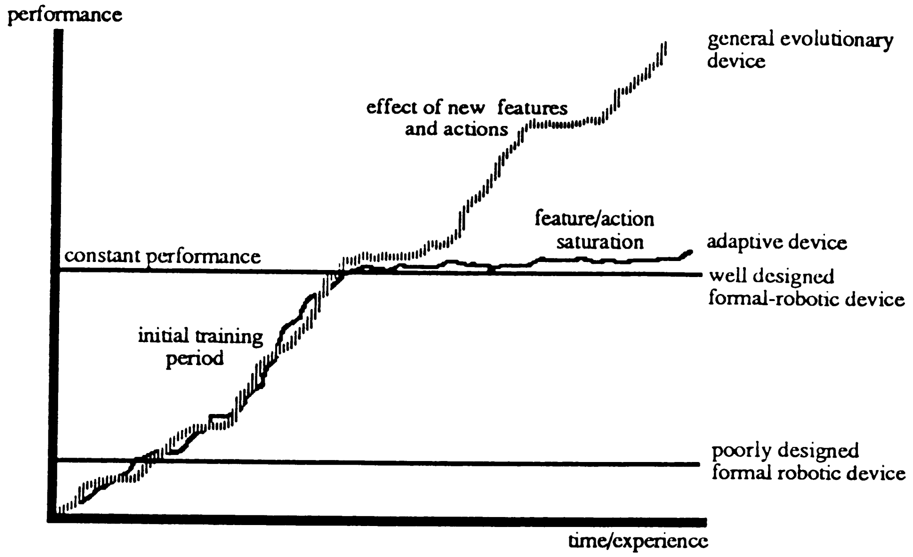


Evolutionary devices. Semantically-adaptive devices can enlarge the confines of the semantic realm by adaptively constructing new sensors and effectors. New sensors and effectors mean new primitive features and actions. The immune system is an example of such a device: sets of antibodies are adaptively produced contingent upon their ability to recognize antigens. General evolutionary devices have the capability of constructing all of their parts: they are both semantically-adaptive and syntactically-adaptive. Some examples would be the evolution of new sense organs and effectors in biological evolution and the construction of tools and sensing instruments by human beings. Like evolving species, we radically enlarge our perceptual and behavioral repertoires by constructing new means of interacting with the world.

Capacities and limitations

The relative capacities and limitations of the various device types are schematically presented in the figures below.

device type	plasticity	capacities	limitations
formal-computational	fixed syntax	reliable execution of pre-specified rules	limited to pre-specified rules and states
formal-robotic	fixed syntax fixed semantics	reliable execution of fixed percept-action combinations	no feedback or learning from environment
adaptive	adaptive syntax fixed semantics	performance-dependent optimization of percept-action coordination	limited to percept & action categories fixed by the sensors & effectors
general evolutionary	adaptive syntax adaptive semantics	creation of new percept & action categories; performance-dependent optimization within these categories	time to construct & test new sensors & effectors may be very long



Designing a semantically-adaptive device

Apparently there is but one device in the literature with the explicit goal of adaptively creating observables *de novo* (Pask, 1958, 1959). In the late 1950's Pask constructed an electrochemical device apparently with the intention of physically implementing an analog learning network, which became sensitive to other kinds of perturbations, such that it could be tuned with the appropriate rewards. Were we to build one today, how would we go about it? One of the requisite properties of an evolutionary device is that it contain a *construction* system for physically building the sensors and effectors. In biological organisms, the transcription>>translation>>protein-folding process serves as the construction system. A genetic algorithm could adaptively direct the *physical* construction of the new sensors and effectors implementing the feature and action primitives of a neural net (or other syntactically-adaptive device). We could thereby achieve adaptation on both syntactic and semantic levels.

Conclusions

It is tacitly understood by practitioners of neural nets and other adaptive devices that a complex feature space which is difficult to effectively partition may be made much more tractable by judicious choice of alternative feature primitives. We need to keep this in mind: that biological evolution generally proceeds by investing heavily in task-appropriate sensors and effectors and only minimally in computation-intensive coordinations. As a complement to the syntactic adaptation of the neural nets approach, we might well want to experiment with the construction of rudimentary semantically-adaptive devices.

References

Cariani, Peter (1989) On **the Design of Devices with Emergent Semantic Functions.** PhD dissertation, Dep artment of Systems Science, State University of New York at Binghamton.

Pask, Gordon (1958) Physical analogues to the growth of a concept. In: *Mechanization of Thought Processes: Proceedings of a Symposium , National Physical Laboratories, November 1958,* HMSO, London.

(1959) The natural history of networks. In: **Self-Organizing Systems.** MC Yovits & S Cameron, eds. Pergamon Press, New York, 1960 (Proceedings of a Conference on Self-Organizing Systems, Chicago, May 5-6, 1959).

Pattee, Howard H. (1982) Cell psychology: an evolutionary view of the symbol-matter problem. *Cognition and Brain Theory* **5**: 325-341.

(1985) Universal principles of measurement and language functions in evolving systems. In: **Complexity, Language, and Life.** J Casti & A Karlqvist, eds. Springer-Verlag, Berlin.

MODULARITY OF NEURAL NETWORK ARCHITECTURE

William P. Coleman
MIEMSS, UMAB; Baltimore, MD 21201
Department of Mathematics & Statistics, UMBC; Baltimore, MD 21228

David P. Sanford
Aeronautical Radio, Inc.; 2551 Riva Road, Annapolis, MD 21401

Andrea De Gaetano
C.N.R. Centro di Studio per la Fisiopatologia dello Shock,
Istituto di Clinica Chirurgica, Università Cattolica del Sacro Cuore;
Via Pineta Sacchetti, 644 I-00168 Roma, Italia

Fred H. Geisler
Division of Neurosurgery, UMAB; Baltimore, MD 21201, *and*
Department of Neurosurgery, Patuxent Medical Group; Columbia, MD 21045

Abstract

We explore the premise that the brain should be considered as a modular system of neural networks: that it can be viewed at different levels of granularity, and at each of these levels it forms an interconnected system of subnetworks whose components can be taken to be black boxes, computing outputs from inputs. At higher resolution, each such component itself forms such a system. We use categorial logic to see what structures are possible and how they can be related to cognitive function.

Introduction

Section II of the DARPA Neural Network Study concludes [1, 16] with a number of recommendations, one of which is that the study of advanced architectures with multiple modules should be encouraged. This paper is an exploration of the mathematical consequences of this single assumption.

Thus we regard a neural network as simply a black box that can compute a function, or can act as a content addressable memory, when called upon to do so. How it does so, and even whether it always can do so in complete generality, is irrelevant at this point. We assume that the net is connected to itself or to other nets in a system of nets in such a way that the output of one net may be the input to another, so that the corresponding functions can be composed. A sequence of inputs to the system of nets produces a sequence of outputs at the outputs of the individual nets.

Computational logic [2] is a very general theory that attempts to analyse the logic of computational and natural processes. The point of view of this paper, namely that we are studying a system of function computations and their compositions is a prototypical example. There is an extended review, with many examples, in [3].

Categories

Computational logic is based on the mathematical theory of categories [4], discovered by EILENBERG and MAC LANE, which has been used, particularly in work deriving from LAWVERE, both explicitly [5] and implicitly [6], to provide a foundation for logic. (There is an accessible presentation in [7].) The automata theory books [8] [9] are also very relevant to the present work.

In this section we give a brief and informal review of some elementary notions. An obvious example of a category is **Set**, the category of sets. It contains *objects,* namely sets, and *arrows* or *morphisms,* namely ordinary functions between sets. An arrow goes from its *domain* to its *codomain*. For each object c, there is an *identity function* i_c, and for suitable pairs of arrows $f : c \rightarrow d$ and $g : b \rightarrow c$ there is the *composition* $f \circ g : b \rightarrow d$. These obey the obvious rules, $i_d \circ f = f = f \circ i_c$, and $f \circ (g \circ h)) = (f \circ g) \circ h$, for any $h : a \rightarrow b$.

More generally, a *category* is any system of objects and arrows that obeys these rules. The objects might be sets with some structure, and the arrows functions that preserve that structure. Or, the category might just be a system of dots and arrows.

A *functor* from a category $\mathbf{C}$ to a category $\mathbf{D}$ is a mapping F between them, carrying objects and arrows in $\mathbf{C}$ correspondingly to objects and arrows in $\mathbf{D}$. Again, structure must be preserved: $F(i_c) = i_{F(c)}$, for all $c \in \mathbf{C}$, and $F(f \circ g) = F(f) \circ F(g)$, for composable f and g.

A *natural transformation* is a way of converting one functor $F : \mathbf{C} \rightarrow \mathbf{D}$ into another one, $G : \mathbf{C} \rightarrow \mathbf{D}$ between the same categories. The way to do this is by supplying a set of arrows in $\mathbf{D}$ that drag the image of F across to the image of G. For each $c \in \mathbf{C}$ one gives an arrow $\alpha_c : F(c) \rightarrow G(c)$ in $\mathbf{D}$. This is to be done in such a way that applying these functions to the image $F(d)$ and $F(c)$ of the domain and the range of any arrow $f : c \rightarrow d$ of $\mathbf{C}$ converts $F(f)$ to $G(f)$. Thus, $G(f) \circ \alpha_c = \alpha_d \circ F(f)$: One writes this as $\alpha : F \overset{\cdot}{\longrightarrow} G$.

Schemata

First, we try to capture the notion of a system of processes. A net in a system of nets has inputs and outputs to and from other nets. Each of these inputs and outputs has a *state.* A *type* is the set of states that an input or output can possibly assume. Let $\mathbf{C}$ be a category freely generated from a diagram indicating a finite number of types and a finite number of arrows – "basis" arrows – between them. $\mathbf{C}$ shows schematically the connectivity of the system. Let $C : \mathbf{C} \rightarrow \mathbf{Set}$ be the functor that sends each type to the set of states that it comprises, and each arrow to a function between the corresponding sets. Thus C lets the logic of $\mathbf{C}$ act on **Set**. We call C a *schema,* in a sense [10] perhaps closer to that of [11] and [12] than that of psychology or AI.

Next, we have to distinguish between a schema, which is a template according to which certain processes can occur, and an instance in which such a process actually occurs in the system of networks. A *development* of $\mathbf{C}$ is [2] [3] a functor from a finite totally ordered category $C_1 : \mathbf{C}_1 \rightarrow \mathbf{C}$ that maps basis arrows of $\mathbf{C}_1$ to basis arrows of $\mathbf{C}$. The development induces a schema $C \circ C_1 : \mathbf{C}_1 \rightarrow \mathbf{Set}$. It is evident that a development of C corresponds to a path of C. As any instance, or *particle,* of the process unfolds, it corresponds to successively longer developments.

We hypothesize that part of the semantic power of cognitive processes comes from the fact that a development can function simultaneously in several schemata. To do so, we need a notion of how one system of processes can be modeled by a simplified version. A *model* $M = \langle F, \alpha \rangle$ of $C' : \mathbf{C}' \rightarrow \mathbf{Set}$ in $C : \mathbf{C} \rightarrow \mathbf{Set}$ is given by [2] [3] a functor $F : \mathbf{C}' \rightarrow \mathbf{C}$, and a natural transformation $\alpha : (C \circ F) \overset{\cdot}{\longrightarrow} C'$,

$$\begin{tikzcd}
\mathbf{C} \arrow[r, "C"] & \mathbf{Set} \arrow[d, "\alpha"] \\
\mathbf{C'} \arrow[u, "F"] \arrow[r, "C'"'] & \mathbf{Set}
\end{tikzcd}$$

Intuitively, we think of C' as a simplified version of C in which the states have been relabeled by α. To discover the effect of some process $f : c \rightarrow d$ in C' on a state c of a type $C'(c)$, find a state that α_c sends to c, do the corresponding process $F(f)$ on it, and then use α_d to relabel again.

A ***conceptual schema*** is [2] [3] a schema $C : \mathbf{C} \rightarrow \mathbf{Set}$ together with a system $\{C_i : \mathbf{C}_i \rightarrow \mathbf{C}\}$ of schemata with models between them and from them to C. A ***development*** of a process is a development of one of the schemata $C_i C : \mathbf{C}_i \rightarrow \mathbf{Set}$.

Semantics

Some cognitive processes describe natural processes outside the mind. Some cognitive processes describe other cognitive processes. We use the notion of 'model' to capture how one process can describe another.

Let $C : \mathbf{C} \rightarrow \mathbf{Set}$ and $C' : \mathbf{C'} \rightarrow \mathbf{Set}$ be the conceptual schemata of two processes. An ***interpretation of*** C' ***in*** C is [2] [3] a model $M = \langle F, \alpha \rangle$, together with some extra data. For each schema $\{C'_i : \mathbf{C'}_i \rightarrow \mathbf{C}\}$ of C there must be given a schema $\{C_i : \mathbf{C}_i \rightarrow \mathbf{Set}\}$ and a functor $F_i : \mathbf{C'}_i \rightarrow \mathbf{C}_i$, and there must be a function assigning to each particle of $\mathbf{C'}_i$ a particle of $\mathbf{C}_i$.

These mathematics allow us to speculatively hypothesize a theory of cognitive function. At the lowest level, the brain has sensory process. At higher levels, it has other processes that are modeled in sensory ones. There are also processes that capture the modeling and interpretation relations and can decide whether one process is correctly instantiated in another. Sensory processing is a meeting in real time of bottom-up information, from the senses, with top-down information, from a conceptual schema, in which we agree (in a sense) with Grossberg [13]. In case of mismatch, a different schema is tried. If no available schema matches, the information is ignored, or a schema is built or rebuilt to fit. The successive replacement of schemata, as well as the integration of schemata into successively more powerful models, are basic features of learning.

The fact that there is are relatively small number of schemata available at one time is central to the semantics. Not every brain state is possible without rebuilding the schemata, but the number of them is large and depends on the possible combinations of schemata. The states that are possible are semantically linked by the relationships of the schema processes that can generate them — the brain doesn't find them by searching, but by using a pattern to generate them.

Network behavior can be viewed at any level, and the picture obtained will be correspondingly simplified, idealized, abstracted from the sensory, but the modeling relation ensures that it reflects the lower levels in a certain precise sense. It is always possible to look to the lower levels to recover detail.

References

[1] *DARPA Neural Network Study*. AFCEA International Press, 4400 Fair Lakes Court, Fairfax VA 22033 USA, 1988.

[2] William P. Coleman. Models of computational processes. (To be submitted to Journal of Symbolic Logic).

[3] William P. Coleman. Computational logic of network processes. In Martin D. Fraser, editor, *Advances in Control Networks and Large Scale Parallel Distributed Processing Models*, Ablex Publishing Company, 1989.

[4] Saunders Mac Lane. *Categories for the Working Mathematician.* Springer-Verlag, New York, 1971.

[5] J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic.* Cambridge University Press, Cambridge, 1986.

[6] M. Barr and C. Wells. *Toposes, Triples and Theories.* Springer-Verlag, New York, 1985.

[7] Robert Goldblatt. *Topoi: The Categorial Analysis of Logic.* North-Holland Publishing Company, Amsterdam, revised edition, 1984.

[8] Samuel Eilenberg. *Automata, Languages, and Machines.* Volume A, Academic Press, New York and London, 1974.

[9] Samuel Eilenberg. *Automata, Languages, and Machines.* Volume B, Academic Press, New York and London, 1976.

[10] William P. Coleman, David P. Sanford, and Andrea De Gaetano. Logic of computational processes. (Submitted to Theoretical Computer Science).

[11] *Information Processing Systems — Concepts and terminology for the conceptual schema and the information base,* ISO/TR 9007: 1987. International Organization for Standardization.

[12] *Working Document for the Directory Schema* N 3318. International Organization for Standardization, Secretariat ISO/IEC JTC1/SC21 - American National Standards Institute; 1430 Broadway, New York, NY 10018, December 1988.

[13] Stephen Grossberg, editor. *Neural Networks and Natural Intelligence.* MIT Press, Cambridge, MA, 1988.

Symbolic Networks with Timers, Latches and Classifiers May be Mapped to the Nervous System

Armand de Callataÿ
IBM A.K. Watson IEC, La Hulpe, Belgium (callatay at bbribm11 on EARN)

Summary. *A study of cybernetics and artificial intelligence mechanisms to control animal behavior has led to select integrable devices for a detailed hybrid brain model (Callataÿ 1986). I have suggested that intelligent behavior is best controlled by rhythmic, symbolic, all-or-none processors with write-once memory (Callataÿ 1969). A simulation has shown that a general purpose controller for an android arm can learn to reach a target found by a pointing device. The training time is very fast (Callataÿ 1986). The kernel of this model is implemented in a radically new type of computer without ALU (Callataÿ 1988). The research method and its relevance for studying brains is discussed in Callataÿ (1989). The first part of this paper briefly summarizes these researches. The second part evaluates how the model may be mapped to the central nervous system according to recent biological findings.*

Introduction. A symbol, as opposed to an analog value, is a value chosen among discrete possibilities. Categorization is a non-linear computation finding a winner in a competition. Categorization selects a symbol. Categorization is described by a dynamic system bifurcating to one of its potential attractors, depending on the external conditions. Instantiating one cell per group is a simple implementation of symbolic computation. Instantiations are changed in steps after a time interval. Thus symbolic processors have phases or rhythms (Callataÿ 1969).

My research method, derived from engineering methods, is to design, simulate and study fully autonomous learning robots having a symbolic central component. The resulting partly-conceptual device is a conjectural brain model "Hybrid."

I briefly review the main features of "Hybrid," described with their neuroanatomical mapping. Then I argue that the neural mapping of the all-or-none processors may be realistic.

Hybrid symbolic/analog computations. A representation in a neural layer occurs when the cell activations normally produced by a sensory situation are generated in another way. This representation is an intention when the system can find the robot's sequence of muscular actions generating this situation in the physical world. Conscious content is intentional and seems to be representable by symbols. The connectionist rule of the radical behaviorists: *stimulus → response*, has been enhanced in the cognivitists' formula: *stimulus & internal state → intention → response*. We must represent intentions and will them before acting. I suggest that most vertebrates also use this formula.

The first phase selecting an intention is adaptive and can be implemented with a simple perceptron because stimuli are discriminated elsewhere. The second phase is a resolution analogous to problem solving in logic programming. After each action, the controller adds in its memory an invariant transition rule: *the body must do the actions C when it is in posture A and has to go in posture B after a time T.* These transition rules use static symbols to store dynamic events. Each rule is associated with a timer or a clock to take dynamics into account. The rules translate repetitive physical laws in naive physics.

Some thermostats adjust their parameters from experience to minimize the oscillations following a signal change. More complex systems describable by differential formulas can adjust their connection strengths to walk or keep equilibrium with a body having changing weights or muscular forces. Walking is stereotyped, but successful. Insects may be controlled only by such dynamic regulators. Decerebrate cats can even change their walking or running mode (Grillner 1985). The modification of a few parameters by a wired-in mechanism (electrical or neural) is called "adjustment" to stress the difference with generalized learning. The symbolic commands are set-points for the above adjustable servo-mechanisms (in medulla and colliculi).

The cerebrum symbolic command is followed by a cerebellar corrective action to skillfully stop the movement and keep equilibrium. To offset the neural computational delay, the cerebellar cortex predicts the subsequent situation, and provides signals with a negative delay to the cerebellar nuclei.

The cortical neuron groups are recording content addressable memories (RCAM). Rules are documents described by keywords (inputs). RCAMs use the weighted document retrieval algorithm. When there is no logically applicable rule, the most similar rule-document is chosen, even with missing keywords. This non-logical inference method is suggested as the spontaneous mechanism of intuition. It implements "reasoning by analogy," as described by Minsky (1986).

Learning at once is implemented by irreversible all-or-none switches (latches) instead of adaptive connections. Adaptive learning necessarily provokes the partial forgetting of some previous knowledge. Additive memorization keeps previous information.

What is true of an event is the whole rule associating inputs, internal states and outputs. The rule is stored in a directional lattice whose nodes are RCAMs. The event, represented by an activated subnetwork, is automatically decomposed into a structure of simple rules delimited by the partitions of the neuroanatomically defined lattice. The cascade of memorizing classifiers gives non-logical opportunities to group many similar simple rules. The unsolvable combinatorial explosion problem is thus replaced by a classification problem which may have acceptable solutions (Callataÿ 1986). Situations recognized and processed as similar can afterwards be memorized as different events. This method avoids the "duplication problem" (Minsky 1986).

I have suggested a brain genesis rule based on scheduled and induced cell expressions to build a circular directed network processor systems (DNPS) of RCAMs having the hardware property to be a logic problem solver (Callataÿ 1988). Recent findings on substrate and cell adhesion molecules (SAM and CAM: Edelman 1988) explain how each developing axon can grow on recognized glial cells, leading axons or target dendrites.

Using the single (or non destructive) assignment in pure Lisp or Prolog ensures that growing consistent programs remain consistent. The irreversible closure of open latches implements this fundamental feature of logic in symbolic neural networks. DNPS are logical resolvers (give results which are proven true) in exceptional cases: when all needed rules are known, are still valid after hierarchical decomposition and categorization, are based on predictable events in which the relevant input/output data have been recorded, and are such that a solution can be found in one network sweep. RCAMs always find an action before a fixed reaction time, although not necessarily an efficient one. Similarly DNPSs always solve the given problem within the rhythmic period (or find an analogous solution), whatever the data and the redundantly stored relations.

Integration with feature extractors and servo-mechanisms. The natural feature extractors (retina, auditory nuclei, dorsal horns, olfactory bulbs, colliculi) are much more complex than cells adding weighted impulses. The dynamic computation in these feature extractors produces an output subsequently classified into symbols. Only a tiny amount of relevantly correlated information is extracted from the sensory captors (compare what is seen in a briefly presented picture with the information captured by a photography having the discriminative power of the retina).

In Hybrid the dynamic feature extractors cannot learn in real-time, due to the following difficulties: Too many experimental trials are required to find which features are interesting to extract. The reward signal used by natural selection is survival, delayed for a lifetime. Interposing an adaptive connectionist layer between the feature extractors and the symbolic neural system jeopardizes the logical consistency of the stored rules and does not seem useful.

Hybrid knowledge representation. Distributed memory can mean the coding of a datum in many components, the redundant storage of the same datum, and different data coded in the same component. Hybrid uses non-coded redundant representation.

On the one hand, signal processing uses sensory captors giving continuous data; thus the feature extractors must work with this high density type of inputs. On the other hand, AI programs work better when very little knowledge is instantiated. We do not use more than 10 concepts at the same time, although we know 100,000 words and expressions. So, with one concept per neuron, only one neuron per 10,000 should be concurrently active. This is not compatible with the observation that each neuron fires on average many times per second. The hypothesis here is that only the bursts (high frequency sequence of impulses in an axon) are logical impulses in the central nervous system.

Predicted neural functions. The wiring of Hybrid is roughly mappable to the mammal neuroanatomy, but the predicted physiology is not conventional, The following lists the main physiological functions for which a mapping, even an unlikely one, is searched:

- a computation in steps of the weighted sum.
- synaptic weights which are negligible before learning, can be made permanently positive at once and can be slightly modified after learning
- a recognizing and a (different) memorizing classifier in each neural group
- a central rhythmic control
- a synchronization of the processing wave
- a centrally rewarded habituation
- enabling systems for attention, command and memorization
- a memory associating the state before and after an event (260 ms delay)
- neurogenesis rules building a specific network

Phasic computation of the weighted sum. The usual conceptual cell adds weighted inputs until a threshold occurs, resetting the cell. Thus the output frequency continuously measures the weighted sum. The output cells are activated by excitatory inputs also activating the local neurons inhibiting these output cells. The two effects are variably compensated because the relative timing of impulse arrival and the relative position of each synapse type on the dendrites and spines modify the cell firing (Shepherd 1988). In Hybrid, the computation is done in steps instead of continuously. The weighted sum is integrated for 30 ms, allowing decisions to be timed by clock signals.

Hybrid has two simple perceptrons for finding the subsequent intention (in globus pallidus and substantia nigra) and the corrective action (in cerebellar nuclei: Thomson 1986). These nuclei receive inhibitory GABA inputs. Because the addition of negative impulses does not provoke firing, the inhibited nuclei can count for a long time. The subsequent rebound burst may be proportional to the depth of inhibition (Eccles et al. 1967, Llinas 1989)

The dendrites have mainly Ca channels. The dendritic Ca ionic concentration increases with activity. The voltage dependent Ca channels have less efficient Ca openings than the Na channels, and the dendrites have few ionic pumps, so that the dendritic micro-impulses last up to 10 ms (instead of 1 ms), have a lower amplitude, and are usually stopped at the dendritic branches (Llinas 1979, 1989). The repetitive activation of dendrites creates "hot spots" (concentration of primed Ca channels) from which micro impulses are initiated. The transient physico-chemical changes in the dendrites have a lifetime longer than 10 ms and may store the integrated weighted sum. The neuron whose dendrites were recently activated is bursting instead of firing when it rebounds (Llinas 1979).

All-or-none learning. The impulse conduction increases in epileptic areas, or in electrically stimulated (kindled) areas. This suggests the presence of a natural mechanism of memory formation which becomes visible when it is spatially generalized in epilepsy ("Epilepsy as a memory model," Goddard 1967). In long term potentiation, learning occurs only after a few bursts (not after simple firing). The associated input is best learned when it occurs after a 200 ms interval (Larson and Lynch 1986). In the hypothetical Hybrid mapping (Callataÿ 1986, 1989b), memory is formed only after periodic bursts in the very few areas enabled.

In vertebrate brains, most long axons do not synapse on dendrites, but on spines. Spines make connections with a low weight. Those with larger stalks have more weight (Koch and Poggio 1983). The filaments maintaining the spine shape are attached together and to the membrane proteins by fodrin, a glue protein. Ca ions indirectly destroy the glue (Lynch 1986). The spines contain the filaments found in muscles. The spines may also contract with Ca ions. The result of these two effects is a permanent increase of the spine diameter (Lynch 1986).

An extreme view is that the spine is fully resorbed when many activations converge on it, so that the synapse is permanently moved to the dendritic wall (Callataÿ 1986). Synapses on dendritic wall are much more efficient (Koch and Poggio 1983). Bursts increase Ca uptake (Gamble and Koch 1987). If the Ca buffers are full, the Ca excess can kill the cell or a part of it. The reorganization can eject or kill the subsynaptic apparatus located in each spine stalk. This apparatus is believed necessary for spine maintenance. Old rats have more synapses on their dendritic wall than young ones (Connor and Diamond 1982). In Hybrid, the hot spots are resorbed spines activated by bursts. A computation has shown that up to 6% of the spines can be resorbed before interferences disturb the logic deductions (Callatay 1986).

Neural timers. The "low-threshold voltage-dependent Ca channels" are activated by hyperpolarization and remain activated a little later. These channels explain that cells can behave like controllable oscillators and resonators (Llinas 1989). Synchronized brain clocks for fast rhythms (3 to 14 Hz) are observed in the inner brain. The inferior olive controls the cerebellum rhythm. The thalamus controls the cerebral rhythm. Depending on the neuromodulators, the thalamus produce fast rhythms (10 Hz) favoring tonic firing or slow rhythms (6 Hz) favoring bursts (Llinas 1989).

The electro-encephalogram (EEG) shows that brains have rhythms when idle. On the one hand, EEG desynchronization occurs for conscious activities. Therefore the models computing in a continuous or chaotic way were favored by neuroscientists. On the other hand, the skilled movements are done according to specific rhythms synchronizing each phase relation of the limbs. Sophisticated controls by clocks would explain our easy recognition of musical rhythms. The average evoked potentials (AEP) may be a clue to hidden rhythmic operations. AEP are neural waves of a few μvolts isolated in noisy EEG of 50 μvolts. AEP shows a sequence of waves repeated for each trial, whatever the content of the task. AEP detects that very few columns are computing. AEP is spontaneously generated by the cortical column operations in Hybrid (figure 144 in Callataÿ 1986). The AEP short waves after 300 ms (if they exist) cannot be detected due to rhythm irregularities and noise.

During sleep, action and memorization are turned off. During wakefulness, this should also be true in most brain areas in Hybrid. A neural area cannot compute and memorize at the same time. The memorization phase may be locally similar to a sleep transition phase. A specific enabling is needed for generating attention and real action, and this action must be automatically followed by memorization (during AEP P300), except if nothing is recognized as new, like in most of our automated movements. The computation with bursts only occurs in 0.01 % of the brain groups in Hybrid, and the experimental EEG data do not preclude rhythms in few enabled areas.

Enabling of cerebral columns. Attention is the first enabling level; authorization to act is the second; memory assignment is the third. To restrict the brain areas assigned to learning, the Hebb conjunctive learning should be further limited. To choose only one path to store each relation in a network, a central system must select a restricted well-connected network for learning. The columns have a bidirectional axonal network to use back-propagation algorithms for cell reservation (Callataÿ 1969, 1986). The cerebellar learning is enabled by direct climbing fibers because what should have been predicted is known at the period end.

The neuromodulators are distributed by long axons covering long narrow brain territories. The number of neurons producing these neuromodulators (10,000 to 100,000) is too low to represent specific data, but may represent the categories on which attention can be focussed. The neuromodulators may act like hormones in delimited brain volumes. They are not secreted where the receptors are located (Herkenham 1987). The lifetime of the internal channel ligands is much longer than the neurotransmitters, allowing a persistent enabling (say a second instead of a ms).

Brains must have control circuits comparable to the computer ones, much more complex than those here sketched. "Hybrid" controls are tentatively mapped as follows (modified from Callataÿ 1986).

- *Serotonin* controls attention. It produces second messengers, whose actions make the NMDA channels significantly more effective when they are subsequently activated by excitatory glutamate bursts (most modulators also act by cascade actions from second messengers).
- *Acetylcholine* starts the processing and memorization mode. It may activate the Ca dependent K pumps to switch from the tonic mode to the burst mode. A cell should be enabled by several modulators to reach a bursting threshold sufficient for acting or memorizing.
- *Norepinephrine* prevents habituation. It is emitted when the reward or threat is high. It may transiently control the "persistent Na channels" controlling the cell resting level (Llinas 1989).
- *Dopamine* holds intentions. It may transform the recently bursting neurons in oscillators.

Classification. To find the most activated cell of a group, one must compare the cell activities on an equal footing, for example by giving a reset signal with a basket cell inhibition, and subsequently by probing the increasing activity of all cells. Then the most active cell should inhibit the others. The best defined categorization in biology is done in Mauthner cells to decide whether a fish will escape to its left or right. The axon cap around the initial segment of Mauthner cell controls the firing threshold, and is similar to those found in cerebellum, but there each local axon surrounds hundreds of cells. The non-firing retinal horizontal cells enhance contrast by transmembrane release of inhibitory transmitters to the non activated areas. The different states of each part of this cell permit differentiated local actions. The chandelier axon synapses with the axon initial segment of the large cerebral neurons. The parallel contact lines look like candles. This cell may be used for categorization as it can probe the voltage of all cells of a group. The chandelier cell was discovered in 1975 and fully described in 1977 (almost all other cortical neurons were known in 1911, and described in Cajal's book). The chandelier cell was expected ("echo cell" in Callataÿ 1969), as it is strategically located to categorize. A differentiated inhibitory action, less efficient for the cell with the fastest bursts, would provide a categorization.

Conclusion. It is hoped that brain analogies will be suggested by the design of controllers experimentally testable. Neuroscientists, examining the functions of models, may find which neuronal mappings are realist, or set-up experiments ruling out model predictions.

References

Cajal S. Ramon y (1909-1911), *Histologie du Système Nerveux de l'Homme et des Vertébrés,* Maloine, Paris.

Connor J.R. and Diamond M.C. (1982) A comparison of dendritic spine number and type on pyramidal neurons of the visual cortex of old adult rats from social and isolated environments, *J. of Comp. Neurol,* 210: 99-106.

de Callataÿ A. (1969) Brain Model with Periodic Processing. *Curr. Mod. Biol.*, 2, pp. 307-319.

de Callataÿ A. (1986) *Natural and Artificial Intelligence: Processor Systems Compared to the Human Brain,* North Holland, Amsterdam.

de Callataÿ A. (1988) Logic programs directly processed in a network of content addressable memories, *Future Generations Computer Systems,* 4, 117-131.

de Callataÿ A. (1989) Can Artificial Intelligence help in finding how brains may work? in *Springer Series in Brain Dynamics, Vol. 2,* eds. E. Basar, T. Bulloch, Springer, Heidelberg (in press).

de Callataÿ A. (1989b) Biological aspects of neural networks, *Actes des Journées d'Electroniques 1989,* Presses Polytechniques Romandes (in press).

Eccles J.C., Ito M., Szentagothai J. (1967) *The Cerebellum as a Neuronal Machine,* Springer, Berlin.

Edelman G.M. (1988) *Neural Darwinism,* Addison Wesley.

Gamble E. and Koch C. (1987) The dynamics of free Calcium in dendritic spines in response to repetitive synaptic input, *Science,* Vol 236, pp. 1311-1315.

Goddard G.V. (1967) Development of epileptic seizures through brain stimulation at low intensity, *Nature,* Vol. 214, 1020-1021.

Grillner S. (1985) Neurobiological bases of rhythmic motor acts in vertebrates, *Science,* Vol. 228: pp. 143-149.

Herkenham M. (1987) Mismatches between neurotransmitter and receptor localizations in brain: observations and implications, *Neuroscience,* vol. 23, No. 1, pp. 1-38.

Koch C. and Poggio T. (1983) Electric properties of dendritic spines, *Trends in Neurosciences,* pp. 80-83 (March)

Larson J. and Lynch G. (1986) Induction of synaptic potentiation in hippocampus by patterned stimulation involves two events, *Science,* 232, 985-988

Llinas R. (1979) The role of Calcium in neuronal function, in *The Neurosciences: Fourth Study Program,* F.O. Schmitt and F.G. Worden.eds. M.I.T., Cambridge, Massachusetts, pp. 555-572.

Llinas R. (1989) The intrinsic electrophysiological properties of mammalian neurons, Insights into central nervous system function, *Science,* Vol. 242, pp. 1654-1664.

Lynch G. (1986) *Synapses, Circuits, and the Beginnings of Memory,* MIT Press, Cambridge, Massachusetts.

Minsky M. (1986) *The Society of Mind,* Simon and Schuster, New York

Shepherd G.M. (1988) *Neurobiology,* 2nd Edition, Oxford University Press, Oxford.

Thompson R.F. (1986) The neurobiology of learning and memory, *Science* 233: 941-947.

Modelling of Human Neocortical Surface and Its Growth

Vinod D. Deshmukh, M.D., Ph.D., Department of Neurology, University of Florida, Jacksonville Division, Jacksonville, Florida and V. Ramamurthi, Ph.D., Department of Mathematical Sciences, University of North Florida, Jacksonville, Florida.

Developmental transformations of human neocortex appear to be complex and almost chaotic. This is an attempt to make topographic measurements of its surface, to develop a mathematical model of its transformations and to understand the probable underlying mechanisms of its morphogenesis. As a part of this research project, topographic measurements were made on published, life-size photographs of seven fetal cerebral hemispheres, and twelve human fetuses.

Material:

Published, life-size photographs of seven fetal brains from a standard textbook of Human Anatomy [1] were obtained. This series of photographs shows the superolateral surfaces of human fetal cerebral hemispheres at ages 21, 24, 26, 28, 30, 34, and 40 weeks. These photographs demonstrate visually, the sequential changes in neocortical size, surface topography and the progressive emergence of characteristic gyral pattern. By 40 weeks, most of the characteristic features of adult human neocortex, in terms of surface topography, are evident.

For the topographic study of fetal head at different ages, a series of published photographs from the Photographic Anatomy of the Human Body [2] were obtained. This series provides a set of proportionally sized photographs of the lateral view of human fetuses at ages 35, 40, 50, 60, 90, 120, 150, 180, 210, 240, 270 and 300 days of fetal life. A series of measurements were made of the perimeter and area of fetal heads at different ages.

Methods:

The topographic measurements were made using the "SigmaScan" scientific measurement system (Jandel Scientific). The system was calibrated for a distance of 1 mm and an area of 1 sq. mm. The photograph of each neocortical or fetal specimen was fixed on a graphic digitizer tablet. Simultaneous digital measurements of neocortical or cephalic perimeter and area, were made. The digitizing cursor (puck) was used to trace the neocortical margins including the complex opercular border. The continuous trace formed a digitized polygon for data input and analysis. This digital data was then subjected to further statistical analysis and graphic presentation.

Results:

a) Fetal Neocortical Measurements:

The perimeter and area measurements of the seven fetal neocortical specimens are shown in Table I. Linear regressions (LG) were performed on the acquired digital data. Excellent correlation was found between the neocortical surface area and fetal age in weeks. The correlation coefficient was 0.98609 with standard error (SE) estimate of 226.25524. The LG equation for correlation was:

$$Y = (-2914.52712) + (191.43319)\,X$$

Table 1

Fetal Age Weeks	Perimeter mm	Area Sq. mm
21	188.40	1079.54
24	245.94	1571.75
26	250.89	1803.43
28	320.51	2677.64
30	346.76	3021.38
34	398.79	3784.35
40	432.19	4521.13

Legend: Topographic measurements of neocortical surface in terms of perimeter and area at fetal ages 21 - 40 weeks.

The correlation between neocortical perimeter and fetal age in weeks was also good. Its correlation coefficient was 0.96979 with SE estimate of 23.55714. The LG equation for their correlation was:

$Y = (-75.29811) + (13.35271) X$

The correlation between neocortical perimeter and area, as expected was outstanding. Correlation coefficinet was 0.99394 with SE estimate of 149.59960. The LG equation was:

$Y = (-1734.44013) + (14.01425) X$

b) Fetal Cephalic Measurements:

The data of fetal cephalic measurements in terms of perimeter lengths and area are documented in Table 2. Again excellent correlations were found between i) cephalic perimeter and fetal age, ii) cephalic area and fetal age and iii) cephalic area and perimeter. The correlation coefficients for these linear regressions were 0.98598, 0.97280 and 0.97550 respectively.

Table 2

Fetal Age Days	Perimeter mm	Area Sq. mm
35	3.50	0.66
40	4.16	1.13
50	5.55	2.09
60	11.07	7.79
90	23.79	36.41
120	37.22	100.25
150	70.60	330.52
180	78.55	418.78
210	76.64	433.34
240	96.48	568.66
270	97.97	608.39
300	122.02	927.03

Legend: Topographic measurements of fetal head in terms of perimeter and area of fetal ages 35-300 days.

Discussion:

As a result of the development of new technology and related software, it is now possible to perform accurate topographic measurements, in terms of distance, area, volume, angle etc. Once such data is collected in the digital form, it can be easily subjected to sophisticated mathematical analysis and transformations. This can be further processed to develop a realistic model with a three dimensional reconstruction. Such realistic modelling, based on actual structural data, can give us a better understanding of the complex process of morphogenesis and the immense variety of neurobiological structural transformations. The modelling of developmental stages of human neurocortex with its complex gyral pattern is one such challenging task.

At this early stage of this research project, it has been shown that accurate topographic measurements of complex biological forms can be made with added statistical and graphic analysis. The measurements and subsequent modelling are realistic as it is based on actual life-size photographs of fetal human brains and bodies at different stages of growth. Excellent statistical correlations based on linear regressions between different parameters have been shown. The neocortical or cephalic area correlates well with perimeter and fetal age. The area (unpublished data) and probably the volume of a biological form seems to correlate better with its age and the daily changes in the weight of the developing chick embryo (White Leghorn) from the reference [3]. The gradients of morphological transformations seem to follow a logistic or sigmoid growth curve, as all biological growth is self-limiting at some point in time.

Our future goal in this project is to verify our hypothesis that the complex gyral pattern of the human neocortex is due to a nonlinear dynamic or chaotic transformation of the basic process of neocortical fissuration which causes radial and concentric infoldings of the developing neocortical surface. It appears to be centered around the insula in the lateral cerebral fossa, and to be related to the dynamic processes of subcortical and neocortical vascularization and the intracranial fluid dynamics.

Summary:

Mathematical modelling and computer simulation of human cerebral neocortical surface, at different stages of fetal development, were performed using the "SigmaScan" measurement program. Simultaneous measurements of perimeter and area were made of the neocortical surfaces from published life-size photographs of 7 fetal brains and twelve human fetuses. Excellent correlation was found between the fetal age and the neocortical surface area. Future work is planned to model the complex and chaotic gyral pattern of the developing human neocortex. A hypothesis of such a biological transformation is briefly proposed.

References

1) Gray's Anatomy, 36th British Ed. Chap 2, edited by P.L. Williams and R. Warwick, Embryology, p. 172, published by W.B. Saunders Co. 1981.

2) Photographic Anatomy of the Human Body, p. 65, C. Yokochi, University Park Place, 1971.

3) A Guide to Vertebrate Development, 7th Ed., p. 148, Roberts Pugh, Burgers Publishing Company, 1977.

SIMULATION AND ANALYSIS OF A MODEL OF MITRAL/GRANULE CELL POPULATION INTERACTIONS IN THE MAMMALIAN OLFACTORY BULB

Jay A. Edelman and Walter J. Freeman
Department of Physiology, University of California, Berkeley, CA 94720

This report details the results of analyses and numerical simulations of various embodiments of a model of mitral/granule cell population interactions within the olfactory bulb. These cells form dendrito-dendritic connections in the external plexiform layer (EPL) of the bulb. The EPL receives excitatory input from olfactory receptors via the primary olfactory nerve (PON) and the olfactory glomeruler layer. The average level of input oscillates at the respiratory cycle frequency. Output from the EPL consists of pulses transmitted by mitral axons to higher level limbic system structures. (Freeman, 1975)

In the EPL, mitral cells act to excite granule cells as well as other mitral cells, while granule cells act to inhibit mitral cells and other granule cells. Normally EPL activity exhibits a high degree of disorder. However, an input surge to the bulb resulting from inspiration, within a range of intensity, will cause the system to bifurcate to a more ordered state, resulting in oscillations of mitral and granule cell activity of equal frequency within the gamma range (40-80 Hz) with the mitral cell activity leading the granule cell activity by one quarter cycle. The cessation of input accompanying exhalation causes the system to bifurcate back to its original disordered state. This activity can be observed by recording mitral cell spike activity or bulbar electroencephalograph (EEG) activity, the latter which has been shown to reflect the voltage level of a granule cell population. (Rall, Shepard; 1968)

Oscillations in the EPL have been shown to vary spatially in rms amplitude. It is thought that patterns of rms amplitude in the bulb may represent conditioning responses of the animal to odors and that the olfactory bulb can store several of these patterns simultaneously by adjusting synaptic strengths between mitral cell populations, thereby acting as an associative memory. (Baird, 1986)

What is unclear is whether neural nets operating under simplified dynamical schemes can store and retrieve such patterns, and if so, to what extent or degree of robustness. The design of hardware associative memory devices which roughly mimic the olfactory bulb's architectonics and presumed processing abilities would benefit from such an analysis, since it would be easier to hardware implement a more simplified dynamical scheme. From a neurophysiological standpoint, we would like to know what types of dynamical behavior a complex system of differential equations can produce that a more simplified system cannot.

The study is confined to assessing the gross temporal dynamics of EPL input response. Thus a lumped model of the EPL is considered (see Figure 1). Spatial properties of the embodiments are assessed using a distributed version of the model by others in this laboratory.

The lumped physiological behavior of the EPL can by modeled by use of second order differential equations in the following form,

$$\ddot{V}_{m1} + r\dot{V}_{m1} + sV_{m1} = k_{ee}\, G(V_{m2}) - k_{ie}(G(V_{g1}) + G(V_{g2})) + \mathrm{INPUT}$$
$$\ddot{V}_{m2} + r\dot{V}_{m2} + sV_{m2} = k_{ee}\, G(V_{m1}) - k_{ie}\, G(V_{g2})$$
$$\ddot{V}_{g1} + r\dot{V}_{g1} + sV_{g1} = k_{ie}\, (G(V_{m2}) + G(V_{m1})) - k_{ii}\, G(V_{g2})$$
$$\ddot{V}_{g2} + r\dot{V}_{g2} + sV_{g2} = k_{ei}\, G(V_{m1}) - k_{ii}\, G(V_{g2})$$

where,

V: average voltage of a particular mitral or granule cell population normalized to its resting state value

k: coupling factors between various cell populations

G: sigmoidal input/output function

r: synaptic delay factor (.94)

s: voltage leakage factor (.158)

INPUT: combined PON and periglomerular input

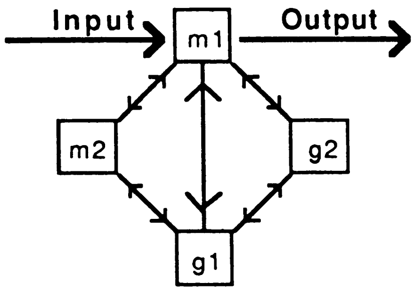


Fig. 1 Embodiment 1, which uses mutual excitation and inhibition. m1, g1, etc. represent the mitral and granule cell populations respectively.

The left hand sides of the equations are the 2nd order differential equations describing the behavior of an uncoupled neuronal population while the right hand side can be seen as the forcing term for the equation: the cell population's input. The coefficients of the 2nd-order differential equations were determined by fitting a pair of exponentially decaying terms to the response of the cell populations to an impulse-like stimulus to the PON. (Freeman, 1975) The first-order coefficient represents a capacitative voltage leak while the 2nd-order term is presumed to represent synaptic delay. Linear relationships were assumed to exist between the amount of neurotransmitter released and magnitudes of post-synaptic potentials (PSPs) as well as between synaptic input and membrane voltage at the soma.

The non-linear term on the right hand side represents the physiologically determined relationship between the average voltage value of the soma of the neurons and the quantity of neurotransmitter released. The sigmoidal curve is normalized such that the resting-state values of cell voltage and pulse magnitude are zero. The equation of the sigmoid is:

$$G(V) = .869 * [1 - \exp(-(\exp(V)-1)/5)]$$

This sigmoid bilaterally saturates, has a near-linear range close to the origin, and is asymmetric with respect to the origin, having a maximal gain above the resting state. (Freeman, 1979). We will refer to this scheme as Embodiment 1. (See Figure 1) Note that the sigmoidal input-output relations introduce non-linearities into these differential equations, forcing us to solve the equations numerically.

THE ALTERNATIVE DYNAMICAL SCHEMES

Outside the brain, self-inhibition and excitation is no longer prohibited. We wish to know if self-excitation and inhibition alters the system's dynamics. We thus can simplify Embodiment 1 to a two-population version, each population capable of exciting or inhibiting itself. (See Figure 2)

The following pair of non-linear ODEs is applicable:

$$\ddot{V}_m + r\dot{V}_m + sV_m = k_{ee}\, G(V_m) - k_{ie}\, G(V_g) + \text{INPUT}$$
$$\ddot{V}_g + r\dot{V}_g + sV_g = k_{ei}\, G(V_m) - k_{ii}\, G(V_g)$$

We will refer to this scheme as Embodiment 2.

We would also like to know what properties the 2nd-order nature of the differential equations confer to the model. Therefore, we also studied a 1st-order version of the above model:

$$\dot{V}_m + sV_m = k_{ee}\; G(V_m) - k_{ie}\; G(V_g) + \text{INPUT}$$
$$\dot{V}_g + sV_g = k_{ei}\, G(V_m) - k_{ii}\, G(V_g)$$

We will refer to this scheme as Embodiment 3. (see figure 2)

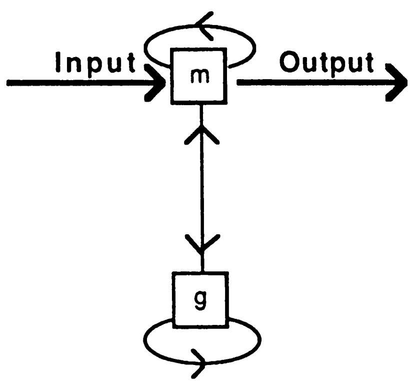


Fig. 2 Schematic diagram for Embodiments 2 and 3, which use self-excitation and inhibition.

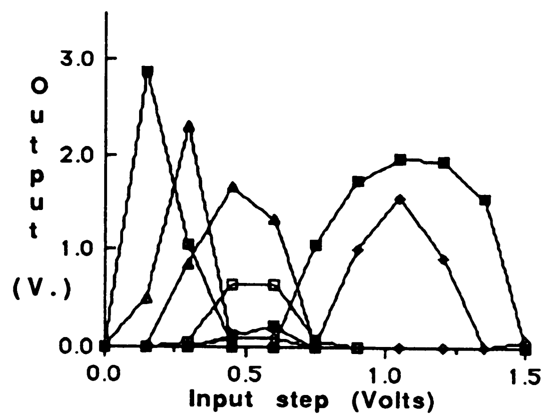


Fig. 3a -- Plot of amplitude of output oscillations for Embodiment 1 for kee values of 0. to 1.6. As kee increases, the curves first increase in height, then drop to 0, and finally grow again in a lower input domain. For all curves: kii = 1. kei = 1.414 kie = .707.

ANALYSIS AND SIMULATION METHODS

The behavior of these embodiments in a linear range was studied by performing a root-locus analysis. Currently we are performing linear stability analysis on the three embodiments.

Their behavior in a non-linear range was assessed by solving the differential equations that governed these embodiments' behavior numerically. The dependencies of the three embodiments' step input responses on changes in connection strengths were assessed. We also used phase plane analysis to study Embodiment 3. Currently we are attempting to analyze the more complex embodiments of the model using phase plane and Hopf bifurcation analyses.

In the input step response study, the input consisted of a voltage step of varying amplitudes. Routines were developed to measure the amplitude of the output response. A set of step input/output curves, each curve corresponding to a particular connection strength, were plotted and superimposed. This enabled us to determine the qualitative effect that a particular parameter had on the input-output relationship of the system.

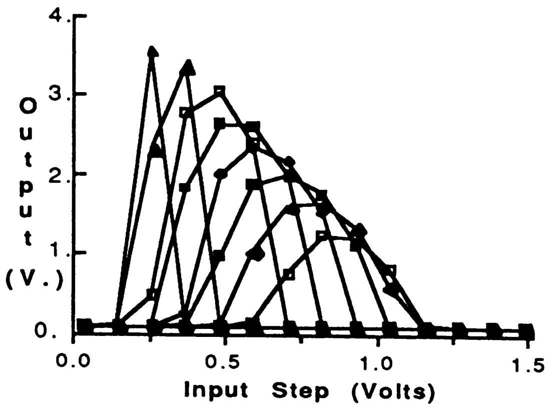


Fig. 3b. -- Equivalent plot for Embodiment 2 As kee increases, the curves increase in height while the input domain moves to the left. For all curves: kii = 1. kei = 2.26 kie = 1.13.

Once again input consisted of voltage steps and the output measured amplitude of the oscillations. Clear differences between embodiments and the varying of different connection strengths became apparent upon comparison of these families of input-output curves.

RESULTS

1. Linear domain analysis

The root loci of the three embodiments were analyzed by varying their connection strength gains. Little qualitative difference of the root loci were found for the three embodiments no matter which gain was being varied. For example, when kee was varied, the root loci for all three embodiments crossed the imaginary axis around kee = .6 All three root loci reached a maximum alpha for kee values of approximately 1.

2. Non-linear domain

Families of curves for Embodiment 1 were much more complex than for the other two models. In virtually all sets of input/output curves within this family, two distinct bifurcation regions were found for values of kee somewhere within the range of .2 to 1.0, giving this family of curves a two-humped appearance, that distinguished it from the other models Both bifurcation point and saturation point were highly sensitive to kee.

Families of curves for Embodiment 1 with kii varying as well the families for the other embodiments were much simpler in appearance, with the bifurcation point, saturation point, and amplitude response monotonically dependent on the changing connection strength.

PHASE PLANE AND LINEAR STABILITY ANALYSIS (Embodiment 3)

Phase plane analysis

Phase plane analysis was simplified by the saturating nature of the sigmoid wave-to-pulse conversion. The saturation implies a maximum firing rate of the populations, which in combination with the capacitative decay results in the boundedness to any solution of the differential equations. By assessing the effect of the connection strengths on the shape of the nullclines we determined that a high kee/kie ratio and a large difference between kii and kei can cause the system to jump to an equilibrium state where a population's output has saturated, a non-physiological or at least pathological condition. Thus the phase plane analysis suggested limits to place on our settings of the four connection strengths.

Linear stability analysis

Determining the position of the signs of the eigenvalues of the linearized system involves analyzing the quantities b and c in the quadratic formula, where

$$x = (-b +/- \sqrt{b^*b - 4c}\ ) /2$$

Thus if $(b^*b - 4c)$ is negative than the eigenvalues are imaginary while their signs can be determined by the combination of signs of b and c. (Odell, 1980) Such analyses shows that the maximal gain of the sigmoid must be to the right of the origin and that a family of input/output curves for Embodiment 3 cannot exhibit the complex properties that such curves corresponding to Embodiment 1 can exhibit.

DISCUSSION

We found a complex dependence of Embodiment 1's input/output properties on kee. Such complex dependencies were not found for kee for the other two embodiments nor on kii for any of the embodiments.

The two distinct input domains for which the system bifurcates could allow two sub-systems within a distributed system using embodiment 1's dynamical scheme to perform different associative memory tasks. Each sub-system could be sensitive to distinct input domains. Furthermore, if the more sensitive sub-system fed into the less senstive sub-system, input limited to the former could spread throughout the system, causing the entire system to bifurcate into an oscillatory mode.

Physiologically, this study suggests that the olfactory bulb's associative memory relies upon an array of plastic mutual excitatory connection strengths. This study's conclusions also implore engineers to pay attention to what the brain is telling us.

Perhaps only after studying the global behavior of distributive versions of these embodiments can we assess the virtues and limitations of each of the embodiments presented here. The present study has yielded tools with which to attack this higher-order problem.

NOTE: Supported by National Institute of Mental Health Grant, MH-06686

REFERENCES

Baird, B.: Nonlinear Dynamics of Pattern Formation and Pattern Recognition in the Rabbit Olfactory Bulb. *Physica 22D,* 1986, 150-175.

Freeman, W.J.: *Mass Action in the Nervous System.* New York: Academic Press, 1975

Freeman, W.J.: Nonlinear gain mediating cortical stimulus-response relations. *Biological Cybernetics,* 1979, **33**, 237-247.

Odell, G.M. In: *Mathematical Models in Molecular and Cellular Biology.* Cambridge University Press, 1980.

Rall, W., Shepherd, G.M. Theoretical reconstruction of field potentials and dendrodendritic synaptic interactions in the olfactory bulb. *Journal of Neurophysiology,* 1968, **31**, 884-915

CONNECTIVITY IN THE OBSERVED PORTION OF AN AUDITORY NEURONAL NETWORK.

Ismael E. Espinosa, PhD
Universidad Nacional Autónoma de México
Facultad de Ciencias, Laboratorio de Cibernética
Ciudad Universitaria, D.F. 04510 MEXICO

INTRODUCTION The role of auditory cortex in both sound localization and pattern discrimination has been demonstrated by observing that ablations of it disrupt these functions. The results of a number of experimental and clinical investigations in man and animals show evidence that severe deficits occur for several auditory discriminations, among others[2,9,10]: a) Change in temporal pattern of sounds, b) Comparison of signals involving recent or short-term memory, c) Identification of temporal order.

We have assumed, invoking the results of the ablation experiments mentioned above, that sound patterns are strong activators of neurons in the auditory cortex. We used a combination of well developed neurophysiological techniques in order to obtain a substantial body of data from neuron clusters in the primary auditory cortex (AI) of the cat[4,5]. The main goal of this research was to study functional connectivity variations as a function of modifications in the temporal pattern of acoustic stimulation.

STIMULUS The stimuli used were six very simple melodies (3 sequential tone bursts of different frequency, around best frequency) each one presenting a different temporal pattern and followed by a pause of approximately one second. The neural response to the third tone in the sequence was compared with the response to the first tone in the sequence and also with the spontaneous conditions.

ANALYSIS The comparison consisted in analyzing pairs of edited crosscorrelograms[4] after taking into account the corresponding shift-predictor histograms[3]. Each complete recording was separated into 19 edited records for separate analysis. 18 of these records corresponded to the 18 stimuli and had a maximum edit window duration of 195 msec to avoid the next stimulus presentation. The 19th record corresponded to the spontaneous period between sequences and had an edit window duration of 500 msec to avoid the next sequence. These procedures made possible the calculation and comparison of cross-correlation that was associated with various particular stimuli. Because of the stimulus presentation design it is possible to compare correlation associated with physically identical stimuli that are placed at different positions in a three-tone sequence. We made the appropriate normalizations in the histograms and eliminated the synchronizing effects of the stimulus by using the shift-predictor[3] so that the simultaneous effect of the stimulus on the neurons firing was eliminated and only direct effects (synaptic) were left.

RESULTS Our results show several levels of complexity which can be summarized as follows. For spontaneous or no-stimulus conditions (NS) there is a connectivity diagram D0. For simplicity, let us assume only three different stimuli S1, S2, and S3. We found evidence that each stimulus produces a corresponding connectivity diagram, that is, stimulus S1 produces connectivity diagram D1, S2 produces connectivity diagram D2, and S3 produces connectivity diagram D3. Moreover, if we apply as a stimulus the sequence S1-S2 we found a connectivity diagram D4 (not D2) and if we apply S2-S1 we found connectivity diagram D5 (not D1). Now, when we applied the sequence S1-S2-S3 we found the connectivity diagram D6, and the connectivity diagram D7 when we applied S2-S1-S3. It is noteworthy that connectivity diagrams {D1, D2, ... , D7} are variations of D0, the connectivity diagram for spontaneous conditions. The most interesting result is related to the situation above where S1-S2-S3 produces connectivity diagram D6 and S2-S1-S3 produces connectivity diagram D7. In these two cases, in which S3 is the same physical stimulus in the third position, the history of previous tones is not. In one case we have S1-S2, and the reversed situation S2-S1 in the other case.

We have discussed elsewhere[4] the results summarized above. Here we want to discuss subtler differences noticed in a refinement of the analysis of our data. We have shown that connections in the spontaneous activity diagram get reorganized according to the stimulus: some connections stay, others dissappear. This description gives a picture of an ON-OFF condition for the functional connections. This is due to the visual analysis that was performed on the cross-correlation histograms based on the shape of such histograms. To be able of doing a more accurate and quantitative analysis we took a +/- 25 msec window in the histograms and measured the area under it. We found that connections that do not seem to change or that seem to dissappear are actually undergoing gradual changes that are function of variations in the stimulus. These changes are not the same for all stimulus conditions but for a given neuron pair there is a maximum for one of them which could work as an index for a sequence classifier. We took as a reference the crosscorrelated activity for the case when there is no previous history of tones, and noticed that the gradual changes occur above or below such reference level for tones with a previous history of two tones.

DISCUSSION These experiments demonstrated how variations in effective connectivity measured by the cross-correlograms depend on the temporal order in which the tones are presented, i.e., on the recent history of stimulus. Differences in functional connectivity due to detection of previous tone-burst history and sequence classification may represent different mechanisms. The former seems to depend on the interplay of excitatory-inhibitory influences due to poststimulus effects and could be responsible for a short-term memory system. The latter, possibly using a transformation of that system, classifies sequences according to the correlated firing they evoke. The short-term memory system is related to correlated firing rates whereas sequence classification as defined here requires, in addition, the presentation of patterned tone sequences. In other words, the short-term memory property seems to be present for any tone sequence irrespective of its temporal patterning.

We have shown that when we infer a connectivity diagram using the cross-correlation histograms shape we get different ON-OFF diagrams according to the stimulus parameters[4]. However, if we measure a histogram area, instead of only observing shape, then we detect the graded nature of the functional connection. There is a result in the literature that is very appropriate for a plausible interpretation of our results. It has been shown that the cross-correlation histogram is an estimation of the magnitude of the post-synaptic potential[6] and this is related to synaptic strength. Since what we observed were variations in the magnitude of a fixed window in the cross-correlation histogram, we could say that such variations represent changes in the synaptic strengths or weights of the connections. This is a very important observation under the framework of contemporary artificial neural networks. In the context of these experiments paradigm and design of the stimulus we have a similar situation to that'of associative memory[7]. Here we cannot say anything about the architecture of the biological neural network since we are measuring from only a portion of it, and we do not know either if the recorded neurons belong to an input, output or hidden layer. In any case, the changes in the cross-correlation histograms, as related to modifications in the weights, remind us of the changes required in an artificial associative memory when several associations are stored in it. This is done using learning algorithms with varying degrees of recall accuracy[11]. As before, we cannot say anything about the biological learning rules. On the other hand, in making analysis by neuron pairs we are certainly missing collective phenomena. If we had a tool for analyzing neuron ensambles we would possibly be able of saying more about the collective organization of connectivity. Also, since we only register a limited number of neurons we need a tool for extrapolating and reconstructing the whole network or module involved, something as the observers used in control theory for reconstructing the state vector from a limited number of outputs or measurements[8]. Nevertheless, the results presented, even if very limited, are encouraging for several reasons. First, they help to refine the knowledge about the biological adaptation of weights and this should give hints for improving the weight organization in artificial neural networks. Second, they support the need of more experiments where separable multi-unit recording is made; we would gain much insight from simultaneous recordings of 100 to 200 neurons which is a very small sample of neurons compared to the thousands used in an artificial neural network but complexity and number have to be traded off to gain in understanding.

We can suggest a network -of unknown architecture so far- that at the neuron level can detect the stimulus as guaranteed by the PST histograms. At the network level we can make slices in the network hyperplane using neuron pair crosscorrelations that show that two neurons interaction perform some sort of redundant classification of some of the stimulus parameters. It is likely that higher order crosscorrelations, if it were possible to build them, would show classification as an emergent property of the network. Such mechanism could work as an auto-training facility, in the jargon of artificial neural networks. It is also likely that the network implements short-term memory; in other areas of auditory cortex, other structures would implement learning, we would assume that all of these networks would be in communication or bound together[1]. Finally, in the awake and trained animal, an interaction of networks including the ones mentioned before and actuators (muscles and visuo-motor coordination) would implement a behavior. For instance, the animal would learn to behave in one way when listening to melody S1-S2-S3 and in other way when listening to melody S2-S1-S3.

REFERENCES

1.- DAMASIO, A.R., The brain binds entities and events by multiregional activation from convergence zones, Neural Computation, 1: 123-132, 1989.

2.- DIAMOND, I.T. AND NEFF, W.D., Ablation of temporal cortex and discrimination of auditory patterns, J. Neurophys., 20: 300-315, 1957.

3.- DICKSON, J.W. AND GERSTEIN, G.L., Interactions between neurons in auditory cortex of the cat. J. Neurophys. 37: 1239-1261, 1974.

4.- ESPINOSA, I.E. AND GERSTEIN, G.L., Cortical auditory neuron interactions during presentation of 3-tone sequences: effective connectivity, Brain Research, 450: 39-50, 1988.

5.- GERSTEIN, G.L., BLOOM, M.J., ESPINOSA, I.E., EVANCZUK, S., AND TURNER, M.R., Design of a laboratory for multineuron studies. IEEE Trans. Syst., Man and Cyber., 13: 668-676, 1983.

6.- KIRKWOOD. P.A., On the use and interpretation of cross-correlation measurements in the mammalian central nervous system, J. Neurosci. Meth., 1: 107-132, 1979.

7.- KOSKO, B., Bidirectional Associative Memories, IEEE Trans. Syst., Man and Cyber., 18: 49-60, 1988.

8.- LUENBERGER, D.G., An introduction to observers, IEEE Trans. Autom. Control, 16: 596-602, 1971.

9.- LURIA, A.R., Higher cortical functions in man, Basic Books, Inc., 4th printing, 1973, pp. 94-122.

10.- NEFF, W.D., DIAMOND, I.T. AND CASSEDAY, J.H., Behavioral studies of auditory discrimination: central nervous system, In Keidel, W.D. and Neff, W.D. (Eds), Handbook of sensory physiology. Auditory system. Physiology. Behavioral studies. Psychoacoustics, Vol V/2, Springer-Verlag, 1975, pp. 307-400.

11.- STILES, G.S., A quantitative comparison of the performance of three discrete distributed associative memory models, IEEE Trans. Computers, 36: 257-263, 1987.

Fast synaptic modulation provides a ubiquitous mechanism to support an instruction-data distinction in biological neural networks

Chris Fields
Knowledge Systems Group, Computing Research Laboratory
New Mexico State University, Las Cruces, NM 88003-0001, USA

Introduction

Artificial neural networks (ANNs) are conventionally designed as special-purpose machines that accept a single data vector as input, and calculate a single data vector as output. This design convention reflects a tacit assumption that perceptual systems provide biological neural networks solely with data to process, not with instructions specifying which function to compute. Indeed Kohonen (1988, Ch. 9) has explicitly argued that biological neural networks are incapable of accepting instructions or executing stored programs, concluding that "genuine neural computers should not be programmable at all" (p. 268). The view that inputs to biological systems can function as instructions seems, however, quite appropriate in some cases: Ron et al. (1989), for example, explicitly use the term "programming" to describe the effect of inputs on the collicular saccade-generation system. Direct programmability by inputs could, moreover, potentially solve some of the problems with learning efficiency and generality that plague ANNs; biological neural networks might, therefore, also be expected to incorporate programmability to improve efficiency and generality.

The purpose of the present paper is to suggest that some biological neural networks function as *interpreters* that accept and process both data and instruction streams in real time, and to outline an architecture which would accomplish this. Both instructions and data may be derived from sensory input; a programmable system need not store its programs explicitly. The instructions may specify the functions to be executed on current data, or as in the case of instructions to the saccade system, control the acquisition of additional data. Fast synaptic modulation provides one mechanism for implementing real-time instruction interpretation in neural networks, just as slow synaptic modulation, e.g. the activation of NMDA receptors, provides a mechanism for slow (Hebbian) learning (Cotman et al., 1988; Gustafsson and Wigstrom, 1988). The view that neural networks can function as interpreters allows a number of common anatomical and physiological motifs not readily explained by conventional ANN models - e.g. the ubiquity of neighborhood-preserving projections between layers, local modulatory interactions, and functionally distinct inputs to single cells - to be understood as supporting the modulation of data processing by signals, often transmitted from distant populations of neurons, that encode instructions.

Architectures supporting an instruction-data distinction

Any neural network can be described as a programmable system by interpreting its input vector space U as a direct sum $U = V \oplus W$ of two disjoint subspaces, a data subspace V and an instruction subspace W. Input vectors u are then treated as sums of data vectors v and instruction vectors w. An instruction vector may be viewed as encoding a single complex instruction, or a sequence of simpler instructions. The functions specified by the instructions can be studied by fixing w and varying v; conversely the range of available instructions can be studied by fixing v and varying w. The input lines carrying data and instruction words to a conventional microprocessor may be viewed as encoding (binary) vectors in just this way; hence this characterization of the instruction-data distinction applies equally to neural networks and conventional computers.

The instruction-data distinction is gratuitous in a standard ANN in which all of the nodes, and all of the connections, are functionally identical. This is, however, not the case in biological neural networks, which typically comprise cells of functionally distinct types that have synapses of functionally distinct types. While the functions computed by biological neural networks can presumably - given the Church-Turing thesis - be computed by standard ANNs, such emulations may not shed much light on how the functions of interest are actually computed by nervous systems. The instruction-data distinction may prove useful in understanding the algorithms actually employed by such systems, and their implementations in biological neural networks.

Figure 1 shows a simple, feed-forward ANN architecture designed to support an instruction-data distinction. The data vector is processed by a set of layers, which are separated by transmission delays to enforce sequential processing. The instruction vector is divided into distinct components, which are viewed as names of instructions. Each instruction name is input, together with a component of the output of the previous processing layer, to a content-addressable memory (CAM), which encodes a full instruction. The CAMs provide additional inputs to the units in the corresponding processing layers. The inputs from the CAMs are assumed to gate, modulate, or otherwise control the behavior of the units in the processing layers; hence each instruction name encoded by the instruction vector effectively controls the processing of the data vector by a single layer.

In order for the architecture shown in Fig. 1 to be distinguishable from a conventional feed-forward ANN, the connections from the CAMs to the processing layers must be functionally distinct from those between the processing layers. The principle requirement for modelling programmable networks is, therefore, an ability to model distinct classes of synapses. This may be done in the conventional ANN formalism by introducing unit-specific signal and feedback functions into the node equation (Cohen and Grossberg, 1983; Grossberg, 1988), by introducing units having multiplicative connections that "set the weights" of the additive connections in the network (McClelland, 1986; Pollack, 1987), or by introducing units that multiply instead of summing their inputs (Durbin and Rumelhart, 1989). McClelland (1986) and Pollack (1987) both motivate their formalisms as methods of achieving some degree of programmability.

Fast modulation as a programming mechanism

The requirement, implicit in Fig. 1, for a class of inputs that control the computational behavior of a set of units is met, in biological neural networks, by synaptic inputs that effect fast modulation of the responses of other synapses. A variety of such mechanisms have been characterized biochemically (reviewed by Levitan, 1988). While less is known about the functions of modulatory inputs in alterating the behaviors of specific cells, some pathways involving multiple modulatory synapses have been partially characterized (e.g. McCormick, 1989). It seems reasonable to hypothesize, based on the ubiquity of receptor-channel systems that can support modulatory interactions, that such interactions are likewise ubiquitous, and are of functional importance.

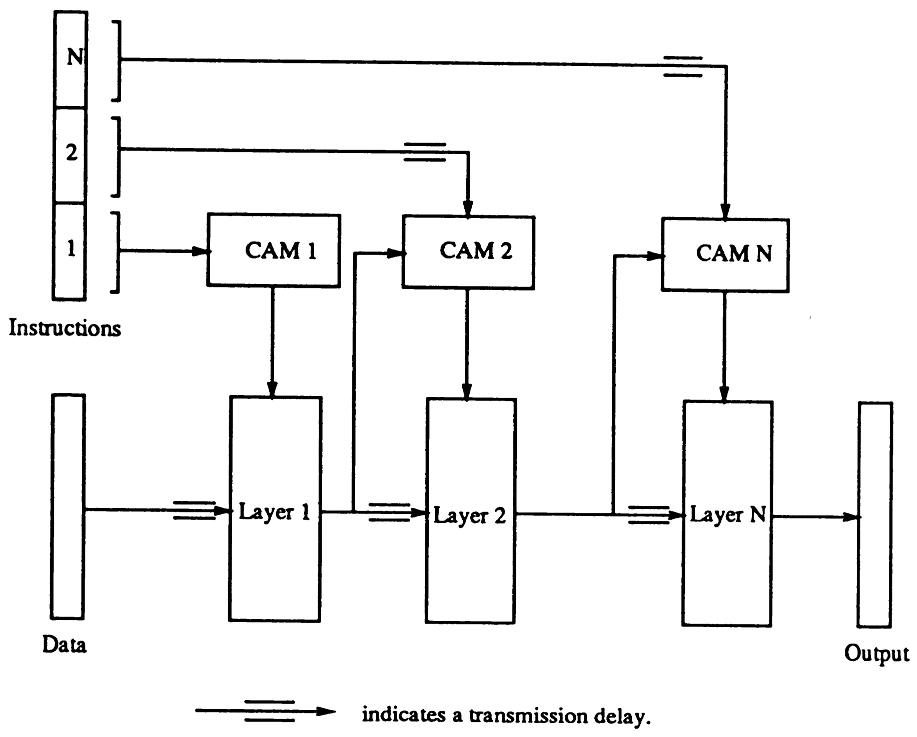


Fig. 1: A programmable ANN architecture. CAM = Content addressable memory.

If fast modulation by secondary synapses is in fact functionally ubiquitous, and if it indeed supports processing architectures of the general type shown in Fig. 1, then a number of common motifs of brain organization may be seen as supporting programmability in biological neural networks. In particular, Fig. 1 suggests that an organization into sequential layers, each of which also receives secondary, modulating input from elsewhere in the system, will be ubiqitous. Such an organization apparently occurs in the visual system, in which layers primarily devoted to one processing stream appear also to modulate, and be modulated by cells belonging to other processing streams (DeYoe and Van Essen, 1988).

Electric circuit models appear to provide the best formalism for investigating such modulatory interactions, as both the individual conductances and their characteristic time constants can be represented in such models (e.g. Wilson and Bower, 1989). A generic processing element that can be used to study both facilitory and inhibitory interactions is currently being developed in this laboratory.

Computational advantages of programmability

Investigation of the encoding and execution of instructions by biological neural networks using a combined experimental and modelling strategy can be expected to yield useful design principles for programmable ANNs. The usefullness of conventional ANNs is limited both by the inefficiency of standard learning algorithms, which require solving global nonlinear interpolation problems, and by the need to add processing nodes to the system - thereby further increasing the time required for learning - to cope with increases in the complexity of the function being computed. The development of ANN architectures that support programmablity addresses both of these issues: programmable ANNs would only need to learn the functions associated with each interpretable instruction, and complex functions could be executed by a network comprising a set of layers that executed a series of instructions sequentially. A programmable ANN would, therefore, combine the positive features of both conventional ANNs and general-purpose sequential computers.

Acknowledgement

This research was partially supported by NASA Innovative Research Program grant NAGW-1592 to the author and J. Barnden.

References

Cohen, M. and S. Grossberg (1983) Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Transactions on Systems, Man, and Cybernetics* 13: 815-826.

Cotman, C., D. Monaghan, and A. Ganong (1988) Excitatory amino acid neurotransmission: NMDA receptors and Hebb-type synaptic plasticity. *Annual Review of Neuroscience* 11: 61-80.

DeYoe, E. and D. Van Essen (1988) Concurrent processing streams in.monkey visual cortex. *Trends in Neurosciences* 11: 219-226.

Durbin, R. and D. Rumelhart (1989) Product units: A computationally powerful and biologically plausible extension to backpropagation networks. *Neural Computation* 1: 133-142.

Grossberg, S. (1988) Nonlinear neural networks: Principles, mechanisms, and architectures. *Neural Networks* 1: 17-61.

Gustafsson, B. and H. Wigstrom (1988) Physiological mechanisms underlying long-term potentiation. *Trends in Neurosciences* 11: 156-162.

Kohonen, T. (1988) *Self-Organization and Associative Memory,* 2nd Edition. Berlin: Springer.

Levitan, I. (1988) Modulation of ion channels in neurons and other cells. *Annual Review of Neuroscience* 11: 119-136.

McClelland, J. (1986) Resource requirements of standard and programmable nets. In: D. Rumelhart and J. McClalland (Eds) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Cambridge, MA: MIT. pp. 460-487.

McCormick, D. (1989) Cholinergic and noradrenergic modulation of thalamocortical processing. *Trends in Neurosciences* 12: 215-221.

Pollack, J. (1987) *On Connectionist Models of Natural Language Processing.* Ph.D. Thesis, University of Illinois.

Ron, S., T. Vieville, and J. Droulez (1989) Use of target velocity in saccadic programming. *Brain, Behavior, and Evolution* 33: 85-89.

Wilson, M. and J. Bower (1989) The simulation of large-scale neural networks. In: C. Koch and I. Segev (Eds) *Methods of Neuronal Modeling.* Cambridge, MA: MIT. pp. 291-333.

FUNCTION MAPPING AND ITS RELATIONSHIP WITH THE PSYCHOPHYSICAL FUNCTIONS IN THE THEORY OF NEURAL NETWORKS

J. G. Figueroa£, C. Flores, E. Vargas & M. Romero
£ UNIVERSIDAD AUTONOMA DEL ESTADO DE MEXICO
UNIVERSIDAD AUTONOMA METROPOLITANA-IZTAPALAPA

Laboratorio de Sistemas Complejos
Ap. Postal 70-499. C.P. 04510
México, D.F. MEXICO

A modern, central problem in neural network research is to find a relationship between the formalisms at use and a general neurocomputational theory, as it can be the problem of function mapping using different kinds of perceptrons.

The problem of representing continuous three valued functions by using continuous functions of less than three variables arises from the 13th problem of Hilbert (Lorentz, 1976). This problem request the demostration that the seventh-grade equation

$$X^7 + xX^3 + yX^2 + zX + 1 = 0 \qquad (1)$$

CANNOT be solved with the help of any continuous two valued function.

Kolmogorov (1957) contributes to the solution of this problem with a theorem in which he showed the existence of fixed functions Ξpq, continuous and increasing on the interval I = [0,1] such that any continuous function defined in I can be written as

$$f(x_1,\ldots,x_n) = \sum_{q=1}^{2n+1} gq \left(\sum_{p=1}^{n} \Xi pq \; (xp) \right) \qquad (2)$$

with gq continuous functions of a suitable chosen variable.

Some authors have simplified Kolmogorov's presentation of **f**. For example:

$$f(x_1,\ldots,x_n) = \sum_{q=1}^{2n+1} g\left(\sum_{p=1}^{n} \sigma p \Xi q(xp) \right) \qquad (3)$$

with σp constants. Expanding the preceeding formula

$$\begin{aligned} f(x_1,\ldots,x_n) = {} & g(\sigma_1\Xi_1(x_1) + \sigma_2\Xi_1(x_2) + \ldots + \sigma_n\Xi_1(x_n)) \\ & + g(\sigma_1\Xi_2(x_1) + \sigma_2\Xi_2(x_2) + \ldots + \sigma_n\Xi_2(x_n)) \qquad (4) \\ & + \ldots + g(\sigma_1\Xi_{2n+1}(x_1) + \ldots + \sigma_n\Xi_{2n+1}(x_n)) \end{aligned}$$

we obtain the simplification carried out by Lorentz (1966) and Sprecher (1964)

In other words, Kolmogorov proves that any continuous, n-valued function can be represented with the aid of functions of a single variable. Following Kolmogorov's work, it is possible to represent functions in terms of other functions; and it can be used superposition of functions:

$$f(x,y,z) = F[g(x,y),h(q[x],k[z])] \qquad (5)$$

or linear superpositions in which combinations of fixed functions with variable ones are considered, such as

$$f(x,y,z) = \sum_{i=1}^{m} p_i(x,y)g_i[q_i(x,y)] \qquad (6)$$

in which p_i and q_i are fixed.

Recent work (Cybenko, 1988) has demonstrated that it is enough to use neural networks with two hidden layers to represent sets of continuous functions.

Hilbert's conjecture, and Kolmogorov's reply, are very important if the relationship is made with experimental work in sensorial psychophysics. Within this framework, psychophysicians tried to find which is the function that maps a physical dimension into a psychological one. The answer to this question has been the development of psychophysical functions (Stevens, 1975; Falmagne, 1985) where we know that, in general, all sensorial magnitudes are psychologically mapped with a power function of the type $Y=aX^b$, **a** and **b** constants empiricaly determined.

Nowadays, it has been found that the **b** exponent is given by the sensorial modality. However, in previous work (Figueroa et al., 1974; Figueroa et al., 1981; Figueroa et al., 1982a; Figueroa et al., 1982b) we have experimentally demonstrated that the **b** exponent is directly linked with the cognitive abilities of the subjects, particulary with the information processing speed in tasks that require the use of "internal" images. When the exponent is corrected with this value of information processing, its variability is drastically reduced. The relevance of this kind of work is that, not long ago, it was not known "what to do" with such psychophysical functions. However, we can now suppose that probably, if a superposition of functions is made -as Kolmogorov proved-, it is possible to "represent" the physical world in a neurocomputational system of a general kind known as "perceptrons" with at least two hidden layers. The transformation between the external physical event to the subject and its internal representation is given by the previously adressed psychophysical functions.

The relationship of the power functions in sensorial psychophysics makes possible to find a very strong foundation of the neural networks as information representing systems.

REFERENCES

Cybenko, G. (1988). Continuous valued neural networks with two hidden layers are sufficient. Tech. Rep., Dept. of Computer Science, Tufts University, march.

Falmagne, J. C. (1985). Elements of psychophysical theory, Oxford University Press, New York.

Figueroa, J. G., Solis, V. M. González, E. G. (1974). The possible influence of imagery upon retrieval and representation in Long Term Memory. Act. Psych., 38, 424-428.

Figueroa, J. G.; M. Carrasco; G. Hernandez. (1981). Effects of Human Information Processing on psychophysical functions. UNAM, VII Int. Bioph. Cong. and I Panam. Biochem. Cong., February.

Figueroa, J. G., Carrasco, M. y Sarmiento, C. (1982a). Magnitude Estimation of the Psychophysical Function and its Relationship with the Central Information Processes. Mex. Cong. Phsyc., México, D.F. July, (in spanish).

Figueroa, J. G., Carrasco, M., Sarmiento, C. y Bravo, P. (1982b). Distinction between Sensorial and Memory Aspects in Visual Psychophysical Judgements of Magnitude in Humans. XXV Nat. Cong. Physiol. Sc., Guadalajara, Jal. México. July, (in spanish).

Kolmogorov, A. N. (1957). On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition, Dokl., 114,679-681.

Lorentz, G.G. (1966) Aproximation of Functions, Holt, Rienhart & Winston, New York.

Lorentz, G.G. (1976) The 13th Problem of Hilbert, Proc. of Symp. in Pure Math., vol. 28, pg. 419-430

Sprecher, D.A. (1964) On the Structure of Continuous Functions of Several Variables, Trans. Amer. Math. Soc., vol 115, pg. 340-355.

Stevens, S. S. (1975). Psychophysics: introduction to its perceptual, neural, and social prospects, John Wiley & Sons.

Pattern Recognition in Primate Temporal Cortex: But is it ART?

Paul M. Gochin. Department of Psychology, Princeton University Princeton, NJ 08544-1010.

Introduction: Biological mechanisms of visual pattern recognition have received considerable attention in "neural network modeling" (eg. [9]). However, it has been information about the earlier or "more" peripheral cortical areas, particularly striate cortex, which have served as the basis for most of these models. Early stages of visual information processing in the brain appear to serve a preparatory function modifying the form of incoming information for use by higher-order visual areas. It is these higher-order areas, especially inferior temporal cortex in the primate) which show properties desired by "neural network" modelers, such as image size and position invariance, as well memory and selective attention. Models of high-level vision have been suggested by visual system investigators, however these are usually restricted to the context of the investigations and do not provide implementation level details (eg. [25]). The objective of this report is to suggest, and support the usefulness of adaptive resonance theory (ART)[9] as a mechanistic model of high-level biological vision, and in addition to recommend that inferior temporal cortex be closely evaluated as a source of information for further development of "neural network" models.

Form vision in the primate: Processing of visual information begins at the earliest stage of primate vision, the retina. The output of retinal ganglion cells is luminance normalized and enhances edge contrast. Yet, the information at this point still remains essentially a dot matrix. Already, however, a branching process has begun separating information into channels presumably for feature, and motion processing. At the first cortical stage of processing, striate (V1) single neurons begin to show specialization for form (e.g., selectivity for local orientation of edges), motion direction [12] and depth detection [21]. Neurons at the next stage, V2 show evidence of pattern completion [11]. Form processing continues in V4 where neural responses can show limited invariance to edge location. Local coloration appears to be represented relative to the surrounding area [1,27].

A branching in regional specialization of cortex occurs at the level of V2, with a so-called "dorsal" stream destined for parietal cortex involved in processing spatial information and, of particular interest here, a "ventral" stream terminating in IT which processes form. The contrast in properties of IT neurons compared to striate is striking. Whereas striate neurons process relatively small focal regions of the visual field, IT neurons almost always include the central visual field with highest acuity and a relatively large surrounding area. Striate cortex is highly organized with a retinotopic map and repeating modules processing each point in the visual field for a stereotyped set of parameters. IT is not retinotopically organized and neurons dispersed throughout IT can become active for any given stimulus. Although a small portion of IT neurons appear to be relatively selective for specific stimuli (such as faces), the majority respond to a diverse array of visual patterns [8]. Typically, IT neurons respond to visual patterns in a manner which is invariant with respect to retinal location and pattern size [7,23]. Perhaps the most striking difference between IT and striate is that IT has been shown to manifest attention [19,22], show evidence of short term memory [2,18] and be involved in long term visual memory [17].

Functional similarities between ART and IT: Correspondence between theoretical visual system components [9] such as the Boundary Contour System (BCS) and biological subdivisions remain unclear, however, the Object Recognition System which utilizes ART may be analogous in function with IT (in conjunction with other brain areas). The adaptive resonance model is constructed as a multi-layer system, each layer of which utilizes a form of "lateral inhibition" to enhance contrast in a parameter space appropriate for that layer. These layers are then interconnected both feedforward and feedback. Critical to the functionality of the system is top-down influence which serves in selective attention and pattern matching. Also critical is that some form of habituation exist so that once a pattern has been unsuccessfully tried as a match it is not immediately tried again. Also, a general control of "vigilance" is required to modulate selectivity of an acceptable pattern match.

A minimum requirement for acceptance of ART as a model of high-level visual function is that these fundamental principles be consistent with experimental observations. The first point is that of lateral inhibition. This mechanism has its historical origin [10] in the retina. Recently we [15] have found signs of widespread inhibitory interactions in IT cortex. Specifically, the discharge rates of single neurons are decreased by the simultaneous introduction of several visual patterns, relative to the introduction of a single pattern. This occurs irrespective of the details of the patterns, and does not require active attention to the stimulus by the animal.

The second point is that top-down influences should be evident in IT cortex. There is substantial anatomical evidence that connections between most visual cortical areas are reciprocal [24]. Furthermore, physiological evidence has been available for some time [5] which suggest top down influences in IT, i.e. that the neural response to a stimulus can be dependent on extra-retinal factors. More specifically, the response to an identical stimulus can differ depending on behavioral context during which the pattern is viewed. Furthermore, it has been shown that attention can be spatially directed, since response to a stimulus within an IT receptive field can be modulated by instructing an animal to behaviorally direct its attention to differing points within the visual field [19].

One of the unique features of adaptive resonance is its explicitly defined sequential search algorithm. Critical to the successful operation of this system is a means of temporarily locking out patterns which have already been tried. This lock-out can be achieved with a habituation-like process. It has been noted in several investigations of IT that neurons tend to vary their response with successive presentation of the same stimulus [8]. We have quantitatively investigated this phenomenon and demonstrated that most neurons in the anesthetized animal show this property [14].

Though evidence for variation in vigilance has not yet been reported for IT, such a demonstration has been reported for area V4, which directly supplies input to IT. In these experiments it was shown that neural selectivity for the orientation of bar stimuli could be modulated by the degree of attention given to the task [25].

IT is believed to be the final stage in visual form processing [16] and a repository for visual memory. Thus, the way information is represented in IT could serve as a guide for "neural networks". It seems to be common, at least during the development of models (e.g. [9]), to employ "winner takes all" algorithms. For many reasons this approach is ultimately not desirable, and in the context of IT this is clearly not the mechanism employed. The simple fact that IT can be studied by sampling a very small portion of its neurons demonstrates that a rather large portion become activated during most visual conditions. From the standpoint of the physiologist this raises the issue of whether IT is · simply non-selective. Although it is clear to those who study IT that it's neurons are highly selective, until recently there was no quantitative demonstration of the form recognition capacity. In a recent study we have compared the ability of a single neuron to partition a small set of complex test stimuli with a small population of such neurons. While some single neurons were reasonably precise in identifying the stimuli, a selected small group approached perfect partitioning. These observations support the hypothesis that information in IT is distributed across a population of neurons [4].

Although IT does not appear to be retinotopically organized, there is evidence that some form of spatial organization exists. We [3] have observed that neurons with similar stimulus selectivity properties occur in clusters. Furthermore we have observed that neurons in proximity have a greater tendency to share input than those more distant. We have also observed excitatory connections between nearby neurons within IT (Both observations have also been made in striate cortex, eg. [13]). Local clustering of stimulus properties and spatially biased input projections are consistent with requirements for masking fields. The excitatory interconnection between neurons does not appear to be part of the general principles of ART, but has been employed for special purpose circuits such as in Boundary Contour System modeling [9], and may be predicted if Short Term Memory involves a population of neurons. Other models such as "neuronal group selection" [20] explicitly require excitatory local interactions and localization of functions (i.e. map formation).

Conclusions: Adaptive resonance theory, more than any other "neural network" model, embodies properties which can be directly related to actual neural networks involved in high-level form vision. This does not mean that ART should be considered the mechanism of biological pattern vision, but rather that it could serve as a base to be sculptured by biological observations.

References

1) Desimone, R., Schein, S.J. (1987). J. Neurophysiol. 57:835-868.
2) Fuster, J.M., Jervey, J.P. (1982). J. Neurosci. 2:361-375.
3) Gochin, P.M., Miller, E.K., Gross, C.G., Gerstein, G.L. (1988). Soc. for Neuro. Sci Abst.
4) Gochin, P.M., Gerstein, G.L., Miller, E.K., Gross, C.G. (1989). (In preparation).
5) Gross, C.G., Bender, D.B., Gerstein, G.L. (1979). Neuropsychologia 17:215-229.
6) Gross, C.G., Desimone, R., Albright, T.D., Schwartz, E.L. (1984), in Study Group on Pattern Recognition Mechanisms, C. Chagas, R. Gattass, and C.G. Gross Eds., (Pontifica Academia Scientiarum, Vatican City.
7) Gross, C.G., Mishkin, M. (1977). In Lateralization of the nervous system, S. Harned, R. Doty, J. Jaynes, L. Goldberg, G. Krauthamer Eds.
8) Gross, C.G., Rocha-Miranda, C.E., Bender, D.B. (1972). J. Neurophysiol. 35:96-111.
9) Grossberg, S. (1988). In: Neural Networks and Natural Intelligence, S.

Grossberg Ed. MIT Press, Cambridge, Mass.
10) Hartline, H.K., Ratliff, F. (1957). J. Gen. Physiol. 40:357-376.
11) Heydt, R., Peterhans, E. (1989). J. Neurosci. 9:1731-1748.
12) Hubel, D.H. and Wiesel, T.N. (1968). J. Physiol. 195:215-243.
13) Michalski, A., Gerstein, G.L. Czarkowska, J., Tarnecki, R. (1983). Exp. Brain Res. 51:97-107.
14) Miller, E.K., Gochin, P.M., Gross, C.G. (Unpublished observation).
15) Miller, E.K., Gochin, P.M., Gross, C.G., Gerstein, G.L. (1989). Soc. for Neuro. Sci Abst.
16) Mishkin, M. (1972), in The Brain and Human Behavior, A.G. Karczmar and J.C. Eccles Eds. (Springer-Verlag, New York.
17) Mishkin, M. (1982). Phil. Trans. R. Soc. Lond. B 298:85-95.
18) Miyashita, Y., Chang, H.S. (1988). Nature 331:68-70.
19) Moran, J., R. Desimone. (1985). Science 229:782-784.
20) Pearson, J.C., Finkel, L.H. and Edelman, G.M. (1987). J. Neurosci. 7:4209-4223.
21) Poggio, G.F., Fischer, B. (1977). J. Neurophysiol 40:1392-1405.
22) Richmond, B.J., Wurtz, R.H., Sato, T. J. (1983). Neurophysiol. 50:1415-1432.
23) Schwartz, E.L., Desimone, R., Albright, T., Gross, C.G. (1983). Proc. Natl. Acad. Sci. USA 80:5776-5778.
24) Seltzer, B., Pandya, D.N. (1978). Brain Res. 149:1-24.
25) Spitzer, H., Desimone, R., Moran, J. (1988). Science 240:338-340.
26) Ungerleider, LG. and Mishkin, M. (1982). Analysis of visual behavior, D.J. Ingle, M.A. Goodale, R.J.W. Mansfield Eds. MIT Press, Cambridge, Mass.
27) Zeki, S.M. (1980). Nature Lond. 284:412-418.

The emergent self: A phylogenetic and ontogenetic evolution of biological networks a psychiatric point of view

Ronald Goulet, Hopital Jean-Talon
1385 Jean-Talon est, H2E 1S6, Montréal Que

Summary: The problem of "self"could be summarized in a way, as the problem of synchronism and adaptive auto-organisation of appropriate behaviour in real time. This paper try to integrate new findings in neural networks theory and neurobiology, in a general neuro-bio-psychological theory of emotional disorders. Specifically it consider new developpement in artificial reinforcement and associative learning as genuine artificial emotional systems. Also, the recent developpements in the analysis of the style of computation and learning that takes place in the olfactive cortex and in the hippocampus are presented as an early stage in elaboration of a novel theory of psychopathology that could in return, be beneficial to the study of artificial autoorganised neural networks.

1- Working as a " biological neural net physician"

As a psychiatrist, I would rather consider that the point of view that we have on human psychopathology is the best top-down (from property to structure) consideration of the potential weaknesses of big neural networks conceived to discriminate important signs scatered in the environment in a real time base. We think that a majority of people in their attitudes doubt the existence of mental disorders : mainly because of the discomfort that caused the sight of largely damaged funtioning mind (especially for psychotic disorders). People have their own concept of these disorders in terms of reaction to the stress of life in an exagerated manner because of some weakneses in the moral strength of those afflicted by these problems. In fact there's not yet objective testing that can prove the existence of medical entities that corresponds to the psychopathological disorders. But the study of neural networks could solve part of the problem by considering such disorders as dynamic's defect of the working brain. This imply mostly the study of non linear dynamic of neural network processing in real time. Deepening our understanding of such complexity, it is even more intriguing to consider that the human mind works so well so frequently.

If we don't understand how brain function it is not surprising in fact, that we know almost nothing about the mechnisms implicated in his malfunction. Now, suppose, that a previously psychotic patient comes to you and say: "thanks for you doctor, I'm now cured: since this morning, I'm functioning like before"..during that time you are mummbling in your self: "How come tanks for me... how can I get him back to his "sickness" so I could continue to "cure him". Puzzling indeed : in fact, you can do almost nothing that can surely reverse the process. After all, one solution is to to tell your patient that even if he thinks he is cured according to your " theory" he is not , and he will surely relapse if he stops the treatment. In fact there is statistical evidence that most psychotic disorders may relapse. But you can use this fact to continue to "treat " "patients" wich are sane like most doctors and therapists prefer to do. Patient somehow knows that; and generally don't like to take medication that have sides effects or fastidious psychotherapy in prophylaxis matter. But we have no way to identify those who can benefit some of these prophylaxis other than relying on statistics, considering the type of disorder and the rate of relapse with or without prophylaxis.

The artificial neural networks theory could change a bit of these inaccuracies by analysing the dynamics of the brain. Particularily, I like to look at mental disorders as domain of attraction in a state space that represent the evolution of the mean mental state driven by some unknown control parameters. So, I became interested in the theoretical aspects of neural nets and neurobiology that could help us to identify those parameters and makes us understand how they act on the nervous system.

2-The "self" as a bounding problem:

We could start to what would seem to a neural engineer as a bounding problem (like the image invariance) in neural nets : the persistence in the sense of unicity of "self". Despite the fact that the

mutiples sub neural nets and layers had relative independant functions and especially despite the fact that in our life we are confronted with a vast variety of situations, we are reacting to them with an almost imprevisible although, auto-similar variety of ways, implying a variety of instantaneous mental sets. So the problem of self is a rather intricated bounding problem. We can understand also, that the question:"Who am I?" is a crucial one... but a question not to ask: as somebody always asking himself, or worse: others : "Who am I ?"; has surely something going wrong with him.
In fact, there is a paradoxical problem in autorepresentation trying to improve ones internal way of representation by feeding them back to themselves : that leads to the theorem of Godël. In brief, the implementation of a level of hidden units mapping all the representations of all other functional levels in a network, in order to correct them, is impossible due to the paradoxe of autoreference. In fact, autorepresentation in a redundant matter is indeterminate: like the sentence: "all the sentences in this text, even this one, are false". That lead Marvin Minsky to write :" ..perhaps,its because there are no person in our heads to make us do the things we want -nor even ones to make us to want-that we construct the myth that we're inside ourselves"(1). Now what is maintaining the myth? If the self is like a riding bicycle standing up just because it moves forward : what makes us believe that there's a unity and more a unicity in all our reactions?
In some way these questions reduce the bounding problem of the self to a more reachable question of stability in the dynamic of sequential adaptive organizer of behaviour. This question find an interesting solution in the concept of "auto-poïesis" or "gated coupling" (2). This means that at some deep level a neural system, when it is auto-organising himself in the environment in real time, must cease to just consider the patterns in the input as a recognition process. On the contrary, it has to become a self sustained dynamical system (called internal dynamic) reacting to all inputs coming from internal or external pathways : as if it could be a perturbaion of internal dynamic.In front of this perturbation the internal dynamic could remain structurally stable or bifurcate. By this way, we can conceive that a system without being aware of its "self", could perceive the boundary between reaction originating from the "self": withstanding a perturbation ; and reaction extraneous to it: when a perturbation lead to a bifurcation.
We can consider the orienting subsystem of the ART network of Steephen Grossberg, the Olfactive Cortex simulation of Walter Freeman : a specific cahotic density mapping, as first steps in the implementation of neural nets that react to an input and learn to react to it globally and in agreement with kind of internal determinant (drives). Theese studies illustrate that a kind of artificial emotional system (defined as a neural system acting as an internal dynamic) seems to be essential to implement autoorganised learning devices, in real time.

3-Hippocampus : a biological internal dynamic for inputs coming from the cortex

In the evolution of neurobiological networks, at first neural systems behave like internal dynamics without trying to discriminate in the environement other things than stimuli biologically significative. These stimuli elicit a response specific to the needs of the individual but largely correlated with fixed species patterns. These structures were preserved in the human brain as archaic internal dynamic, such as automomous nervous system, ascending reticular systems and hypothalamus. They are part of an intricated system of smalls nuclei coding for different aspect of the vital needs for the survival of the individual and the species and modulating in cyclic matter the relative inportance of each of them by complexely interacting systems of neuropeptides and hormones circulating in the blood as well as released at the synapse clift . They control for example the stages of conciousness and sleep the relative food intake, the activity level ect.This system is coupled to the hippocampus by neurohormones but mostly by three neurotransmitter : nor-adrenalin ,serotonine and acetylcholine , part of an ascending system driving attentional mechanisms in significative situation (pain, interest and sorrow,fear or rage).
As far as the hippocampus is concerned: it seems that it is designed to associate the signals or cues that are picked up by the highly discriminating cortex(coming from entorhinal cortex) to stimuli that are mapped as reaction in the archaïc dynamical systems from above (from afferent fornix). The hippocampus is a recurrent network reassessing new inputs with those who preceded them. It feedforward outputs to the orbito frontal cingular and temporal cortex relaying from the efferent fornix by two nuclei : the maillary body and the anterior thalamus. The output seems to be a kind of reaction : an error signal when it's needed destinated to parts of the cortex specialised in planning (cingular and frontal : sequencing mechanisms) and in apprehention directed to new sensory recognitions (temporal). From these parts of cortex new solution could be initiated or if impossible a

relay from cortex : the amygdala feedforward down to the archaïc system initiating urgency mechanism (figth-flithlt-submission ect.).
It seems that compressed and multimodal sensorial information as input could be there classified in terms of outcoming reaction and associated results. The CA1and SUB regions of the hippocampus acted as a memorising comparator of these situations and their real versus predicted outcome. So, if all works like predicted the system would not be perturbated but if something strange happens the system could detect it and try to classify it. It could thus be: a cortical internal dynamic.
So, in this view, the hippocampus is not as much as a center of the self, but not also as litlle as a startling reflex center. It is meerely a "oh-oh, change-o" system that react to imprevisible outcome of situations by trying to reclassify it by means of past experiences, and by learning the new outcomes that follow a remastering of the situation by the cortex. The physiology of this system seems to have been disigned for it to assume a kind of curiosityfunction in the brain.

4- Dynamic of the septo- hippocampal system: a speculative model of emotional function

The dynamical physiology of hippocampus contain much information susceptible for becoming cues in the unfolding of its computational mode:

1- One can describe the septum like a beat generator driving the neurons of the hippocampus like a conductor drive his musicians in an orchestra : there are tree types of frequencies: 1- theta low rhythms (under 6-7 hertz in rodents) corresponding to non ambulatory activity oriented toward drive satisfaction 2- high rhythms (above 8 hertz in rodents) whose frequency seems to be somewhat related to the velocity to be atteined for reaching a goal in ambulatory behaviour and finally 3- an intermediate rhythm associated by Jeffery Green(3) to anxiety and behavioural inhibition in reaction to a novel situation. These types of rhythms could be seen as a variation of a paramete of control transforming the whole dynamic of the network, like a phase variation in thermodynamic systems out of equilibrium.

2-There are at this time two types of learning known to occur in the synapses of hippocampus (mostly CA1 an SUB region associated by J. Gray to the comparator function between actual and predicted outcome):
a) The LTP (long term potentiation) associated with reinforcement of NMDA type of glutamate receptor when in a situation of exitation of the post synaptic neuron a receptor is activated two times in a row according to a delay corresponding to the phase of theta rhythm (200 msec)- (G. Lynch)
b) the associative learning when a first synapse can activate another one, in the same neuron when the activation of the first correponding to a UCS (unconditioned stimulus) is followed by the activation of the second CS (conditioned stimulus), after an interval corresponding to a fixed ISI (interstimulus interval) (D. Avlon)

3- It seems also that the archaïcal dynamical systems (ascending tegmental systems) bring their action firstly by means of Influence in the thêta driving activity (reticular cholinergic activating system) and also by generating in (F.D) fascius dentata (the opening gate: of the hippocampus receiving inputs from entorhinal cortex via perforant pathway): a lowering of the treshold in these neurons to repond to the intermediate theta rhythm : thus providing a kind of gain control in recruiting the function of discriminating new elements in the input. This latter function seams to be devoted to the nor-adrenergic and the serotoninergic ascending systems relaying the hilus of F.D, (J. Gray, J.O'Keefe). These systems are known to be habituating systems (adaptive but subject to exhaustion) and largely implicated in depressive and manic disorders.
Now, if we postulate that the kind of convergence of this network as a cahotic nature like the one that was recently proposed in simulating biological olfactif cortex (W. Freeman) : we can imagine this system classifying strange situations by means of pre-empt experiences in a self-similar matter (families of cahotic attractors usually are self-similar): meaning that the global reaction to a globally analysed puzzling situation could be similar and much more detremined, by a certain degree of learned or physiologically predetermined self confidence or adaptability to novelty than corresponding to the type of novelty in the stimulus.
In the phylogenetic evolution : it seems that hippocampus had been involved in the reaction of animals to his environment according to an increement of the discriminative abilities that enables them to consider much more cues in an infinite variety of strategies for ensuing their goals. It is thus not surprising that

it appears to the experimentator (J. O'keefe) like a spatiotemporal mapping for territoriality : say: an abstract and global analysis for each animal of a plastic and ambulatory environment with whom they became familiar considering the kind of reward and danger it could provide.

5- Ontogenesis of psychodynamical structure: from nature to nurture:

What happens to territoriality in primate and more specifically in Humans : we postulate that it became a much more abstract analysis of the interpersonal environment. The position in the territory become: the position of the "self" in front of others in terms of submission, leadership, friendship, love etc. So, condense information coming in input concern mostly the mental state of others. And the reactions bounded to it defined the self as opposed to the behaviour of others predicted in the plannings coming from frontal cortex. We can thus imagine that reactions of human beings, to all novelty even when it have nothing to do with interrelation, could be caracterized like on a template on primordial relashinships pre-empted in infancy.
The internal dynamic of hippocampus doe's not contain the personnality but by means of a self similar structure could lead to perseveration in choices that maintain it. The hippocampus as proposed , here, is implicated in the integration of novel knowledge to the previous: the stage (REM) sleep associated with dreams and activated when theta rhythm start to burst in the hippocampus could have as principal function to integrate in the cortex the newly learned informations to all the others by "smoothing away" the incongruency with a cost of forgetting something.
We are now speaking of self sturcturing dynamical system by means of better adaptation and implying in the process loss of information: this is the growing quantity of informational complexity in the ontogenesis; this is also the freudian notion of unconcious.
We speaked above of synchronisation with regard to the expected delay of optimal reaction : the hippocampus does not seem to be able to assure a stability in this regard .In humans the frontal cortex had greatly evolved with regard to what it was in ancestral primates. This cortex helped by the dopamine driven striatum seems to be the synchronizing device of the brain in an adaptive and modulated way. The hippocampus by influencing it, seems just able to activate or slow down the genral pace of the whole system.

5- Psychopathology : as weakness of self integrated adaptive neural net

It is interesting to consider that the two principal types of psychotic disorders concern the two systems considered here as central for self-cohesion.
If we first consider the postulated gain control in behavioural inhibition of the hippocampus it is plausible that this system is implicated when exhausted in a kind of disorder implying a behavioural inhibition,a loss of energy, loss of adaptive capacity and concentration with continuous anxiety and worries about the future like depression or in contrary, a loss of normal inhibition and anxiety in dangerous situation with hyperactivity like in mania. These disorders appering to be a disorganization of the internal dynamic system of the brain mostly the ascending nor adrenergic and serotoninergic systems. And we can see that the pharmacotherapy that treat these disorder act on serotoninergic or noradrenergic central systems.
The sequencing system being much more caracteristic of the human brain with his huge frontal lobes may be more implicated in disoder of thinking particularily sequencing of thinking witch caracterise the schyzophrenic disorders. Thus it is not surprising that pharmacotherapy most efficient in schyzophrenia block the dopaminergic system that could stimulate the striatum (modulating the sequencing in frontal lobes).

6- Conclusion

We conclude that considering the brain as a dynamical neural network could surely be of benefit in the near future for the development of conceptual models in psychiatry. Considering the brain as computing cahotic attractors could also be an interesting field: for example the mesurement of fractal dimmension of EEG could be interesting in the assesment of mental disorders and in the follow up of treatment. It offers also a contribution in the field of artificial thinking trying to understand why are we feeling the external world in ourselves like we are.

References

1- Minsky Marvin - 1985-The society of mind, Simon Shuster edit
2- Varela Franceso -1983- L'autoorganisation de l'apparence au mécanisme: Colloque de Cerisy : l'auto-oragnisation (Paul Dumouchel et Jean Pierre Dupuy) SEUIL edit

3- Gray Jeffery-1983-The neuropsychology of anxiety an enquiery into functions of the septo-hippocampal system OXFORD SCIENCE PUB.
4- O'Keefe John, Nadel L. -1978- The hippocampus as a cognitive map Clarendon Press OXFORD UNIVERSITY PRESS
5- Freeman W Yong Y Burke B Central pattern generating and recognising in olfactory bulb 1988 Neural netwoprk vol 1- no-4

ON THE BEHAVIOR AND SIGNIFICANCE OF RANDOM NEURONAL NETWORKS

Guenter W. Gross*, Jacek. M. Kowalski, and David Golden
Department of Biological Sciences*, Department of Physics, and Center for Network Neuroscience,
University of North Texas, Denton TX 76203.

Before the recent theoretical and experimental "network revolution", neuronal networks were considered mostly hard wired circuitry that operated on incoming sensory information for transformation into output spike patterns appropriate for the function of a particular network and organism. Function was based primarily on morphology (i.e. circuit structure) and fault tolerance was a matter of redundant interconnectivity (for reviews see Getting, 1988, 1989). This view has changed dramatically in the last 5 years. Networks giving rise to simple invertebrate behavior were found remarkably complicated (Krasne and Wine, 1984; Selverston, 1985; Getting, 1988) which now includes the switching of neurons from one network to another (Hooper and Moulins, 1989) Recently, Getting (1989) concluded: "Knowledge of connectivity alone is not sufficient to account for the operation and capabilities of neural networks". This complexity is compounded further in vertebrates where even the most basic functions are governed by cell ensembles with highly redundant interconnectivity displaying formidable morphological and functional plasticity.

Until recently, experimental techniques were not available to capture the simultaneity of electrical events in small networks. Investigations of network activity patterns with a large number of electrodes to obtain a representative view of the internal network dynamics could not be conducted. Introducing many electrodes into a small volume of tissue *in vivo* or into slices *in vitro* will alter or destroy the circuitry being investigated. In cell culture, where neuronal circuits can be grown as monolayers, the mechanical problem of holding and positioning many conventional microelectrodes was a serious barrier to multisite recording. As a consequence, there exists in this area such a paucity of basic biological data that theoretical efforts cannot be adequately tested or guided.

The development of photoetched, thin film multielectrode surfaces has provided a convenient method for the simultaneous monitoring of many neurons in small monolayer networks in culture (Gross, 1979; Gross and Lucas 1982, Gross et al, 1986; Droge et al, 1986). The method yields networks that grow on the electrodes, thus stabilizing the cell-electrode coupling and eliminating destruction of cells by invading electrodes. It allows easy manipulation of the physical and chemical environments as well as observation in the living state. However, the dissociated culture approach destroys the original circuitry and the subsequent random cell seeding generates randomized networks. Although there is some evidence of preferential synapse formation in culture (Camardo et al.,1983) it is most likely that such networks contain large and perhaps predominant random components.

Our **experimental strategy** for the analysis of network dynamics involves long-term monitoring of 64 electrodes that capture a large proportion of the signal traffic within 1 to 2 mm diameter monolayer networks consisting of 100 to 500 neurons. Spontaneous, evoked, and pharmacologically altered activity is being collected from monolayer networks derived from mouse spinal cord, olfactory bulb, cortex, or cerebellum. Analyses are conducted with light microscopic, electrophysiological, pharmacological, and laser surgical methods, and the subsequent application of histological, histochemical, and electron microscope techniques. Network alteration via laser cell surgery can be accomplished during recording to test for system homogeneity, existence of percolation thresholds, critical mass phenomena, and fault tolerance. Specific cell and neurite elimination as well as network alterations can be achieved with precision (min. focus diam.: 1um; positioning accuracy: +/- 0.5 um).

Our **theoretical strategy** involves application and modification of "coupled planar rotator models" to describe the complex spiking and bursting phenomena observed in culture. We have built a computer model of network dynamics that uses nonlinearly coupled, nonlinear planar rotators of the type $\varphi_i = \omega_i - \beta_i \sin \varphi_i + \Sigma J_{ij} \sin (\varphi_i - \varphi_j)$ solved by a Runge-Kutta approximation to determine the temporal evolution of each network element. We have begun with stability investigations of small systems with several strongly interacting neuronal "rotators". From our preliminary efforts (Kowalski et al, 1988) and from other pertinent theoretical work (Kopell and Ermentrout, 1986; Sagakuchi and Kuramoto, 1986) we judge the coupled rotator formalism sufficiently versatile to simulate random networks in culture. We are using three different time scales to represent the observed dynamics: two are necessary to describe fast relaxational excitations in single neurons. A third, slower, time scale emerges in a process of neuronal "tuning" to different dynamic regimes (with different excitability thresholds, refractory characteristics, and instantaneous spiking frequencies). This slow variable, modulating neuronal activity, may be the most significant for the dynamics of interneuronal communication. The resulting network models then become equivalent to dissipative systems of bistable, nonlinearly coupled planar rotators reflecting the cyclic character of the slow variable. Dynamics of such systems are rich and not fully understood.

RANDOM NETWORK DYNAMICS in CULTURE

A characterization of network dynamics requires systematic analyses of spatio-temporal patterns. *This is possible in monolayer cultures.* Spatial patterns represent variations in the location of activity within the 1 x 1 mm, 64 electrode recording matrix. Some regions dominate and activity spreads to adjacent regions depending on burst intensity and other as yet unidentified factors. Temporal patterns range from random, to chaotic, to periodic, are generally very volatile, but may be "locked in" pharmacologically by blocking inhibitory circuits and possibly by interfering with the NMDA receptor complex. They may be totally or partially coupled (synchronized or entrained - the latter with fixed or variable phase differences between bursts), necessitating a derivation of coupling functions. Two levels of information traffic must be considered: spike patterns and burst patterns. The former is presently deemphasized because bursting usually predominates. The latter is described in terms of burst frequency, duration, type (i.e. spike patterns within bursts) and burst type sequence.

The following specific questions are now being asked to determine the "dynamic range" of random networks. (a) Is regional dominance permanent or volatile and what are the conditions that enhance of alter such dominance? (b) What factors determine synchronization and entrainment? (c) What is the range of patterns that can be produced by the ensemble, spontaneously and under pharmacological perturbations, and what are the most likely patterns? (d) What are the causes of spontaneous pattern fluctuations, and what manipulations are required to "lock in" patterns? (e) How does internal (i.e. neuronal) noise and external noise (via electrical stimulation) influence the network behavior and can it trigger changes in pattern type, or kindle quiescent networks? (f) What are the typical network responses to controlled chemical and patterned electrical stimuli? (g) Can patterns be stored and what are the biochemical and biophysical requirements for storage?

Summary of Important Observations from Mouse Spinal Monolayer Networks

(1) All cells in a 1mm diameter miniculture are directly or indirectly connected. Preliminary data from burst phase shifts (between different electrodes) suggest that the number of synapses between any two cells ranges from 1 to 6 with an estimated mean at 3.
(2) All neurons in the culture are capable of entrained bursting. Entrainment is a function of burst intensity; whereas weak bursts or random spiking show islands of independent activity, strong bursts tend to recruit all cells in the network into a common pattern.
(3) The most common and basic behavior of these networks is entrained bursting. Phase shifts vary from a constant minimum of several msec (conduction and synaptic delays) to highly variable tens and hundreds of msec (processing dynamics).
(4) There are no endogenous bursters in these cultures since all activity ceases in 12 mM magnesium chloride. Bursting is conditional, network dependendent, and requires a minimum background activity for kindling of network activity.
(5) The separate or combined blocking of the inhibitory GABA and glycine receptors (with bicuculline and strychnine, respectively) generates, after a short paroxysmal activity period, remarkably

regular oscillatory burst patterns with 100% coupling between electrodes. Strychnine causes burst stretching and enhances pattern regularity by reducing variations in burst frequency and burst duration. Bicuculline is more effective for improving pattern regularity but does not produce burst stretching (Hightower, 1988).
(6) The addition of either GABA or glycine may decouple electrodes and stops all spontaneous bursting at approximately 30 uM. Some random spiking, however, is maintained.
(7) The GABA antagonists picrotoxin and penicillin do not enhance pattern regularity although they are equally effective for producing intense paroxysmal firing (Hightower, 1988).
(8) NMDA responses, including typical reactions to agonists and antagonists, have been observed in every culture implying that LTP storage mechanisms are intact. Not all electrodes respond in the same manner and uncoupling is often seen (Gordon et al, 1989).
(9) Excitatory transmitters (aspartate, gluatamate, acetylcholine) reduce pattern regularity and may induce uncoupling.
(10) Electrical stimulation is ineffective unless a number of electrodes are pulsed simultaneously possibly reflecting network fault tolerance via resistance to external noise.

The Significance of Bursting

Bursting is a predominant feature of spinal cord and olfactory bulb monolayer cultures and may play an equally important role in cerebellar and cortical cultures for which we have only pilot data. The clustering of action potentials into identifiable "bursts" is a common phenomenon in both invertebrates and vertebrates (Provine, 1972; Miller and Selverston, 1982; Cohen et al., 1988; Grillner et al., 1988). Most important is the recent recognition that changes in synaptic efficacy require spike clusters: facilitation, augmentation, and potentiation are most easily obtained with high frequency inputs (Zucker, 1988). The mechanisms underlying long-term potentiation (LTP) clearly depend on large local depolarizations generated by spike clusters (Brown et al., 1988). In the rat visual cortex, LTP could be induced only in cells that showed bursting. Regular spiking cells resisted LTP generation until low doses of the GABA antagonist bicuculline were applied to change them into bursters (Artola and Singer, 1987). As a result of such diverse observations, bursting must be considered a vital network phenomeon that greatly influences information processing and storage.

CONCLUSIONS

The presence of structural and dynamic fluctuations in these monolayer cultures does not prohibit statistical descriptions that concentrate on highly probable, gross behavioral features of a large family of macroscopically identical cultures (with identical origin, similar neuronal densities, environmental parameters, etc). The complex of these features can be called the network "macrostate", in analogy with "reduced" descriptions in statistical physics. "Pathological", large deviations from the expected behavior are also of interest because they relate to the stability limits of the investigated system. The main problem is the collection of a sufficiently large experimental data base to attempt a "thermodynamic" network description in terms of a few crucial variables. We believe that the modified rotator model can serve as a realistic approximation of the dynamic behavior of networks consisting of several hundred neurons. The partial entrainment phenomenon (synchronized or almost synchronized) to a common frequency is generic for some classes of these networks. Additionally, we have evidence of a "coarse-grained" synchronization also in transient states, before the system reaches a steady state. Chaotic behavior is not excluded in these systems with random interconnections ("random interaction multi diagraphs") subjected to problems of reliability and percolation resulting from both spontaneous and experimentally-induced "site" (neuron) and "bond" (synapse) failure.

Predictions and Extrapolations

(1) The basic signal processing dynamics as well as the biochemistry of information storage is already present in random networks. The culture is constantly and spontaneously "learning" which occurs randomly or chaotically in the absence of sensory feedback.
(2) The random network is probably capable of performing most CNS functions but with low

efficiency and less reliability. Specific circuits have evolved to increase the efficiency of realtime processing and the amount of incoming data that can be handled.

(3) Network states are determined by a complex balance of excitatory and inhibitory influences with rapid and extensive up and down regulation of receptors and channels.

(4) Random networks enter a rapid, highly regular oscillatory phase as total disinhibition is approached. This oscillatory behavior is a basic network characteristic.

(5) All neurons in these networks are able to store information. The random network in culture may become one of the most powerful experimental systems for detailed investigations of biochemical and biophysical mechanisms of cellular and network information (i.e. pattern) storage.

So far the random biological network has not aroused much interest because it was not considered representative of the intact CNS tissue. *However, it is now apparent that "order" in the nervous system must be seen statistically and that structure and function is linked less deterministically and much more probabilistically than had heretofore been assumed.* This seems to be especially true of small networks where synaptic weights and activity dynamics exert as much influence over network function as specific network structure (Getting, 1988). Therefore, investigations of random network characteristics, including most probable pattern generation, capability limits, reliability, and fault tolerance are important to studies of network behavior and to our understanding of why specific, highly ordered networks evolved. It is clear that network dynamics can be studied quantitatively with monolayer cultures and it should be possible to achieve a comprehensive theoretical explanation of the internal dynamics and self-organization of random neuronal networks.

REFERENCES

Artola, A. and Singer, W. (1987) Long-term potentiation and NMDA receptors in rat visual cortex. Nature 330:641-652. **Brown, T.H., Chapman, P.F., Kairiss, E.W., and Keenan, C.L. (1988)** Long-term synaptic potentiation. Science 242: 724-728. **Camardo,J., Proshansky,E., and Schacter,S. (1983)** Identified Aplysia neurons form specific chemical synapses in culture. J. Neurosci. 3: 2621-2629. **Cohen, A.H., Rossignol, S., and Grillner,S. (1988)** Neural Control of Rhythmic Movement in Vertebrates. John Wiley & Sons, N.Y. **Droge, M.H., Gross, G.W., Hightower, M.H., and Czisny, L.E. (1986)** Multielectrode analysis of coordinated, multisite, rhythmic bursting in cultured CNS monolayer networks. J. Neurosci. 6: 1583-1592. **Getting, P.A. (1988)** Comparative analysis of invertebrate central pattern generators. In: Neural Control of Rhythmic Movement in Vertebrates (A.H. Cohen, S. Rossignol, S. Grillner, eds.) pp 101-128, John Wiley & Sons, N.Y. **Getting, P.A. (1989)** Emerging principles governing the operation of neural networks. Ann. Rev. Neurosci. 12: 185-204. **Gordon, M., Fracek, S.P., and Gross, G.W.(1989)** NMDA responses of small neuronal networks in culture. Soc. Neurosci. Abst.15, in press. **Grillner, S. (1985)** Neurobiological bases of rhythmic motor acts in vertebrates. Science 288: 143-149. **Gross, G.W. (1979)** Simultaneous single unit recording in vitro with a photoetched, laser deinsulated gold, multimicroelectrode surface. IEEE Trans. Biomed. Eng. BME 26: 273-279. **Gross, G.W. and Lucas, J.H. (1982)** Long-term monitoring of spontaneous single unit activity from neuronal monolayer networks cultured on photoetched multielectrode surfaces. J. Electrophys. Tech. 9: 55-69. **Gross, G.W., Wen, W., and Lin, J. (1985)** Transparent indium-tin oxide patterns for extracellular multisite recording in neuronal cultures. J. Neurosci. Meth. 15:243-252. **Hightower, M. H. (1988)** PhD Dissertation, University of North Texas. **Hooper, S.L. and Moulins, M. (1989)** Switching of a neuron from one network to another by sensory-induced changes in membrane properties. Science 244: 187-189. **Koppel, N. and Ermentrout, G.B. (1986)** Symmetry and phase locking in chains of weakly coupled oscillators. Commun. on Pure and Applied Math.39: 623-660. **Kowalski, J.M., Ansari, A., Prueitt, P.S., Dawes, R.L., and Gross, G.W. (1988)** On synchronization and phase locking in strongly coupled systems of planar rotators. Complex Systems 2: 441-462. **Krasne, F.B. and Wine, J.J (1984)** The production of crayfish tail flip escape responses. In: Neural Mechanisms of Startle Behavior (R.C. Eaton, ed) pp179-211, Plenum, N.Y. **Provine, R.R. (1972)** Ontogeny of bioelectric activity in the spinal cord of the chick embryo and its behavioral implications. Brain Res. 41: 365-378. **Sakaguchi, H. and Kuaramoto. Y.(1986)** A soluble active rotator model showing phase transitions via mutual entrainment. Prg. Theoret. Phys. 76: 576-581. **Selverston, A.I. (1985)** Model Neural Networks and Behavior. Plenum Press, N.Y. **Zucker, R.S. (1988)** Frequency dependent changes in excitatory synaptic efficacy. In: Mechnisms of Epileptogenesis (M.A. Dichter, ed.), Plenum, N.Y., pp 153-167.

A cognitive triangular relationship

Arno J. Klaassen
Delft university of technology, department of computer architecture
P.O. box 5031, 2600 GA Delft, the Netherlands
phone: +31-15-786177; telefax: +31-15-783622; email: arno@duteca.tudelft.nl

Abstract

A design method for cognitive systems is presented in relation to the Neuronal Group Selection theory and the Modularity of Mind philosophy. It is indicated that these three supply each other and that their combination makes it possible to design cognitive systems no longer suffering from the brittleness problem.

1 A design method for a 2-layer cognitive system

Cognitive systems, i.e. systems capable of acquiring knowledge from their environment, have not been very successful so far. One reason is the fact that they are implemented as symbolic processes, which lack flexibility because such processes can solve properly only well-defined problems which do not involve ambiguous data [10].

Another reason is that even the most simple cognitive system is complex. Since they are implemented as symbolic processes one gets the drawbacks of symbolic processes: they must be programmed by hand, which is hard to do for complex systems, and are evaluated serially, which implies a low performance for complex systems. This fact of explicitly having to program a system forms the basis of the aforesaid brittleness problem.

A while ago neural networks seemed to be a means to solve these problems. The idea was that a large number of small processing elements doing numerical computation eventually would perform better than a single powerful processor doing symbolic computation. However, a cognitive system which performs perfectly well, the neural network of the human brain, consists of about 10^{11} neurons [8]. But on the other hand neural networks nowadays consist of just a couple of neurons. For instance a relatively complex neural network such as a phoneme recognition network consists of just 384 neurons [11].

It remains doubtful whether such a neural network can be scaled up to a size some orders of magnitude larger and at the same time still can be handled properly by ordinary human beings; I mean, learning and understanding a large network is far from trivial.

In [5,6] a method is indicated to design a system that more appeals to the spirit behind massive parallelism but nevertheless remains workable and understandable. The heart of the matter is to split the system into two layers of which neural networks form the lowest layer (see Figure 1). The higher layer is formed by a symbolic system.

When a theory for cognitive systems fits in this framework it is to be expected that

1. to a certain extent the neural networks involved, can solve the problems mentioned in the beginning of this section,

2. the total number of processing elements and interconnections between them is of such an order that really can be spoken of a massive-parallel system.

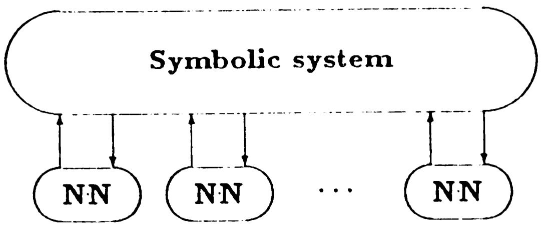


Figure 1: *Architecture of a 2-layer cognitive system*

A very interesting theory in this respect is the neuronal group selection (NGS) theory of Edelman [1,9]. According to his theory two kinds of selection events play critical roles in shaping the development of a neural system. During the formation of the system genetic selection among competing neural networks and their processes determine the shape and modal connectivity of a (sub-)network. Later, during experience and interactions with the environment etc., connectionist selection, accomplished by synaptic modifications, determines the final behavior of the system. In the next section I will shortly indicate the essential part of NGS.

2 Neuronal Group Selection

The NGS theory is a purely biological theory from origin [1,9]. Its main intention is to give a possible explanation for the fact that a brain is capable of development, perception, memory and learning. According to the theory a brain is dynamically organized into populations of individually variant networks, the structure and function of which are selected during development and interaction with the environment by means of selectionist's and connectionist's principles.

The units of selection are neural networks which consist of about 50-10,000 neurons, neuronal groups. These act as functional units. Roughly the theory makes the following three assumptions: first genetic processes lead to the development of primary repertoires of structurally variant neural networks; second synaptic modifications lead to the development of secondary repertoires more fine-tuned for future use (survival of the fittest); and third the existence of feedback at all levels. Or in Edelman's own words: "During experience and after the receipt of input signals that are filtered and abstracted by sensory transducers, by feature extraction networks and by feature correlators in mapped reentrant (feedback, A.K.) sensorimotor systems, certain neuronal groups are selected over others in a competitive fashion. [...] This process, in which groups that are more frequently stimulated are more likely to be selected again, leads to the formation of a secondary repertoire of selected neuronal groups which is dynamically maintained by synaptic alterations." [1](pp.45-46).

This theory fits in the 2-layer framework since naturally the neural networks form the first layer, whereas the selectionist processes and overall feedback could be under control of the symbolic upper layer.

Readers familiar with the modularity of mind philosophy may have noticed that the way Edelman talks about sensory transducers and neural networks indicates that NGS and the

2-layer framework fit in the philosophy, which I will describe in the next section. This is salient because it means that a design method for, a biological theory about, and a philosophy of cognitive systems meet here.

3 The modularity of mind and neural networks

In 1983 Fodor wrote an assay called "The modularity of mind" [2]. Maybe at first sight Fodor is not the man to embrace for a connectionist researcher since he and Pylyshyn wrote a year ago that we should give up the idea that (neural) networks offer a reasonable basis for modeling cognitive processes in general"[3](p.68), although they present that view a little more cautious than I. Nevertheless I think that the philosophy that Fodor has described in his essay is very well suitable in the light of 2-layer cognitive systems and NGS. To explain I first will give a short outline of the philosophy of Fodor.

His theory is a new version of faculty psychology. It alleges that there have to be postulated essentially different kinds of (psychological) faculties in case to be able to explain the facts of mental life. Fodor proposes a mixed horizontal and vertical faculty psychology. His functional architecture consists of input systems and a central system. The input systems with sensory transducers are vertical faculties, although Fodor calls them 'modular'. That is to say that they operate independently of each other and independently of the central system. This central system exhibits horizontal faculties.

Input systems have to have a number of properties. I will mention the most important ones by which it is obvious that neural networks possess such properties as well. Input systems are associated with fixed neural architecture, operate fast and are domain specific. This means that each input system just has a limited view of and an equally limited knowledge about the world. Further input systems are in two ways informationally encapsulated. This implies that both an input system cannot dispose at all the information of the rest of the system and that the central system cannot obtain all information gathered in an input system. This is the property which makes input systems vertical faculties and in fact it is the most important one because it provides a rationale for viewing a cognitive system as consisting of two layers. The point is that according to Fodor input systems should act as reflexes. His a priori argument for this is that otherwise they would become too slow: in perceiving a tiger on your retina you do not want to be bothered by information whether your grandmother likes tigers or not, whether that demonstration against tiger fur were tomorrow or the day after tomorrow, etc.. This can be reached by just not letting pass such information to the input systems. So their is at least limited flow of information from the central system to the input system . Fodor is not very clear about whether their may be no feedback at all between central systems and input systems. I think their must be at least some feedback, if only for the reason that you cannot learn the input systems if their is no possibility to tell them whether their output was more or less correct. Further it is a common notion in psychology nowadays that expectations partly determine what we perceive.

On the contrary the processes in the central system are not informationally encapsulated, not domain specific, are slow, and they reason with the information of the input systems.

4 Prospects and Conclusion

Summed up we have the following situation:

- We have got a philosophy claiming that a cognitive system should consist of input systems and above it a central system.
- There is a brain theory asserting that a cognitive system should consist of repertoires of neural networks and above them forming and maintaining rules.
- And there is a design method claiming that a cognitive system should consists of neural networks and above them a symbolic upper layer.

From the perspective of artificial intelligence this gives the opportunity to design computer architectures for cognitive systems that may be expected to suffer less from the brittleness problem because they have as towers of strength two theories that try to declare the cognitive functioning of a brain . And a brain doesn't suffer from the brittleness problem.

For easy prototyping such an architecture can be implemented e.g. on a Sun by means of Kyoto Common Lisp and the Rochester Connectionist Simulator linked together. It is also possible to implement them on special purpose hardware as currently developed at our university [4,7] which can be controlled by a host computer. It is even possible to make a system consisting of VLSI neural networks and controller chips.

References

[1] Gerald M. Edelman. *Neural Darwinism: the theory of neuronal group selection.* Basic Books, Ney York, 1988.

[2] Jerry A. Fodor. *The modularity of mind.* MIT Press, Cambridge Massachusetts, 1983.

[3] Jerry A. Fodor and Zenon W. Pylyshyn. Connectionism and cognitive architecture: a critical analysis. *Cognition*, 28(1-2):pp.3–71, March 1988.

[4] Jaap Hoekstra. *On the use of multiprocessor systems for neural network simulations.* submitted for publication, 1989. (available from the author)

[5] Arno J. Klaassen. *On massive-parallel cognitive architectures, neural networks, and the modularity of mind.* submitted for publication, 1989. (available from the author)

[6] Arno J. Klaassen and Jaap Hoekstra. Massive-parallel cognitive architectures based on neural networks. In Terry Huntsberger, editor, *Fifth annual computer science symposium*, pages pp.13–27, University of South Carolina, Columbia, SC 29208, April 7-8 1989.

[7] Arno J. Klaassen and Mark Korsloot and J. M. Mulder. The suitability of transputer networks for various classes of algorithms. In M. Reeve and S. Ericsson Zenith, editor, *Parallel processing and artificial intelligence*, John Wiley & Sons Limited, Chichester, England, 1989.

[8] Bryan Kolb and Ian Q. Whishaw. *Fundamentals of human neuropsychology.* W.H. Freeman and company, New York, 2nd edition, 1980.

[9] George N. Reeke, Jr. and Gerald M. Edelman. Real brains and artificial intelligence. *Daedalus journal of the american acadamy of arts and sciences*, 117(1):pp.143–175, Winter 1988.

[10] Luc Steels. *Artificial intelligence and complex dynamics.* AI memo 88-2, VU, Brussels, 1988.

[11] A. Waibel et al. *Phoneme recognition using time-delay neural networks.* Technical Report TR-I-0006, ATR interpreting telephony research laboratories, October 1987.

SUB-NEURAL FACTORS OF NEURAL NETWORKS

Djuro Koruga
Molecular Machines Research Center
Faculty of Machine Engineering
University of Belgrade, 27. Marta 80
11000 Belgrade, Yugoslavia

1. Introduction

During the past thirty years interest in neural networks has generated a number of different models. The main point in our research is sub-neural activities on a molecular level. We have made a link between sub-neural networks based on cytoskeleton and today's models of neural networks. This approach shows that the sub-neural network based on cytoskeleton plays an important role in both control mechanisms of nerve cell geometry and the neural network.

2. Cytoskeleton

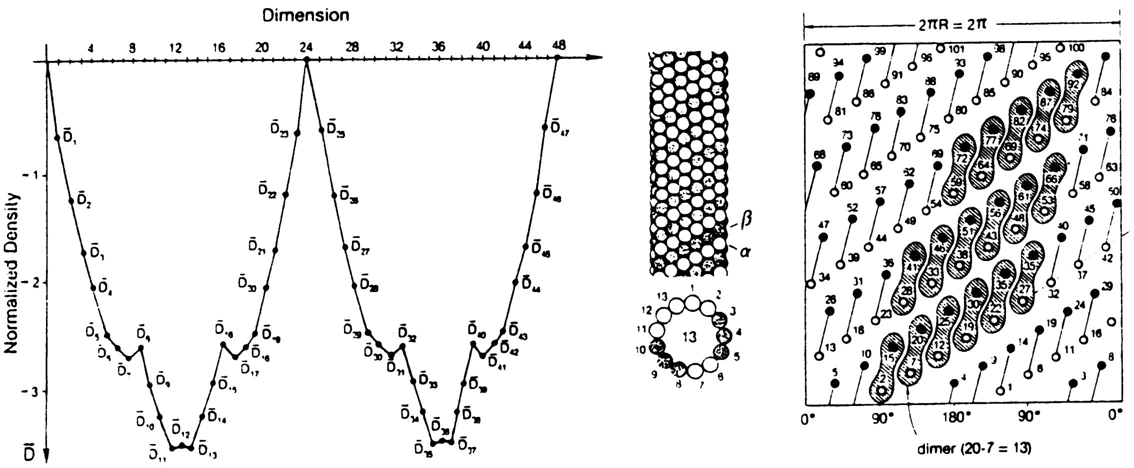
Cytoskeletal lattices include protein polymer microtubules (**MT**), actin, intermediate filaments and more than fifteen other proteins. The major neural architectural elements of **MTs** are cylindrical polymers, which also comprise of cilia, mitotic spindles and other organelles. **MTs** are intimately involved in dynamic biological activities, but mechanisms of "real time" regulation and control of **MTs** or other cytoskeletal filaments are yet completely unknown.

Of all the biological structures that participate in bioinformation molecular processes, only a small number work on such a principle that can be applied to the computer sciences. One of these rare biological structures is a microtubule, a self-organized organelle usually consisting of 13 subunits. These cytological structures create a network of protofilaments in the cell (neuron), similar to the way in which neurons create a network in the brain.

MTs are organelles present in nearly all eucariotic cells, composed of equimolar amounts of the two globular (~ 50,000 Dalton) subunits, α and β tubulin, each having a similar amino acid composition and a similar globular (sphere) shape. The subunits of tubulin molecules are assembled into long tubular structures with an average exterior diameter of ~ 24nm, capable of changes in length by self-assembly or disassembly of their subunits; sensitive to cold, high hydrostatic pressure and several specific chemicals, such as colchicine, vinblastine etc.. The stable form of tubulin, both in vivo and in vitro is a dimer, and the functional unit in **MT** assembly is the α-β heterodimer. **MTs** build, with other proteins, complex assemblies, like the mitotic spindle, centrioles, cilia and flagela, axonemes and neurotubules, and they intervene in the cell shape, the motillity and mitosis, and some part of cell functional information processing.

Microtubules participate in intracellular transport, addressing, growth form and many other dynamic activities [1]. Some experimental results link tubulin and microtubules to bioinformation processes such as memory and learning [2,3]. They are remarkable because they play two distinct and separable roles: as structural materials (cytoskeleton) and as dynamic machines that operate on a molecular level. They are carbon based 3D-intelligent machines which function on the nanometer size scale. Fröhlich has shown that nonlinear dipole excitations among molecular subunits of membranes and proteins of high dielectric strength, will result in coherent oscillations ($10^{-10}$to 10^{-11} sec fluctuations), lower frequency metastable states, coherent polarization waves and long-range order. Coherent oscillations of MTs and other cytoskeletal lattice subunits may derive from intrinsic oscillations dependent on a high dipole moment and dielectric strength, across the MT wall, or coupling with oscillations in membranes [4].

Symmetry properties of hexagonal packing of protein monomers have been used to explain the form patterns of viruses, flagella and MTs [5]. Since hexagonal packing and face-centered-cubic packing have equal density, we used the $Oh(\bar{6}/4)$ symmetry group to explain MT organization and its information system. Since packing is constructed from codes for digital transmission of information, we believe that MTs possess a code system that can provide memory and intracellular dynamic activities [6].



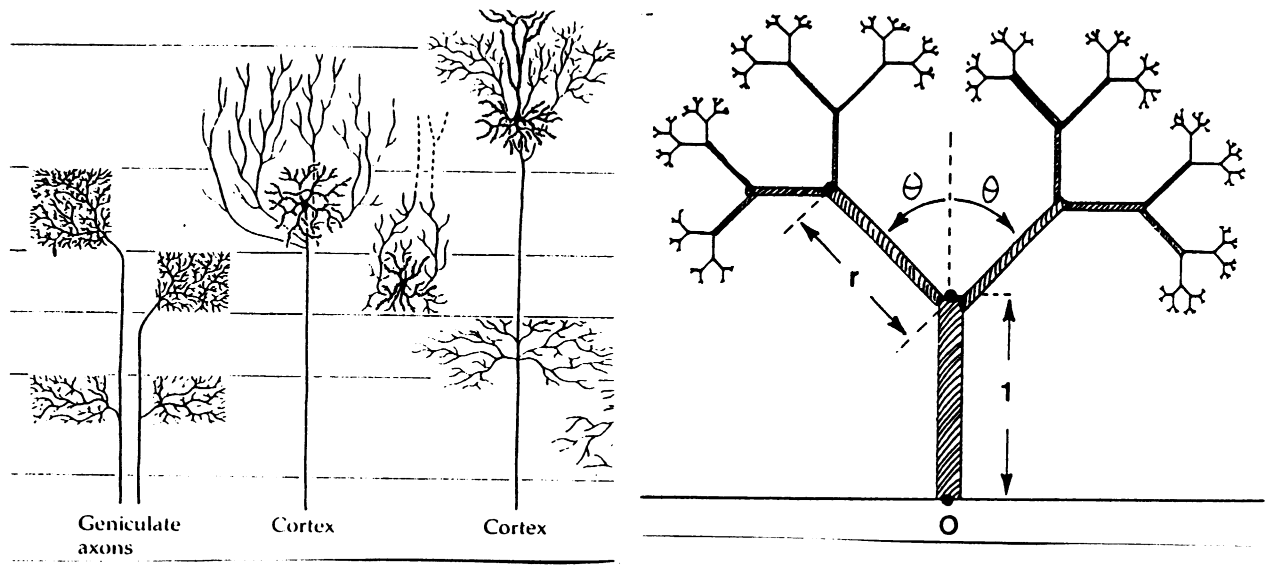
The value of normalized density of the packing of subunits depends on the dimension in which the packing is done. Since packing is constructed from codes for the digital transmission of information, dimensions 11, 12 and 13 are optimal for information processing.That means that MTs are optimal structures for information processing in the neuron.

3. Microtubules and Neuron Geometry

It has been shown in tissue cultures of neuroblastoma cells that MTs determine neuron geometry [1]. By using immunofluroscence it was demonstrated that the shape of the neuron in the process of differentation is related to the microtubule organizing centre (MTOC). Having in mind these experimental facts and theoretical results from the information theory, we believe that MT's dynamic activity in the process of the determination of neuron geomtry possesses *fractal* properties.

MTs, as a part of MTOC and cytoskeleton, generally possess some dynamical properties, which are similar to fractal dynamics. Centrioles possess a nine-fold symmetry and they are constructed of nine triplets, each containing 13 protofilaments. During the process of differenciation, the first step is the maturation of the "mother" neuron, followed by the separation of centriole pairs and their migration to establish the geometry of the "daughter" neuron. MTOC is involved in the orientation, geometry, timing of division, and growth of the neuron in the process of differenciation. MTs with other cytoskeletal structures, determine what neurons dynamically do, when and how they do it, and the type of neurons they are.

Neurons with their dendrites and synapses geometrically look like fractals. Having in mind that cytosleketon determines the geometry of the neuron, we believe that MTs are an attractor of neurons as a fractal.



Schematic diagrams:a) neurons and their cortical connections, b) sketch of a fractal tree.

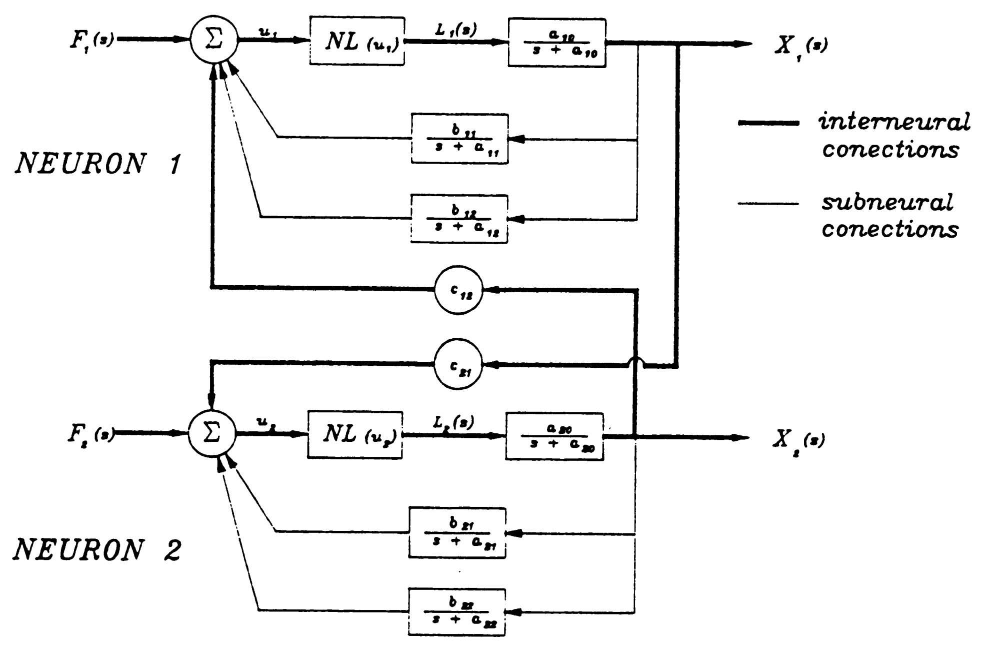
4.A Neural Network based on a Sub-Neural Factor

We have found that the **MT** structure is optimal for information processing on a sub-neural level [6]. Here we consider such nonlinear control networks in which the neuron response depends on: the input signal, sub-neural activities, and its interaction with other neurons. This control model has already been used for response analysis of the vertebrate retina [7], but we believe that it is a better example for the explanation of the neural network based on sub-neural factors.

The neural control network's model based on **MTs** as a main sub-neural factor, may exist in the form:

$$a_{io}^{-1}\dot{X}_i(t)+X_i(t)=NL\{f_i+\sum_{j=1}^{n}C_{ij}X_j(t-\sigma_{ij})+\sum_{k=1}^{m}b_{ij}\int_0^t X_i(\tau)e^{-a_{ik}(t-\tau)}d\tau\}$$

where: **t**-*time*; **n**-*the number of neurons in the network*; **m**-*the number of MT in the subneuron network*; **f_i**-*the external input to the i-th neuron at time* **t**; **$X_{i(t)}$**-*the activity function*; **a_{io}**-*the constant rate characterized by the fact that the stop charge input to the i-th neuron produced an expotential approach from the initial value* **X_i**(o) *to a steady-state firing rate* **X_i** *with a rate constant* **a_{io}**; **b_{ik}**-*inhibition factor for the i-th MT in case* **b_{ik}**<o; *excitation factor for the i-th MT in case* **b_{ik}**> o; C_{ik}-*the intraction coefficient i-th neuron j-th neuron*; σ_{ij}-*the time lag occuring in the transfer of the activity of the j-th neuron to the i-th neuron*; **NL** - $1/(1+e^{-u})$.



Neural network of two neurons with two MT sub-neural activities in each of them

5. Conclusion

The symmetry theory and MT structure lead to the conclusion that the packing of tubulin subunits is equal to information coding. This means that MTs possess code systems which can provide, in the neuron, dynamic information activities.

Sub-neural factors based on cytoskeleton and MTs play an important role in both the control mechanism of neuron geometry and the neural network. Having in mind that geometric neurons look like fractals, we believe that cytoskeleton and MTs are a fractal attractor of the neuron.

From the sub-neural factor point of view we can introduce a new research approach in the field of neural networks in which the neuron is not just a simple *off-on* element, but an element with a history.

References

[1] P.Dustin, "Microtubules", Springer-Verlag, Berlin - New York (1978).

[2] R.Mileusnić et al., Learning and Chick Brain Tubulin, *J. Neurochemistry,* Vol.34, 4:1007 (1980).

[3] S.P.R.Rose, Early visual experience, learning and neurochemical plasticity in the rat and the chick, *Phil, Trans, R.Soc. London B.* 278-307 (1977).

[4] H.Fröhlich, The extraordinary dielectric properties of biological materials and the action of enzymes, *Proc Natl Acad Sci.* 72-4211 (1975).

[5] R.O.Erickson, Tubular Packing of Spheres in Biological Fine Structure, *Science,* Vol.181,No.4101:705-716 (1973).

[6] D. Koruga, Microtubular Screw Symmetry:Packing of Spheres as a Latent Bioinformation Code, *Ann. New York Academy of Sciences,* Vol.466, 953-955 (1986).

[7] M.N.Oguztoreli, G.M.Steil and T.M.Caeli, Control Mechanisms of a Neural Network, *Biol. Cybern* 54:21 (1986).

Comparison of the Moore-Penrose and Drazin Generalized Inverses in Biological Coordinate System Transformations

LACZKO, József, Central Res. Inst. for Physics, H-1525 Budapest, POB 49, Hungary
LE GOFF, Bertrand, CNRS, Lab. Physiologie Neurosensorielle, Paris 75006, France

Introduction

Some local properties of coordinated movements of mechanical devices can be described and investigated in engineering using linear algebraic methods in [2]. Such methods appear to be useful in describing local properties of coordinated actions of multicomponental biological organisms as well; in particular regarding the existence and the uniqueness of a solution selected from an infinite number of possibilities. Special attention is given to the "least squares method" in solving such problems by means of the Moore-Penrose pseudo-inverse of the coefficient matrix of a linear equation system [1]. In an earlier publication [6] one of us [T.S.B.] brought up the possibility of nature's use of the Drazin inverse [3] as compared to the Moore-Penrose generalized inverse. In this paper, we follow up on that suggestion and compare the potential of Moore-Penrose and Drazin inverses in neural computation.

Functional characteristics of the Moore-Penrose and the Drazin generalized inverse

The Moore-Penrose pseudo-inverse is widely used in engineering and more recently also in biology. A characteristic feature of the Moore-Penrose pseudo-inverse is that it provides some kind of optimal solution for a system of linear algebraic equations. Although the optimization criteria might be the minimization of energy in physical applications, it is a more interesting question if it could be an optimization criteria in biological applications.

The Moore-Penrose pseudo-inverse is defined for any matrix whose elements belong to the complex field and, for some special classes of matrices, there are easy computational methods for its calculation [9]. However, there are some algebraic properties that the Moore-Penrose pseudo-inverse does not possess. Therefore, a mathematician can theorize that it can be useful for nature to apply another types of pseudo-inverses like the Drazin inverse [6].

The Drazin inverse is useful in solving linear differential equation systems, which is important e.g. in calculation of the trajectory movements. It is also applicable if the work space is not linear (but algebraic), where besides the vector and matrix addition another operations (such as associative multiplication) are also defined, permitting, E.g. consecutive execution of linear transformations.

Application to biological systems

An important question for coordinated movements of biological and artificial organisms is how to express the spatial movement by overcomplete coordinate systems, defined intrinsically by the moving structure. For instance, how to to decompose a displacement vector into more components than the number of dimensions of the external work space? Tensor Network Theory (cf. [9]) offers a common mathematical language for describing coordinated movements of biological and man-made structures [8]. An even more important remaining question is, however, how to select the coordinate system. It is a challenge to biologists and neuroscientists to find the correct internal coordinate systems for biological systems and especially for underlying neural networks that control them.

Once a coordinate system is defined, one faces distinct mathematical and computational problems. In order to decompose a vector into more components than the number of dimensions of the work space, a linear equation system has to be solved where the coefficient matrix is singular. In such a case motor execution has an infinite number of solutions. A customary way of selecting one particular solution is by using the least squares method, applied by Tensor Network Theory, the most compact geometrical theory of neural networks.

A key operation in Tensor Network Theory for coordinating an overcomplete number of components of a biological action is the covariant-contravariant transformation. (e.g. in a sensorimotor transformation; [9]). In this case the transformation matrix is a real-valued symmetrical matrix (the Gram-matrix of the intrinsic coordinate system). This matrix is singular because of the overcompleteness of the intrinsic biological coordinate system (e.g. joint coordinate-system.) Thus, the contravariant-covariant transformation is not invertible. However, a subspace may exist on which this transformation (the restriction of the transformation to the subspace) is invertible. Tensor Network Theory can be seen, therefore, as raising the rather general question what is the definition of those hyperspaces in which neural networks express their transformations to produce biological actions?

Internal hyperspaces and the Moore-Penrose versus Drazin generalized inverses

It appears to be a natural, although not self-evident question to examine the class of hyperspaces on which the transformation is invertible. Taking a particular transformation, we have the following question at hand: does exist any subspace that the restriction of the transformation to this subspace is invertible?

When investigating the case of square matrixes, take the following linearization equation system:

$$\mathbf{Ax=b}$$

where **x**, and **b** are vectors of the n-dimensional complex vector space, and A in an **n x n** matrix with complex elements. (It defines a linear transformation **A** on the n-dimensional complex vector space).

The range space of A is the set of all vectors which can be the result of the multiplication of the matrix A by any vector. Denote the range space as R(A).

The nullspace of A is the set of vectors by which the multiplication of the matrix A, results he 0-vector. Denote it as N(A).

It is easy to recognize that to make the linear transformation defined by the matrix A a one to one transformation it must be restricted to a subspace, complementary to N(A).

Given a scalar product in the vector space, one can define angles between vectors. In this case an obvious complementary subspace is the one which is orthogonal to N(A). Denote it by $N(A)^{\perp}$. This space is the range space of the conjugate transpose matrix of the matrix A. The conjugate transpose matrix is denoted by A^*.

This suggests the functional DEFINITION of the MOORE-PENROSE PSEUDO-INVERSE:

$$A^+V=0 \qquad \text{if } V\in R(A)^{\perp}$$

and

$$A^+V=(A|N(A)^{\perp})^{-1} \qquad \text{if } V\in R(A)$$

Pellionisz applied a constructive definition of the Moore-Penrose pseudoinverse of symmetrical matrices by their eigenvectors and eigenvalues. This definition is very helpful for calculating the pseudoinverse. A quantitative example for a neural process which implements such a transformation is elaborated in [10].

The functional definition of the pseudoinverse helps one to generalize the definition.

If the matrix is symmetrical with real elements, then $R(A)=R(A^{*})=R(A^{+})$ and its range space is orthogonal to its nullspace: $R(A)=N(A)^{\perp}$. In this case, the Moore-Penrose pseudo-inverse defines a partial isometry on $R(A)$.

In the covariant-contravariant transformation, the range space of the transformation will satisfy the required condition. Thus one easily finds a subspace which is invariant to the transformation. It has the following properties:

1. The restriction of the contravariant covariant transformation tot his subspace is invertible.

2) Not only the covariant but also the contravariant coordinate vector which is found by the Moore-Penrose pseudo-inverse is the element of the subspace.

Consequently, one can define an internal subspace in which the decomposition of an external vector to overcomplete number of components is unique: to every covariant coordinate vector of the external vector there is one and only one contravariant coordinate vector in the subspace of the covariant coordinate vectors.

One finds not only an internal coordinate system but a distinct internal subspace which is defined by the coordinate system. The internal coordinate system may be defined by the geometry of the skeleto-muscular system in the case of movement coordination.

Unfortunately, one does not always faces real-valued symmetrical matrices. Matrices which describe functions or transformations implemented by neural networks, e.g. matrices of synaptic efficacies of neural networks ,are not necessarily symmetrical. For a singular non-symmetrical square matrix the above mentioned 1, 2 properties usually do not hold. It is not always true, therefore, that the matrix is invertible on the range space and that the range space of the pseudo-inverse is the same as the range space of the matrix. What kind of inverse might be used, what kind of invariant subspace might be found in such cases?

It is suggested, in such cases, to search for an invariant internal subspace using the Drazin inverse of the matrix.

The first definition which was given by M.P. Drazin [3] in 1958 was an algebraic definition. Here, a functional or geometrical definition is given.

If A is a real valued symmetrical matrix then $R(A)$ is the orthogonal complementary subspace of $N(A)$. This is not true for every square matrix but there always exist a non-negative integer **k** such that $R(A^k)$ is a complementary subspace to $N(A^k)$. This **k** is called the index of the matrix **A**.

With this notation, the restriction of **A** to $R(A^k)$ is invertible on $R(A^k)$.

DEFINITION of the DRAZIN INVERSE A^D of a square matrix A:

$$A^D v=0 \quad \text{if} \quad v\in N(A^k)$$

$$\text{and} \quad A^D v=(A|R(A^k))^{-1} v \quad \text{if} \quad v\in R(A^k)$$

Let us remember that for real-valued symmetrical matrices $R(A)=N(A)^{\perp}$. (The index of such matrices is 1). Consequently, the Moore-Penrose pseudoinverse and the Drazin inverse of a real-valued symmetrical matrix are identical.

Conclusion

To find a generalized inverse of transformations implemented by neural networks, Tensor Network Theory suggests the use the Moore-Penrose generalized inverse. This is a natural way to find invariant internal subspaces. For real-valued symmetrical matrices, quantitative examples where elaborated, and for movement coordination graphical computer programs were written using the Moore-Penrose pseudo-inverse [4][5][7].

As an extension of the results for non-Hermitian matrices, the application of the Drazin generalized inverse is suggested. It was shown above that for this class of matrices the Drazin inverse or the Moore-Penrose pseudo-inverse will yield the same result.

The relationship between partial isometry and the Moore-Penrose pseudo-inverse of Hermitian matrices was examined. In the future, in order to find internal invariant hyperspaces of non-Hermitian neural transformations, the relationship between the Drazin inverse of neuronal transformation and partial isometry might be important to further investigate.

References

1. Albert, A. "Regression and the Moore-Penrose pseudoinverse." 1972 Academic Press. New York.

2. Campbell, S. I. and C. D. Meyer. "Generalized inverses of linear transformations." 1979 Pitman Publishing LTD.

3. Drazin, M. P. Pseudoinverses in associative rings and semigroups. Amer. Math. Monthly. **65**: 506-514, 1958.

4. Laczkó, J. Coordinated activation of multijointed artificial limbs. SIAM Conf. on Applied Geometry. A47, Albany, N.Y.,1987.

5. Laczkó, J., A. Pellionisz, H. Jongen and C. C. A. M. Gielen. Computer modeling of human forelimb muscle activation in multidimensional intrinsic coordinate frames. Soc. Neurosci. Absts. **14/2**: 955, 1988.

6. Laczkó, J., A. J. Pellionisz, B. W. Peterson and T. S. Buchanan. Multidimensional sensorimotor "patterns" arising from a graphics-based tensorial model of the neck-motor system. Soc. Neurosci. Absts. **13**: 372, 1987.

7. Lestienne, A., P. Liverneaux, J. Laczkó and A. Pellionisz. Tensor model of the musculo-skeletal head-neck system of the monkey. **Suppl. to Vol. 22.**(Proc. of IBRO II. World Congress): S658, 1987.

8. Pellionisz, A. Brain theory: Connecting neurobiology to robotics. Tensor analysis: utilizing intrinsic coordinates to describe, understand and engineer functional geometries to intelligent organisms. J. Theoret. Neurobiol. : 185-211, 1983.

9. Pellionisz, A. Coordination: A vector-matrix description of transformations of overcomplete CNS coordinates and a tensorial solution using the Moore-Penrose generalized inverse. J. Theoret. Biol. **110**: 353-375, 1984.

10. Pellionisz, A. and R. Llinás. Tensor Network Theory of the metaorganization of functional geometries in the CNS. Neuroscience, **16**: 245-274, 1985.

GENNET - System for Computer Aided Neural Network Design Using Genetic Algorithms

Borut Maričić
Zoran Nikolov

CVTŠ KoV JNA, Ilica 256b, YU-41000 ZAGREB, Yugoslavia

Abstract

The problem of finding optimal neural network architecture for solving a given task is introduced. Basics of genetic algorithms as viable optimization method are exposed. Characteristics of implemented system for network optimization using genetic algorithms and current status of the work are described. Plans for the future research are presented.

1. Introduction

Currently a number of neural network models exist. However, a rather common problem faced by neural network designers is the lack of theoretically grounded advice on network architecture to be chosen in order to solve a given problem. Some analysis though [1], give insight in the network architecture required to enable formation of decision regions of a certain type. However, in multidimensional problem spaces, it is difficult or even impossible to visualize the simplest type of decision regions that would suffice. Hence, one still has to experiment a lot in order to find a network architecture that yields acceptable results with minimum computational effort. This process can be seen as a trial-and-error based one, during which network architecture parameter space is investigated. This is usually done with network simulators [2] designed to make this intrinsically uncomfortable task as comfortable as possible.

2. Designing Neural Networks Using Genetic Algorithms

Thus, design process of a neural network of a given model may be viewed as an optimization process: the most appropriate network architecture for solving a given problem has to be found. Analogy may easily be noticed in the real world - living beings may be thought of as systems trying to overcame the entropy by storing the information about the given "infinite" environment in finite space and time. Each phenotype innately possesses some knowledge which makes it more or less fit for "life-time" learning. Evolution appears as a mechanism for phenotype innate knowledge optimization. Mathematical models of evolution are known as genetic algorithms [3, 4]. Recently an optimization algorithm of this class [5, 6], that is suitable for execution on connectionist architectures, has been proposed. It is also suggested to use genetic algorithms for neural network design [7]. Before proceeding with an overview of a system that realizes that task, it is necessary to introduce some common concepts of genetic algorithms. Basic idea of such an algorithm is sketched in Table 1.

```
Create initial population of entities to be optimized;
Evaluate entities in the population;
repeat
  | Choose parent entities, whose genotypes are to be used in order to forme
  |   new ones;
  | Forme new genotypes;
  | Evaluate entities formed according to new genotypes;
  | According to some criteria, replace some entities in previous population
  |   with new entities, thus forming a new population generation;
until terminating criteria is achieved.
```

Table 1. The basic idea of genetic algorithm

In the neural network design context, entity is one of the possible network architecture variations; evaluation of the network implies its testing in order to acquire some numeric measure of its fitness for solving a given problem; network genotype represents suitably encoded network architecture parameters; and finally, formation of new genotypes implies using some genetic operators (such as crossover, mutation, inversion) on selected genotypes, thus forming new ones.

3. System GENNET

CAND[1] system GENNET is currently designed and implemented to support optimization of error back-propagation networks [8]. It consists of two components, Figure 1: GEN and NET. While GEN embodies the essence of genetic algorithm, NET creates the network according to the genotype produced by GEN, and then evaluates it.

GEN. Network designer communicates with the system through GEN. He/she may choose from one of the three genetic algorithms [5]: iterated genetic search using uniform combination, iterated genetic search using ordered combination, or stochastic iterated genetic hillclimbing. Prior to the network optimization, genetic algorithm performance may be tested on one of the three predefined two-dimensional functions. Before starting the network optimization process, user should define characteristics and size of the network parameter space being searched. This is done using a simple language which enables one to specify which parameters are constant and which are being optimized. The latter ones may be of real (such as learning rate constants) or integer type (such as number of hidden layers and their sizes). For each optimized parameter, its minimum and maximum value as well as change resolution are specified. During its operation, GEN invokes NET when evaluation of a network is required according to the used algorithm. Interprocess data passing is realized via files.

[1] CAND stays for Computer Aided Network Design, as suggested by Vladimir Hinič.

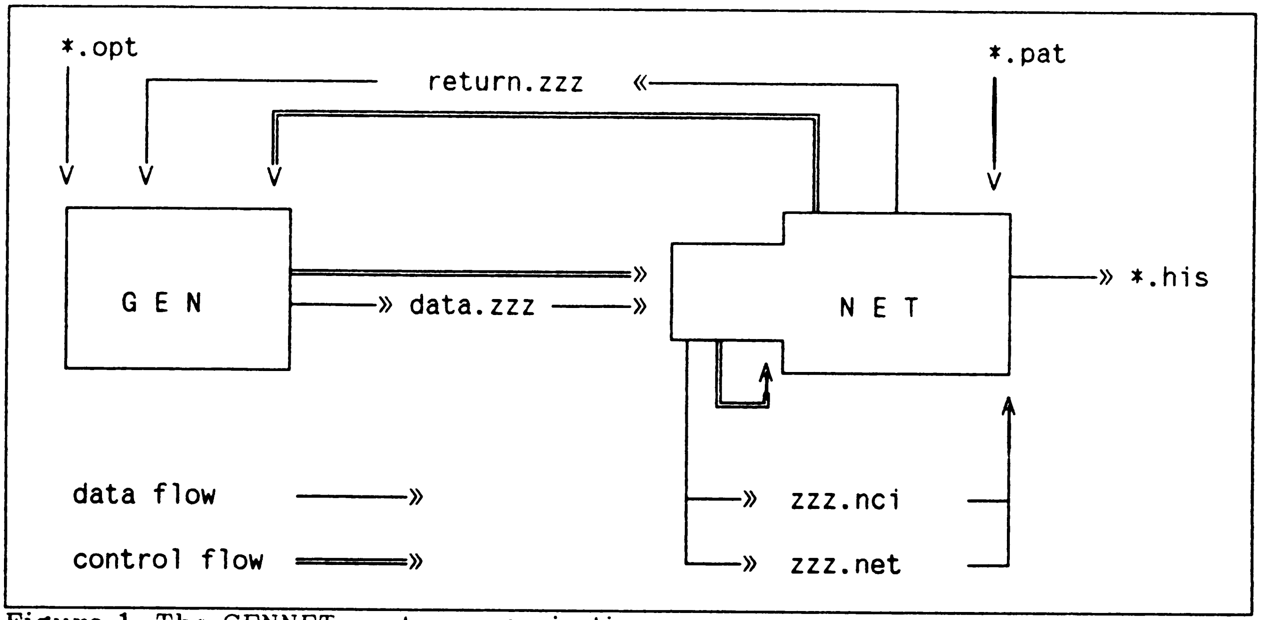


Figure 1. The GENNET system organization

NET. This component may in fact be used as a stand alone error back-propagation network simulator. In this mode user may interactively create network architecture description, create the training set, save learned weights, get previously saved weights, test the network and change some of the learning algorithm parameters. However, NET is able to detect if it is being invoked by GEN. In that case it creates a network according to its description prepared by GEN. Then it initiates a predetermined number of training sessions, each one lasting for a predetermined number of epochs. Based on the learning history, it formes a numerical measure of network's quality. Control is then returned to GEN.

During the whole optimization process several log files are continuously being updated.

4. Current status and future work

Described system is implemented using C language. Currently it is being ported to the UNIX environment on CONVEX system available at Zagreb University Computing Center. We intend to report on some simple applications of it in the next six months. The problem of choosing adequate network fitness measure appears to be particularly interesting for investigation. Currently the fitness is determined on the bases of minimum total sum of squares error encountered during fixed length training session. However, rather frequent cases of learning curve oscillations (which are due to inadequate learning rate constants) should also be taken into account.

We are aware of prohibitively large time complexity of the system described, when applied to the optimization of large parameter spaces and problems with large training sets. That is why such a system is not seen as a replacement for the existing ones, but rather as a subsystem of some advanced (and more distant) CAND system. Such a system might be used in "manual mode" offering highly interactive graphical man-machine interfaces, library support of various network models, expert (possibly classifier) advising system on design tracks to investigate, information retrieval system enabling convenient access to the body of network design know-how. However, it could also be used in "automatic mode", i.e. it would posses capability of network architecture optimization of a sort described in this paper.

Acknowledgement

The help of our colleagues at the Faculty of Electrical Engineering at Zagreb, through which we are given access to Zagreb University Computing Center CONVEX facility, is gratefully acknowledged.

5. References

[1] Huang W Y, Lippmann R P, **Comparisons Between Neural Net and Conventional Classifiers**, Proceedings of the IEEEE First International Conference on Neural Networks, San Diego, California, June 21-24, 1987, vol. IV, IV-485 to IV-493

[2] D'Autrechy C L, Reggia J A, Sutton III G G, Goodall S M, **A general-purpose simulation environment for developing connectionist models**, Simulation, vol. 51, no 1, 5-19 (1988)

[3] Holland J H, Holyoak K J, Nisbett R E, Thagard P R, **Induction - Processes of Inference, Learning, and Discovery**, The MIT Press, Cambridge, Massachusetts, 1986

[4] DeJong K, **Adaptive System Design: A Genetic Approach**, IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-10, no. 9, 566-574 (1980)

[5] Ackley D H, **Stochastic iterated genetic hillclimbing**, PhD Theses (CMU-CS-87-107), Department of CS, Carnegie Mellon University, Pittsburgh, PA (1987)

[6] Ackley D H, **An Empirical Study of Bit Vector Function Optimization**; In: Davis L (editor), Genetic Algorithms and Simulated Annealing, Pitman, London, 1987

[7] Bergman A, **Variation and Selection: An Evolutionary Model of Learning in Neural Networks**, Abstracts of the First Annual INNS Meeting, Boston, 1988, pg. 75

[8] Rumelhart D E, Hinton G E, Williams R J, **Learning Internal Representations by Error Propagation**; In: Rumelhart D E, McClelland J L (editors), Parallel Distributed Processing - Explorations in the Microstructure of Cognition, Volume 1: Foundations, The MIT Press, Cambridge, Massachusetts, 1986

SYNTHETIC CEREBELLUM;
WHAT IT MAY DO, AND HOW IT MAY DO IT.

Mahmood J. Nahvi
Electronic and Electrical Engineering Department
California Polytechnic State University
San Luis Obispo, California 93407 USA

Cerebellum as a Controller

Cerebellum plays an important role in posture and movement control. One of its functions is to coordinate complex sequences of motor acts such that the total movement is smooth, accurate and without osscillation and overshoot. When this function is lost movement is decomposed into individual segments which are inaccurate and oscillatory. To perform the above function, in addition to signals from higher stations of the central nervous system, the cerebellum receives short latency proprioceptive and extroceptive feedbacks from periphery (including auditory, visual, and tactile information). The cerebellar cortex has a unique morphology and architecture, with a repetitive geometrical structure which is the same throughout the whole cerebellum [1].

As a controller in the engineering sense, it handles a large number of control variables (related to large set of nerve and muscle fibers) and involves several reference spaces (position/ velocity spaces in cartesian and joint coordinates , and the spaces of neural activity at the motor unit level or higher up.)

What considerations and constraints apply to the design of a "cerebellar controller" in a synthetic motor control system? And, if such a controller is to be implemented by an artificial neural network, how do those considerations and constraints affect its architecture? The present short note touches upon some aspects of the above questions.

Trajectory Planning or Dynamics Compensation?

Neural signals arriving at the muscles to produce a desired movement have two functions: 1) They carry the information about the pattern of the desired movement, and 2) They compensation for dynamics of the muscles and the load, including the environment. The first function comes from trajectory planning which produces "reference commands". The second function makes the system track the "reference commands" despite its dynamics, variations, and interferences. When the system has negligible dynamics, the first function dominates. The patterns of neural activities then simply reflect the patterns of the desired motion. However, when the system has appreciable dynamics the neural signals become complex despite of the desired movement being simple. The complex patterns of EMG activities recorded from muscles in a fast extension of the loaded human arm, with the desired motion being a smooth displacement of the hand with a simple bell-shaped speed profile, is an example of such a case [2].

In synthetic control systems such as robotic manipulators trajectory planning is normally done separately. The controller, being a simple feedback servo-loop or an implementation of an advanced adaptive algorithm including feed-forward and feedback, is then subordinate to the trajectories. Its task is dynamic compensation.

However, if the information about the performed movement and its consequences is sent back through several nested delay loops built around various reference spaces mentioned previously, then trajectory planning and dynamics compensation become coupled. In such a case, even though the controller is subordinate to the movement plans imposed on it from a central system, it needs to be concerned with trajectory planning. Therefore, if a synthetic cerebellum is to take over the movement unexpectedly, its capabilities should go beyond that of "smoothening" the multiple segments of the pre-planned trajectories.

Dynamics Compensation by Matching the Command to the System

Dynamics compensation does not always mean solving for the inverse in the equation of motion. The following two inter-related examples illustrate control signals which are not found by inverse dynamic or its approximation. Rather, it may be said that they have been modified to match the reference command to the system and the environment, observing given constraints.

Example 1. In a second order system with damped oscillatory step response a controller can produce a deadbeat step response in finite time as illustrated in Fig.1. Other classes of modified inputs can also move the system from an initial state to a final state within a specified time with a smooth transition. The choice depends on optimality criteria, desired transition time, speed profile, constraints on magnitude of the input, its rise time, the area under it, etc. The deadbeat effect is neither due to approximating the inverse of the system, nor friction, nor reversal of the input to cause active break. Rather, the controller may be said to match the command to the system, in open-loop form, feedback, or their combinations.

Example 2. In rapid extensions of the arm in vertical plane the hand moves smoothly with a bell-shaped speed profile. Forearm trajectories are also smooth and have speed profiles similar to that of the hand. The upper arm trajectories may have three segments and bi-modal speed profiles. As one moves from central to peripheral points on the arm, the three segments merge together, producing smooth curves (Fig. 2).

Generally, the triphasic activities of biceps and triceps in such motions have been modeled by acceleration and deceleration periods in accordance with approximation to the inverse equation. However, the biceps activity at the peak of forearm's speed, and its quietness during the deceleration phase of the forearm can hardly be interpreted by such a simple model and it is contrary to expectations derived from it [3]. An alternative interpretation based on the concept of matching the signa to the system can resolve the above apparent contradiction [4].

The simple refrence commands in the above two examples have been modified, under some constraints, into more complex control signals to match to the system and the environment.

Retina and Cerebellar Cortex as Sensory and Motor Interfaces with Outside; Similarities and Differences

Suppose the function of synthetic cerebellum in a sensory-motor system is to interface its motor output to the outside world. In a way, this function is a mirror image of the function of a retina which interfaces the outside world to the sensory input of the system.

Similarities:

The retina enhances the incoming sensory image, being static or time-varying, by extracting its useful features. It produces mappings of many-to-one nature. This helps reduce the amount of data going to the system.

The cerebellum enhances the out-going motor images, being static as in posture or time-varying as in an evolving movement, by introducing necessary features in the motor commands. It produces maps of one-to-many nature. This helps reduce the amount of data needed to come from the system.

The above similar functions may be implemented by similar basic neural network circuits such as lateral inhibition.

Differences:

The perception of time-varying sensory images which come to the system may not require high speed processing. However, generation of time-varying motor images which go out to produce evolving movements, especially when the desired movements are fast, should be rapid [5]. Moreover, an ability to scale the time axis with any desired factor is a very advantageous featur for the synthetic cerebellum. Therefore, for the purpose of time axis generation, the synthetic cerebellum needs to employ not computationally extensive algorithms but hard-wired circuits with adjustable parameters. Parallels from neural network of the living cerebellum may be useful in this respect [6,7].

References

[1] M. Ito, "The Cerebellum and Neural Control", Raven Press 1984

[2] M.J. Nahvi, "Fast movements of human arm: Reflections on control issues", IEEE Intl.Conf.SMC. Nov. 989, Cambridge., MA.

[3] C.D. Marsden, J. A. Obeso and J. C. Rothwell, "The function

of the antagonist muscle during fast limb movement in man". J. Physiol. vol. 335, pp 1-13, 1983

[4] M.J. Nahvi, "Extensions of human arm in vertical plane", parts I and II, prepared for publication, 1989

[5] M.J. Nahvi M.J. and N. A. Farahbakhsh, "Characteristics of fast hand movement", Proc. 4-th Intl. Conf. in Electrical Engineering, pp 572-581, Shiraz, May 1974

[6] M.J. Nahvi and K. Daroudi, "Propagation of activity in a uni-directional network with feedback; Parallels with the cerebellar cortex", abstarct, the 14th Annual Meeting, Society for Neurosciences, October 10-15, 1984, Anaheim, CA.

[7] O.Oscarson, "Functional units of cerebellum; Sagital zones and microzones", Trends Neurosci. vol. 2 pp143-144, 1979

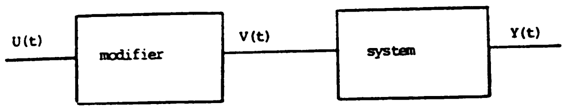
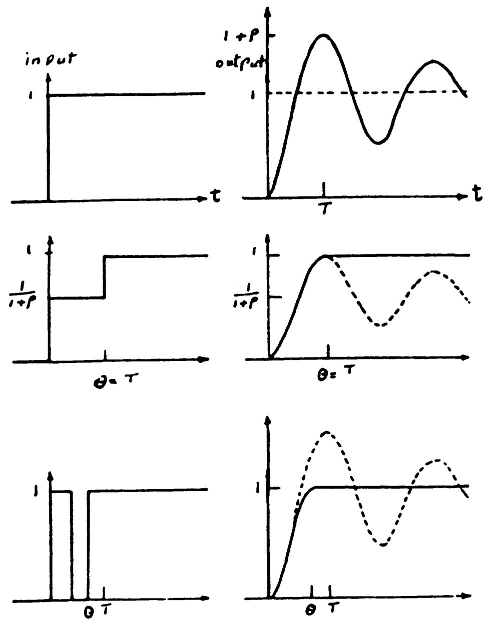


Fig. 1



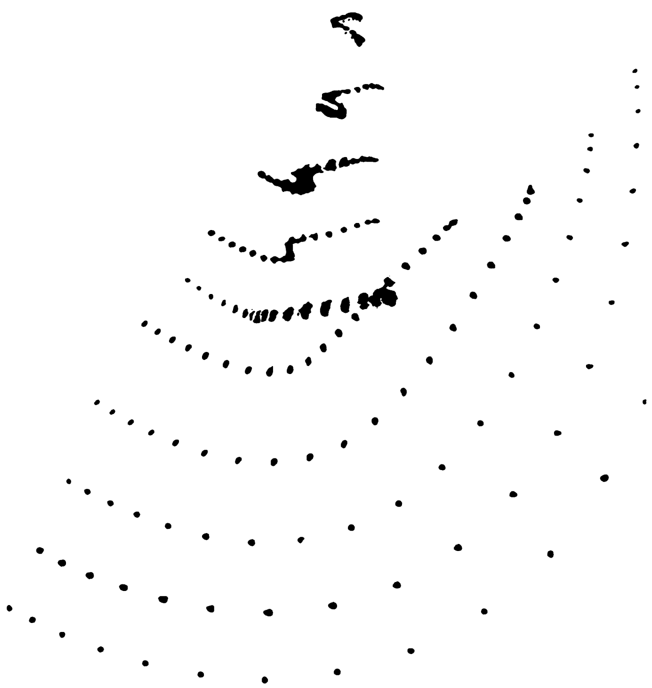


FIG. 2

Cognition and Neural Computing An Interdisciplinary Approach

Markus F. PESCHL[1]

Keywords: cognitive modeling, cognition, artificial life, adaptation, Parallel Distributed Processing, Learning, structural coupling.

1 Motivation

The symbolic approach of orthodox AI has reached its limits in the domain of cognitive modeling. This is due to a non-interdisciplinary or – better to say – pseudo-interdisciplinary manner of investigating cognition. To put this into a few words:

Both orthodox AI and cognitive psychology use models which are restricted to symbols or linguistically describable behavior. NEWELL & SIMON's idea of describing cognitive processes by means of symbols is called the "*Physical Symbol Systems Hypothesis*" (e.g. [NEWE 76]). This approach is quite obvious, it expresses, however, the unreflected use of language, symbols, etc. and can withstand neither philosophical nor epistemological arguments. It ignores, for example, the fact that our actions are controlled by unconscious subcognitive processes ("tacit knowledge" [POLA 66]) having nothing in common with language, words,.... One only can speak of them in terms of natural language from the *observer*'s point of view. Stated another way, this means that we have to distinguish between the observed behavior (and how it is described) and the way how this behavior is generated (i.e. the discrimination of the inner and outer point of view). Orthodox AI does not make this distinction and tries to model inner processes by means of outer (linguistic) descriptions. This seems to be one of the main reasons why the "symbolic approach" is doomed to failure in the domain of cognitive modeling.

The aim of this paper is to show an alternative approach to investigating, modeling and finding a (more) adequate way of explaining, describing and interpreting cognition. For that reason we have to search for a way of modeling cognition which is neither resting on nor restrited to language (i.e. linguistic descriptions which are mapped to symbol manipulating systems), in which language is embedded, however; in other words: as linguistic descriptions are only categories in the *observer*'s cognitive domain, the model should *not* make use of language or symbols in order to describe or generate cognitive processes; so we have to look out for an alternative approach.

2 PDP and Cognitive Modeling

Parallel Distributed Processing (PDP, "neural computing", "connectionism",...) represents an alternative approach and offers some advantages in the domain of cognitive modeling, because of being close to the natural nervous system's organization (in H.MATURANA's sense). It is important to annotate, however, that our cognitive model does *not* follow the usual PDP paradigm (symbolic input is coded as a pattern of activations spreading through the network; the output pattern is interpreted as a code for a symbol); it is not provided with an interface for symbolic in/output. Communication with the environment takes place over sensors and effectors; i.e. there is *only* physical interaction with the medium. Language is *embedded* in this way of physical interaction with the environment.

Thus our cognitive model has *direct* access to the medium and we are not confronted with the problem of linguistically mapping environment to the cognitive system (as it has to be done in orthodox AI) any more, because the interface is coded *analogously* (i.e. the sensors of a modality transforms the stimulus into an *unspecific* signal (neural activity) being proportional to the stimulus' intensity). On the other hand the problem of architecture and how to position sensors and effectors arises. The output of the cognitive system is generated by effectors changing the medium or changing the system's position in the medium. This implies the following *feedbackloop*: action by the effectors $\Rightarrow$changes in the environment $\Rightarrow$changes in the sensory

[1]Dept. of Epistemology and Cognitive Science (University of Vienna), Sensengasse 8/9, A-1090 WIEN, AUSTRIA, EUROPE, Tel. -43 222/42760141; Fax: -43 222 488838.

⇒alteration of the inner state ⇒action by the effectors ⇒...ad infinitum. The network's architecture itself is highly recursive, giving rise to a system with an inner state being perturbated by the environment.

2.1 Epistemological Aspects

If we are interested in finding an adequate model for cognition, it is by far not sufficient to consider only aspects of computer science and (mathematical) logic (as orthododx AI does). We rather have to take into account an *interdisciplinary* point of view, as the investigation of topics such as knowledge, common sense, perception, learning, language, etc. *requires* an *interdisciplinary* approach. The idea of this project is to show the possibilities and implications of combining H.MATURANA's epistemological and neurobiological concepts with the PDP paradigm. Due to the very limited space we can pick out only two of the most most important points:

(i): The universal concept of *structural coupling* is the crucial point in MATURANA's theories [MATU 78] and can be modeled with neural networks as they are capable of adaptation in very small increments; coupling to the medium or to other cognitive systems arises as a result of a sequence of mutual perturbations and adaptations (i.e. recursive interactions).

(ii) MATURANA's idea of *knowledge* and its representation [MATU 70E] is very near to the concepts of *distributed representation* (e.g. [HINT 86]). We have to give up the idea of mapping the environment to linguistic structures, but rather think of a sort of '*representationfree representation*' (as I call it); there is no need of an explicit (linguistic) relation between representans and representandum. As one consequence the notion of knowledge has to be redefined as the ability of behaving adequately in a certain situation [MATU 70E].

In short, MATURANA's approach to understanding and describing cognition represents a sound basis for cognitive modeling covering and considering as well philosophical, epistemological, neurobiological and empirical questions (MATURANA's theories arise from cybernetics and system theory).

3 The Model and its Environment

The basic idea is to construct a model being 'directly' linked to its environment, without having any symbolic instances inbetween, and using a PDP network for generating its behavior.

- Our cognitive model (*C.M.*) moves around in a two-dimensional plane (⇒fig 1) and itself has the shape of a rectangle.
- At each time step – we use a *discrete* time system – *C.M.* has a certain *velocity* v (being generated by the effectors) and a certain angle φ (⇒fig. 1).
- At the moment *C.M.* is equipped with four *effectors* being controlled by the "motor units"' activations. These effectors are responsible for the movement, as they produce an acceleration (being proportional to the motor unit's activation) in each of the four directions at each time step. Suming up the 'old' velocity vector, the resulting velocity of the four accelerations and a friction term results in the new position and velocity of the *C.M.* .
- So far *C.M.* is provided with two types of *sensors*:
  - (i) '*tactile sense*': as can be seen in fig. 1 *C.M.* is surrounded with 8 tactile sensors; i.e. they are like cats' whiskers and signalize a collision with an object in the medium.
  - (ii) '*visual system*': *C.M.* is equipped with a (2-dimensional) retina being studded with optical receptors, each of which transforming the intensity of the light beam striking it into a neural activity which perturbates the inner state of the cognitive system. The intensity of each light beam is computed by an adapted version of a ray tracing algorithm (in two dimensions) as it is well known from computer graphics and computer animation [FOLE 82].
- The *environment* consists of boxes (at the moment); i.e. the *C.M.* 's task is to learn to avoid collisions with those boxes having an certain light intensity (to differentiate them from the rest of the environment).

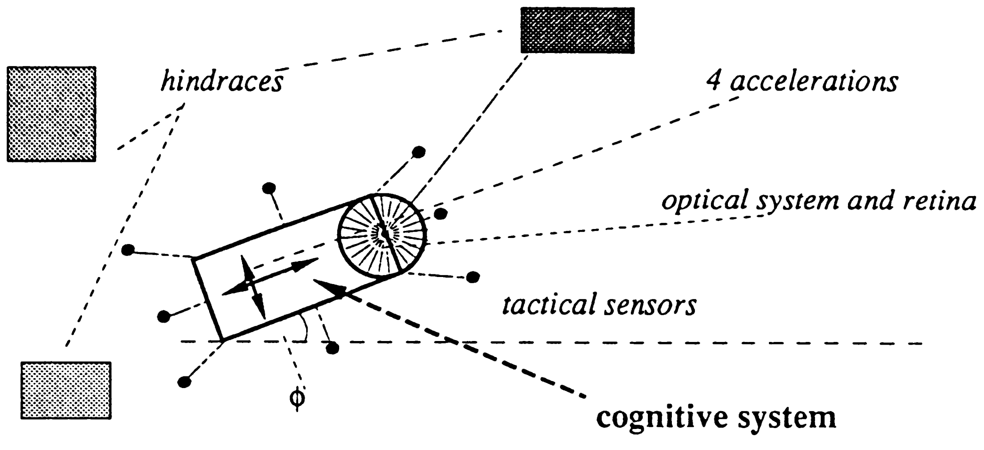


Figure 1: The cognitive model and its environment.

- For the future we are planing to extend the complexity of the environment and tasks to learn. *C.M.* will be provided with more complex effectors and sensors (e.g. for manipulating its environment, for 'tasting', etc.).

3.1 Simulation and Architecture

C.M. may be provided with several network architectures being connected to the peripheral system (i.e. effectors and sensors). ***Lateral inhibition*** is an important part of the architecture to obtain sharp sensory data for further processing. As already stated, the *C.M.* is equipped with a recursive architecture, so that the inner dynamics is responsible for the structure determined organization being perturbated by the signals which are received by the tactile and optical receptors.

Learning takes place at each moment; so we do not differentiate between a learning phase and a 'running phase'. This seems to be a crucial point concering the adequacy of the cognitive model, as each interaction with the environment and/or of the inner states with each other brings about a change in the system's knowledge. As the activation- and weight values are continuous (e.g. $a_i \in [0,1]$) we use the HEBBian learning rule [HEBB 49] for continuous values, being capable of *increasing* and *decreasing* weights. We are working on a modified version of this learning algorithm bringing better results. There are as well weights which are changeable as weights having a fixed value (e.g. connections being responsible for lateral inhibition).

As already stated, methods from computer graphics are used for the computation of the cognitive system's position, of visual stimuli, etc. In eq. (1) an example for the transformation matrices is given, where $(x'\ y'\ 1)$ is the *C.M.* 's new position in the (x,y)-plane, φ is the rotation angle (as shown in fig. 1), Δx and Δy are the translation increments in x and y direction and $(x\ y\ 1)$ is *C.M.* 's old position.

$$(x'\ y'\ 1) = (x\ y\ 1)\begin{pmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \Delta x & \Delta y & 1 \end{pmatrix} \tag{1}$$

3.2 First Results

It is very interesting to observe the *C.M.* just after initialisation. As weights have random values movements are quite chaotic and make think of a very young animal moving around clumsyly and 'exploring' everything. As the *C.M.* is learning at each moment, movements get more planed and the number of collisions with the boxes is decreasing. It is important to say, however, taht the (network) architecture has to be designed in such a way that it is possible to learn to avoid the hindrances for the *C.M.* . There is no need of any symbolic structure which says that it has to learn to avoid collsions with the boxes (this knowledge is learned by adaptation).

It can also be observed that it is very rarely for the *C.M.* to come into the situation of having to learn; i.e. most of its time *C.M.* is just moving around without colliding and thus nearly no learning the task of avoiding the hindraces. This seems to be an important observation, because natural cognitive systems are as well not always learning one task, but rather spend most of their time with roaming about. They relatively seldom get into the situation of having to learn.

4 Conclusion and Perspectives

A non symbolic model for cognition has been presented. As it is directly coupled to the environment *no* symbolic in/output is needed any more. This is due to the paradigm of PDP providing the possibility of subsymbolic processing. This interdisciplinary approach which is influnced by H.MATURANA's concepts of cognition, language, etc. has some interesting implications which can be summed up as follows:

- Behavior may be seen and investigated from two points of view: (a) the '*outer*': one can observe the *C.M.* in the same manner as behaviorists do; inner processes are of no interest. This is a sort of analytical approach.

  (b) the '*inner*': the alternative approach could be called the *synthetic* or *generative* approach. Designing the network architecture means building up the basis for generating behavior, which is observed as described in (a).Thus it is possible to combine those two aspects (the analytical and synthetic) and to investigate the influences of several architectures on the observed behavior.

- *Language* is embedded in the physical interactions and is seen as a phenomenon in a very complex consensual domain. We have to investigate *low level* (non linguistic) interaction *at first*, and only then we are capable of investigating (natural, symbolic) language (and not the other way around as orthodox AI does!).

- The *constructivist* view of knowledge (which is represented in a distributed way) is placed into the foreground, as the environment is perceived by a set of sensors which determine which sort of information can get into the system. As the *C.M.* is capable of learning, architecture determines *how* the environment is represented in the network (no linguistic mapping!).

- For future work we think of enlarging the environment, the motorsystem and sensory, test various architectures, etc. In a further step it is planed to simulate two or even more cognitive systems (at once). Thus not only the interaction environment ⇔ *C.M.* , but also the interaction *C.M.* ⇔ *C.M.* may be investigated ("society of *C.M.* s").

References

[FOLE 82] Foley J.D. & van Dam A. (1982): Fundamentals of Interactive Computer Graphics; *Addison-Wesely Publishing Company, 1982.*

[HEBB 49] Hebb D.O. (1949): The Organization of Behavior; *New York: Wiley.*

[HINT 86] Hinton G.E., & Sejnowski T.J. (1986): Learning and Relearning in Boltzmann Machines; *in Rumelhart D.E., Parallel Distributed Processing, Vol I, pp 282–316, MIT Press, Cambridge, Massachusetts.*

[MATU 70E] Maturana H.R. (1970): Biology of Cognition; *in H.R.Maturana & F.J.Varela, Autopoiesis and Cognition; pp 2–60, D.Reidel Publishing Company, Dordrecht, Boston (1980).*

[MATU 78] Maturana H.R. (1978): Kognition; *in Schmidt, Der Diskurs des Radikalen Konstruktivismus, pp 89–118, Suhrkamp, stw 636 (1987).*

[NEWE 76] Newell A. & Simon H.A. (1976): Computer Science as Empirical Inquiry: Symbols and Search; *Communications of the ACM, March 1976, Vol. 19, Number 3, pp 113–126*

[POLA 66] Polanyi M. (1966): The Tacit Dimension (Implizites Wissen); *Doubleday & Company, Inc. (1966),Garden City, New York (Suhrkamp–Taschenbuch Wissenschaft, stw 543 (1985), Frankfurt/M.).*

MOTION DETECTION IN THE VISUAL CORTEX OF THE CAT

Stanislav Reinis and David S. Weiss

Department of Psychology
University of Waterloo
Waterloo, Ontario
N2L 3G1, Canada

ABSTRACT

The recognition of motion in the cat is accomplished by complex cells in area 18 of the cerebral cortex. The receptive fields (RFs) of the complex cells are formed by a discontinuous accumulation of subunits (active points) which respond to a moving stimulus. Active point density is greatest in the central excitatory core of the RF and least in the peripheral portions of the RF.

Although the RF cores of neighboring cells overlap, the distribution of their respective active points is complementary. A single active point records the position of a moving feature within a small segment of the RF. The same feature then stimulates an adjacent active point belonging to a neighboring cortical cell within the network. The movement of the feature is represented by the sequential activation of complementary active points and the subsequent interaction between neighboring cells.

Mass correlograms calculated from the multiple cell activity records illustrate the interaction between complementary visual cells. A regular oscillating activity within the mass correlogram is generated when a moving visual stimulus activates a common RF field. A model of neuronal interactions derived from the mass correlograms shows that the regular oscillatory pattern of the correlograms is actually caused by a rather irregular but mutually complementary firing of individual units. Mass correlograms calculated up to 1000 msec intervals show that some cells are able to follow the moving stimulus predominantly along a short pathway while other systems follow the stimulus along both long and short pathways. There is, however, a gradual transition between the two types of cellular systems.

INTRODUCTION

The informational content of brain activity depends on complex interactions involving the simultaneous generation and parallel passage of spatially distributed nerve impulses. A simultaneous recording of impulse activity from several neurons may allow an analysis of these interactions and related brain functions. In this paper, we present some new findings on the detection of motion by neuronal systems in the mammalian cerebral cortex.

The two general models of motion recognition by the visual system are the correlation model and the gradient model. Data presented in this paper indicate that the correlation model is the more probable system of motion detection.

MATERIALS and METHODS

Subjects. Adult cats of both sexes were used in these experiments. The animals were immobilized with pancuronium bromide and anesthetized with nitrous oxide. Superficial wounds were infiltrated with a long-term local anesthetic.

The Visual Stimulus. The animals were positioned in front of a transparent plexiglas screen covered with Albanene tracing paper. Visual stimuli consisted of moving light bars which were rear-projected onto the screen. The light bars were presented at a rate of 3 deg/sec, and were either 1 or 13 degrees of visual angle in length. The 1° light bar was used to plot the RFs. The 13° light bar was used for the determination of the directional preference of the cell as well as for the recording of sweeps from which the correlograms were calculated.

Electrophysiological Recording. Neuronal activity was recorded with tungsten microelectrodes insulated with epoxy resin (1-2 $M\Omega$) from area 18 of the visual cortex. Pre-amplified neuroelectric signals from the cortex were digitized and evaluated by an IBM Personal Computer. The output from the preamplifiers to the A/D converter was adjusted so that the amplitude of the spikes produced by the cell nearest to the recording electrode was 1000 mV. This cell was designated the leading cell of the network. The lower limit of the leading cell amplitude was 875 mV.

In several experiments we were able to distinguish and record from a second leading cell with an amplitude between 750 and 875 mV. Neuronal activity was also recorded from a group of units which surrounded the leading cell(s). This background or mass activity was registered in a voltage window between 375 and 500 mV. From the ratio of leading cell to mass activity spikes we estimated that our microelectrodes were recording potentials from 10 to 20 visual cells simultaneously.

Receptive Field Plots. The internal structure of each RF was determined within a rectangle 15 by 25 degrees of visual angle. The rectangle was centered on the tracing paper around the approximate location of each leading cell's RF. The short side of the rectangle, parallel to the long axis of the RF, was divided into 15 1° bands. The 1° light bar stimulus was presented along each band fifty times. The number of spikes which accumulated over the 50 stimulus presentations were displayed as short vertical bars in the exact positions where the spikes were originally detected. RFs were determined for the leading cell(s) as well as for the mass activity.

Leading Cell and Mass Activity Correlograms. Multiunit records for the calculation of correlograms were elicited by the 13° stimulus. The stimulus was presented in eight directions beginning with the optimal angle of directional preference. Neuronal responses were also recorded in the absence of visual stimulation. As with the determination of the RFs, correlograms were calculated for one or two individual leading cells and the mass activity. The position of each spike in the multiunit record was determined to the nearest millisecond. Significant neuronal interactions between and within each level were evaluated from the histograms of all interspike intervals up to 1000 msec (auto- and crosscorrelograms). The histograms describing the mass activity were called mass correlograms.

Functional models of the complex neuronal interactions within the cellular systems were constructed using the correlograms. The shapes of the mass correlograms indicated that some interspike intervals appeared more often than others. As shown in Figure 1, the most common statistically significant interspike intervals were attributed to pairs of real spikes in the original record of neuronal activity. Through such spikes, often common to two or more pairs, a system of neuronal interactions was assembled which showed how the neurons within the vicinity of the microelectrode responded to a moving visual stimulus.

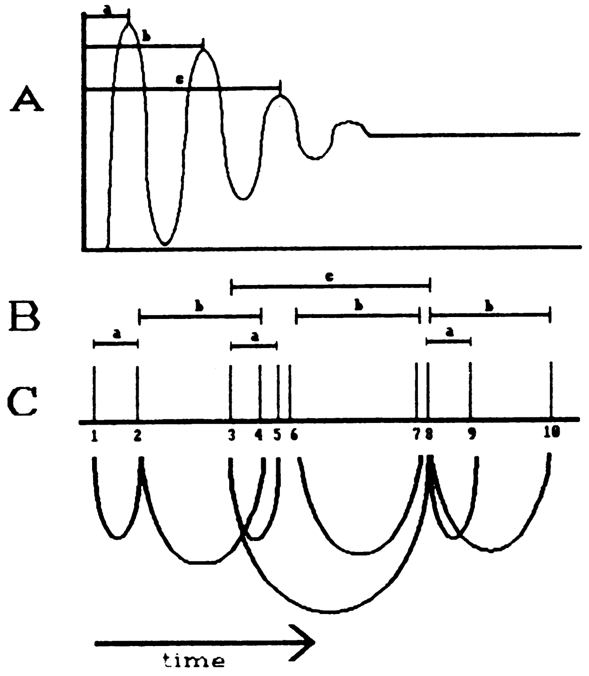


Figure 1. The modelling of complex neuronal interactions in the visual cortex. As shown in this scheme, **A** is the mass correlogram calculated from **C** the hypothetical record of multiple spiking. ***a, b,*** and ***c*** represent the most common interspike intervals which accumulated in the peaks of the mass correlogram. The most common interspike intervals ***(a,b,c)*** are attributed in row **B** to the pairs of real spikes as found in **C** the original record of neuronal activity.

Each of the spikes may, at the same time, be a component of another pair of spikes with a different time interval. Through such spikes, which are common to two (or more) pairs, a system of neuronal interactions may be assembled which shows how the neurons in the vicinity of one recording electrode interact.

RESULTS

Internal Structure of the Receptive Fields. The internal structure of the RFs was not homogenous. All visually driven cells fired repeatedly only when the stimulus was present at certain locations within the RF. These loci were called active points or active subunits. The cells did not respond when the visual stimulus passed outside of the active points.

The RF of a leading cell was composed of a central core region and a peripheral region. The density of the active points was higher in the RF core than it was in the RF periphery. However, active point density was quite variable. In some cells, the subunits were densely packed while in other cells the RFs were formed by a smaller number of loosely dispersed active points.

The difference between the central and peripheral RF regions depended not only on the density of the active points but also on the number of spikes which accumulated in the active points during the 50 stimulus presentations. In all cases, the number of spikes in each active point was greater inside the core than outside the core. The average outside to inside proportion of spike accumulation was 76% with a strong linear relationship between the variables (r=0.911, n=48). A strong positive correlation was also found between the density of active points inside and outside the RF. The best-fit regression equation describing this relationship showed that the relation had an exponential character. The shapes and areas of the RFs of two neighboring cells were very similar, with the exception of the fact that the precise locations of the active points in two neighboring cells did not coincide.

Although the areas and shapes of the RF cores of the leading cells did not differ substantially from their associated mass RFs, the density of active points both inside and outside of the RFs was much higher in the mass fields. The number of spikes which accumulated in the active points was also higher in mass fields than in individual fields.

This increase in the number of active points obviously reflected the increased number of cells collectively producing the mass RF. Also, active point density was very high in the periphery of the mass RF. Out of the total 35 systems studied, only 23 showed a clearly recognizable mass RF core. In the remaining 12 systems, subunit density as well as the number of spikes outside the core were so high that the cores could not be distinguished from the periphery.

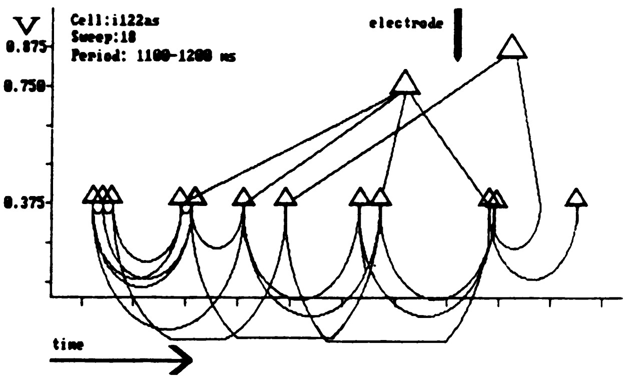


Figure 2. A real system of interactions derived from a single sweep across the receptive field of a visually driven cortical cell. Only a small segment of the complete system of interactions is shown. The most common interspike intervals (exceeding 5% confidence limit of the correlogram) were taken from the mass correlograms as well as from the cross-correlograms between the mass activity and two leading cells which differ in their amplitude and shape of the spike.

A group of interacting cells in area 18 of the cerebral cortex registers the movement of a bar of light across a common receptive field. Each of these cells detects the position of the bar at a number of points within the receptive field, and subsequently interacts with other cells which record the position of the same bar at different sets of active points.

Mass Correlograms. The most important finding with respect to the early phase of the mass correlograms was the existence of a very consistent 3 msec oscillatory activity among the local cell population. When activated by an optimally directed visual stimulus, this rhythmic activity acquired the character of a very regular, smooth sinusoid with some damping. When, on the other hand, the system of neurons around the recording electrode was not activated or when it was activated by a non-optimal stimulus, the 3 msec pattern of oscillatory activity became irregular and variable.

When we attribute the most common interspike intervals (exceeding 5% confidence limit of the correlogram) to real pairs of spikes in the original record, the resulting model indicates that the firing of individual units was rather irregular, and the rhythmic behavior displayed by the mass of cells was not accompanied by any apparent rhythmic firing of individual units (Figure 2).

Mass correlograms computed up to 1000 msec exhibited positive or negative slopes. An upward-going (positive) slope, increasing from the shortest to the longest interspike interval, suggested that longer intervals predominated in the record. A downward-going (negative) slope indicated the predominance of shorter interspike intervals.

However, the correlograms could not be divided into two sharply dichotomized classes because of the wide variability in slope characteristics. When the slopes were regressed against other RF attributes, the relationship between the slopes of the mass correlograms and the total number of spikes accumulating in the peripheral active points was of most interest. This relationship displayed a clear logarithmic character.

DISCUSSION

The data presented in this paper may be summarized as follows:

1) The RFs in area 18 of the cat's cerebral cortex are formed by subunits in which the cells respond to a moving stimulus with a certain regularity.
2) The response of the subunits to visual stimulation is quite variable. Only some of the active points are activated by a moving stimulus with 100% probability.
3) Although the RFs of neighboring visual cells overlap, the actual distribution of their active points appear to be complementary. Thus each segment of the visual field appears to be covered by a set of interacting cortical cells.

The analysis of a segment of the visual image is accomplished by several cells simultaneously. Each cell scans very narrow, scattered portions of the visual field for the presence of a particular feature. The position of a moving stimulus is detected by active points belonging to different cells. This successive detection may result in the perception and analysis of movement.

A minimum of two cells might be sufficient for the detection of the movement and velocity of a feature. In the real brain, however, many cells are involved in the simultaneous detection of complex moving images. This scheme suggests that overlapping of the RFs of neighboring visual cells is not a sign of redundancy in the cerebral cortex but rather a necessary arrangement for the detection of motion.

The rather irregular internal structure of the observed RFs indicates that the most appropriate tokens activating these cells are not straight lines (as there is no linear arrangement of the active points) or spatial frequencies (as the active points are not arranged in bands) but, more probably, irregular surface textures of natural moving objects.

Our data allowed us to distinguish between short- and long-range motion detection (systems with either negative or positive slopes), but the distinction between these systems was not very precise. All long-range detectors had clearly visible RF excitatory cores and the number of short interspike intervals in such systems was usually high as well. Similarly, all short-range systems had active points in the periphery of their RFs and were therefore able to detect long-range movement as well. This finding implies that the long- and short-range detectors belong into the same class of cells, although they differ in their quantitative characteristics.

An open question still remains as to the morphological and physiological character of the active points. Functionally, it appears that the active points have a phasic response, reacting best to the change of light intensity. Since no cells responding to motion detectors have been reported in the lateral geniculate body of the cat, it is possible that the active points are actually a product of the mutual interaction of inputs from individual ganglion cells of the retina accomplished mainly at the level of visual cortex.

The functional interactions within a group of neurons may be made evident by studying the auto- and mass correlograms. The study of the early portion of the mass correlograms shows that neighboring visual neurons respond to motion by a regular, almost sinusoidal rhythm. This rhythm, however, is apparent only in the correlogram and not in the original record of the multiple neuronal firings. The reconstruction of the functional interactions between neighboring neurons involved in the analysis of motion showed that individual neurons participate in this process in a rather irregular manner, only the system as a whole reacted in a rhythmic way. It is as if a number of drummers decided to produce a complex rhythm by each contributing a single beat.

The described mechanisms attributed to complex cells in area 18 of the cortex may, in some way, provide information to various motion-in-depth receptors, looming detectors and other proposed complex analytical systems, and together with them, they may provide a versatile and robust motion analysis system.

Using Neural Networks and Genetic Algorithms as Heuristics for NP-Complete Problems

William M. Spears
Naval Research Laboratory, Code 5510
Washington, D.C. 20375-5000
and
Kenneth A. De Jong
George Mason University
Fairfax, VA 22030

ABSTRACT

Paradigms for using neural networks (NNs) and genetic algorithms (GAs) to heuristically solve boolean satisfiability (SAT) problems are presented. Since SAT is NP-Complete, any other NP-Complete problem can be transformed into an equivalent SAT problem in polynomial time, and solved via either paradigm. This technique is illustrated for hamiltonian circuit (HC) problems.

INTRODUCTION

NP-Complete problems are problems that are not currently solvable in polynomial time. However, they are polynomially equivalent in the sense that any NP-Complete problem can be transformed into any other in polynomial time. Thus, if any NP-Complete problem can be solved in polynomial time, they all can [Garey]. The canonical example of an NP-Complete problem is the boolean satisfiability (SAT) problem: Given an arbitrary boolean expression of *n* variables, does there exist an assignment to those variables such that the expression is true? Other familiar examples include job shop scheduling, bin packing, and traveling salesman (TSP) problems.

GAs and NNs have been used as heuristics for some NP-Complete problems. Unfortunately, the results have been mixed because NP-Complete problems are *not* equivalent with respect to how well they map onto NN (or GA) representations. The TSP is a classic example of a problem that does not map naturally to either NNs [Gutzmann] or GAs [De Jong].

This observation suggests the following intriguing technique. Suppose we are able to identify an NP-complete problem that has an effective representation in the methodology of interest (GAs or NNs) and develop an efficient problem solver for that particular case. Other NP-complete problems which do not have effective representations can then be solved by transforming them into the canonical problem, solving it, and transforming the solution back to the original one.

This paper outlines GA and NN paradigms that solve SAT problems, and uses hamiltonian circuit (HC) problems to illustrate how either paradigm can be used to solve other NP-Complete problems after they are transformed into equivalent SAT problems. The remainder of the paper is divided into four sections. The first section discusses the GA paradigm. The second section discusses the NN paradigm. The third section provides some experimental results and discusses the technique of solving HC problems using both paradigms after polynomial transformation into equivalent SAT problems. The final section summarizes the paper.

GENETIC ALGORITHMS

GAs consist of a population of individuals competing on a survival-of-the-fittest basis in an environment. The algorithm proceeds in steps called generations. During each generation, a new population of individuals is created from the old via application of genetic operators (crossover, mutation, etc.), and evaluated as solutions to a given problem (the environment). Due to selective pressure, the population adapts to the environment over succeeding generations, evolving better solutions [Goldberg].

If the environment is a function, GAs can be used for function optimization. In this case, each individual in a population is a sample point in the function space. Classically, an individual in a GA is represented as a bit string of some length *n*. Each individual thus represents one sample point in a space of size 2^n.

Any application of GAs involves a selection of an appropriate representation of sample points in the function space, and the creation of a function that describes the behavior of the space to be searched. Unfortunately, many NP-Complete problems have constrained spaces that do not map well to bit string representations. The TSP is a classic example of such a problem. However, a SAT problem consists of a search over n boolean variables, resulting in a solution space of size 2^n. Suppose we denote *true* by 1 and *false* by 0. Then each bit in a bit string represents the truth value of one boolean variable. In summary, the bit string representation is natural for SAT.

Given the representation, the problem is to create an evaluation function that adequately describes each point in a SAT search space. For example, suppose we simply create a function that returns 1 when the expression is satisfied, and 0 when it is not. Although the function is logically correct, the solution space that results is simply a plateau with spikes. Any search algorithm degenerates to random search.

GAs derive their power from differential feedback. Non-solutions that are *better* than other non-solutions should have higher function values (if we are maximizing). For SAT, variable assignments that nearly satisfy the boolean expression should have higher function values than those assignments that barely satisfy the expression. This is achieved by basing the SAT function on the parse tree of the boolean expression. The assignment to the boolean variables is done at the leaf nodes. The function value at the root node is determined recursively as follows.

The value of each node in the parse tree is based on the children of that node. If the node is a *NOT*, its value is *opposite* that of its child (ie., 1 – *value*). If the node is an *OR*, its value is the *maximum* of the children. Finally, the value of an *AND* node is the *average* of the children. This treatment of *AND* provides differential feedback that rewards *better* non-solutions. The result is a function space that is smoothed in the sense that progressively better non-solutions receive higher function values.

Unfortunately, this mathematical treatment is not truth invariant under boolean transformation. In particular, it is possible to derive anomalous situations in which a solution receives a function value less than 1. However, the addition of a simple preprocessing step removes all anomalies. This preprocessing step consists of applying De Morgan's theorem to the boolean expression, pushing the *NOT* nodes to the bottom of the parse tree. The preprocessing step is linear in complexity.

In summary, we have outlined an effective GA representation for SAT problems. The individual bit string naturally represents the 2^n possible assignments to the boolean variables. The evaluation function based on the parse tree reflects the structure of the SAT problem and has the following properties: 1) it assigns a payoff value of 1 if and only if the assignment satisfies the boolean expression; 2) it assigns values in the range $0 \leq value < 1$ to all non-solutions; and 3) non-solutions receive differential feedback on the basis of how near their *AND* clauses are to being satisfied [De Jong].

A later section provides experimental results of applying the GA to various SAT problems. The next section indicates how SAT parse trees can also be used in a constraint satisfaction, neural network paradigm.

NEURAL NETWORKS

Any application of neural networks involves selection of an appropriate network representation. Furthermore, a constraint satisfaction approach requires a specification of the domain specific constraints. These constraints must be mapped into an energy function that adequately describes the space to be searched [McClelland]. In general, these tasks can be difficult.

For the specific problem at hand (SAT), however, our previous work in GAs gives us some surprising insights. First, the parse tree used for the GA SAT function describes a natural network representation that is perfectly matched to the structure of the boolean expression. Second, since each node in the network is bound by boolean constraints, an energy function can be created that fully describes the space of constraints.

In a manner similar to Hopfield nets, we let the activations of the nodes be binary [Hopfield]. The activation of each node, then, represents the hypothesis that a particular subexpression is true. We denote *true* as 1 and *false* as -1. The fixed weight on each edge represents a boolean constraint between two hypotheses. If the weight is +1, the boolean constraint is satisfied if both nodes are in the same state. If the weight is -1, the constraint is satisfied if both nodes are in opposite states.

There are several differences between the proposed network and a Hopfield net. First, since this network is based on a parse tree of a boolean expression it contains *AND*, *OR*, and *NOT* nodes. The nodes in a Hopfield net are of one type. Second, the network is directed, with the output node being the root. Each *AND*, *OR*, and *NOT* node has parents and/or children (not just *neighbors*). In a Hopfield net, the links are bi-directional and symmetric.

The asymmetries in the proposed network can be explained by a closer analysis of the constraints inherent in a boolean network. Each node can possibly be influenced by *upstream* constraints, *downstream* constraints, and *bias*.

Upstream constraints represent constraints from nodes that are closer to the root, while downstream constraints represent those constraints from nodes further from the root. Bias provides further externally defined constraints.

Downstream constraints flow from the children of a node. Suppose that some node is a *NOT* node. Then its activation should be opposite that of its child. An *AND* node should be true if all of its children are true. An *OR* node should be true if any of its children is true.

Upstream constraints flow from the parent of a node. Suppose the parent of a node is a *NOT*. Then the activation of the node should be opposite that of its parent. If the parent is an *AND* and it is true, then the node should be true. However, if the parent is an *AND* and it is false, then the node should be false if all siblings are true. Other situations are possible, but they do not constrain the node. The situation with *OR* is symmetric.

Note from the above that there are two types of constraint implied. In the *NOT* example, nodes are constrained to be different. In the *AND* and *OR* examples, nodes are constrained to be similar. In other words, a connection weight of +1 enforces the idea that two nodes are both true or both false. A connection weight of -1 enforces the idea that both nodes are not the same. Note that this latter situation occurs only with *NOT* nodes. In fact, a *NOT* node and a negative connection are equivalent, so only *AND*, *OR*, and input nodes are necessary in this paradigm.

Bias is used to provide more constraint information. For example, since satisfiability is the goal, the root node must have an activation of 1. During the parse of the boolean expression, it is sometimes also possible to determine the activations of subexpressions. If an activation must be 1, the node receives a high bias. If an activation must be -1, the node receives a low bias. This information, coupled with the above upstream and downstream constraints, constitutes the maximum information easily derivable from the boolean expression.

Given the set of logical constraints, an energy function can be derived. For SAT, the local energy for each node is expressed:

$$Energy_i = net_i \, a_i$$

$$net_i = Unet_i + Dnet_i + Bias_i$$

The energy of each node ($Energy_i$) is the product of the net input (net_i) and its activation (a_i). The activation is updated probabilistically using the Boltzmann distribution and an exponentially decaying annealing schedule is used to help avoid local optima. The net input is based on the upstream ($Unet_i$), downstream ($Dnet_i$), and bias ($Bias_i$) constraints. The resulting energy function guarantees that solutions have a predictable maximum energy, while non-solutions have lower energy.

In summary, the parse tree network appears to be a natural NN representation when used in conjunction with boolean constraints to define a proper energy function. The next section describes the results of applying the GA and NN paradigms to some simple SAT problems and some polynomially transformed HC problems.

EXPERIMENTS AND RESULTS

Both paradigms were initially run on two families of boolean expressions with comparison results based on the number of evaluations needed to solve the problem. For the GA, an evaluation corresponds to the function evaluation of one individual in the population. For the NN, one evaluation corresponds to updating the activation of each node in the parse tree exactly once. Since both evaluations use the parse tree, they have equivalent complexity.

The first family selected consists of two-peak expressions of the form:

$$(AND\ X_1\ \cdots\ X_n)\ OR\ (AND\ \overline{X_1}\ \cdots\ \overline{X_n})$$

which have exactly two solutions (all *false* and all *true*). The following table indicates the number of evaluations needed for each paradigm. All results are averaged over 10 trials. The number of variables is *n*.

n	10	20	30	40	50	60	70	80	90
GA	164	696	1257	2283	2741	4060	4966	6973	10208
NN	6	13	19	40	45	61	84	101	110

To make things a bit more difficult, we changed the problem slightly by turning one of the solutions into a false-peak as follows:

$$(AND\ X_1\ \cdots\ X_n)\ OR\ (AND\ X_1\ \overline{X_1}\ \cdots\ \overline{X_n})$$

so that the previous all *false* solution is now almost correct and the only correct solution is that of all *true*. The results are similar to the previous problem.

So far, the NN paradigm clearly outperforms the GA paradigm on the simple two-peak and false-peak problems. To generate more realistic and difficult boolean expressions and to illustrate the technique of solving other NP-Complete problems by mapping them onto SAT problems, we defined a set of hamiltonian circuit (HC) problems of increasing complexity.

Each problem consists of a ring of N nodes labeled alphabetically. Each node has a directed edge to all nodes ahead in the alphabet. There are roughly $N^2 / 2$ edges. The resulting problem has only one solution (a circuit around the ring), with a large number of partial solutions. These problems are easily transformed into equivalent SAT problems in which each variable in the equivalent boolean expression corresponds to one edge in the HC graph. If the variable is *true*, that edge is in the circuit. The expression is satisfied if the assignment to the edges is a hamiltonian circuit. The following table outlines our initial results.

n	6	10	15	21	28	36	45	55
GA	106	239	803	3559	8680	34417	174706	721525
NN	16	42	202	522	2281	20087	160981	1391601

These results suggest an intriguing hypothesis for further study: that the NN paradigm is better on smaller and simpler problems, while GAs have better scaling-up properties and are more effective on the larger and more complex problems.

SUMMARY

This paper presents NN and GA paradigms for heuristically solving SAT problems. Other NP-complete problems can be solved via polynomial-time transformation into equivalent SAT problems. This technique is illustrated for HC problems. Preliminary experiments suggest that while both paradigms are effective, the NN paradigm may be better for smaller problems and GAs more effective on larger ones.

Future work will explore further the limitations of these paradigms by defining even more difficult classes of SAT problems derived from other NP-Complete problems. We also plan to explore the possibility of merging the two paradigms, using the GA for global search and the NN as a local optimizer.

Acknowledgements

I would like to thank Diana Gordon for her valuable comments and suggestions.

References

De Jong, K. A. & William M. Spears (1989). Using Genetic Algorithms to Solve NP-Complete Problems, *Proc. Int'l Conference on Genetic Algorithms and their Applications.*

Garey, Michael R. & David S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W. H. Freeman and Company, San Francisco, CA.

Goldberg, David E. (1989). *Genetic Algorithms in Search, Optimization & Machine Learning,* Addison-Wesley Publishing Company, Inc.

Gutzmann, Kurt M. (1987). Combinatorial Optimization Using a Continuous State Boltzmann Machine, *IEEE First Int'l Conference on Neural Networks,* III-721.

Hopfield, J. J. & D. W. Tank (1985). Neural computation of decisions in optimization problems, *Biological Cybernetics,* 52, 141-152.

McClelland, James L. & David E. Rumelhart (1988). *Explorations in Parallel Distributed Processing,* The MIT Press, Cambridge, MA.

On the Assignment-of-Credit Problem in Operant Learning

J. E. R. Staddon and Y. Zhang
Department of Psychology, Duke University, Durham, NC 27706, USA

The way that operant reinforcement selects one activity over others -- the *assignment-of-credit* problem -- is taken for granted in most learning models which are examples of *supervised* learning, in which a "teacher" (reinforcement) is assumed to present explicit error information to the learning system in a way that has no obvious biological parallel. Assignment of credit is solved for most learning theorists by the unformalized process of temporal contiguity. Unfortunately, contiguity theory is violated by phenomena such as "superstitious" behavior and instinctive drift, which are anomalous either because the activity contiguous with reinforcement is not the one actually strengthened or because activities continue to occur despite response-independent reinforcement. These anomalies have never been reconciled with contiguity theory. We show that the simplest possible contiguity model for the assignment-of-credit problem in operant conditioning also provides a natural explanation for supposed exceptions.

Standard effects. Positive reinforcement is defined by its *selective* effect: when the occurrence of a reinforcer is made to depend on the occurrence of a particular response (out of two or more possibilities), the response probability should increase, and when the reinforcer is no longer presented, or is presented independently of responding, response probability should decline. In addition to this property of *selection*, reinforcement is also sensitive to *delay*, and *contingency*.

Our model is based on assumptions that are already well accepted, even though their effects in combination have not been fully understood: arousal and adaptation, the idea that reinforcement transiently energizes a range of activities; strength and competition, the idea that each activity has a certain tendency to occur and that the strongest will win; and variability, the notion of a repertoire of activities. These five properties are captured in two linear discrete-time equations: We define for each behavior a variable, V_i, its *strength*. The competition rule is winner-take-all: the activity with highest strength is the one that occurs (this is the only nonlinearity). The equations describe the changes in V values from one discrete-time instant to the next in the absence of reinforcement, or following reinforcement:

After nonreinforcement: $V_i(t+1) = a_iV_i(t) + \epsilon(1-a_i), \quad 0 \le a_i \le 1,$ (1)

After reinforcement: $V_i(t+1) = a_iV_i(t) + \epsilon(1-a_i) + b_iV_i(t),$ (2)

where $V_i(t)$ is the strength of the ith activity in discrete-time instant t, a_i and b_i are parameters that depend on *both* the activity *and* the reinforcer and ϵ is a random variable sampled independently for each activity in each time instant. Term $a_iV_i(t)$ represents adaptation or short-term memory (STM): because $a_i < 1$, this term reduces to zero with repeated iterations. Term $b_iV_i(t)$ represents the arousal effect of a hedonic event, which we assume acts on *all* activities. If $b_i > 0$, the effect is to increase V_i (a positive reinforcer); if $b_i < 0$, the effect is to reduce V_i (a punisher). Note that the relation $a_i + b_i < 1$ must hold if V_i is not to rise without limit in repeated iterations of Equation 2.

Consider first the case of two or more identical activities (i.e., $a_1 = a_2$; $b_1 = b_2$), which permits derivation of all the standard reinforcement properties. In the absence of reinforcement, because the two parameters are the same for both activities, each activity will occur equally often, on average. If positive reinforcement is delivered for each occurrence of Activity 1, then at that instant by the highest-wins competition rule $V_1 > V_2$, hence the increment to 1, bV_1, must be greater than the increment to 2, bV_2. If the reinforcement occurs frequently enough that the increment in V_1 does not decay to zero by the next reinforcement V_1 will be steadily incremented relative to V_2, so that Activity 1 will come to dominate. This conclusion holds whichever activity is reinforced; thus the process satisfies the *selection* condition.

The essential feature of the reinforcement mechanism is that reinforcement always adds *some* increment to *all* activities, but the *largest* increment goes to the highest-V activity.

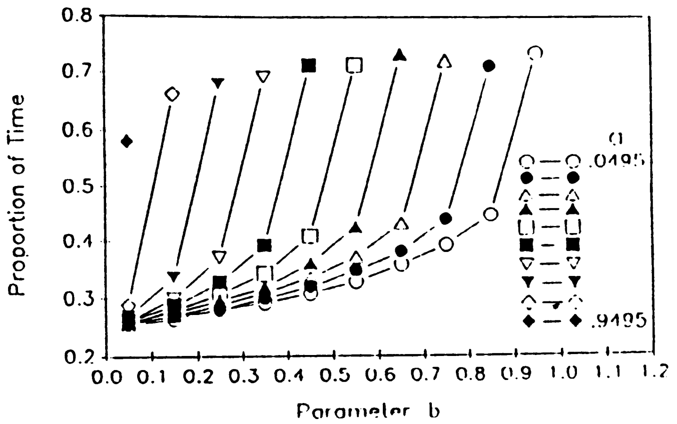


Figure 1 Parameter Space. Simulation of the effects of 100% reinforcement on one of four identical (same a_i and b_i) activities for a range of a and b values (0.0495-0.9495 in increments of 0.1). Ordinate shows the proportion of time (iterations) taken up by the reinforced activity. Note that in the absence of any reinforcing effect each activity should take up about 25% of the time. Each point in this and the next two figures is the average of $2x10^5$ iterations. The random variable ϵ in eqs. 1 & 2 had a rectangular distribution over the interval 0 - 1.

Figure 1 shows the asymptotic effects of reinforcing every occurrence of one behavior, in a set of four, for a wide range of parameter pairs. The reinforced behavior is always facilitated, and the proportion of time taken up increases with increases in either parameter. We have obtained similar results for ensembles of 2, 4 and 8 identical activities.

When reinforcement is delayed, the same increasing pattern is seen, but now the STM parameter, *a*, has the greatest effect: The higher the *a* value, the less likely that the delayed reinforcer, when it actually occurs, will strengthen a behavior other than the target behavior.

Contingency is the fact that the strengthening effect of reinforcement depends on its correlation with the reinforced behavior, not just on contiguity. Thus, if the target behavior is reinforced intermittently, or if it is reinforced every time but reinforcement also occurs at other times, the behavior will be strengthened less than if it is reinforced exclusively. Our model has both these properties: the higher the probability of reinforcement, the larger the proportion of time taken up by the reinforced activity. Conversely, the higher the probability of "free" reinforcers, the lower the level of the explicitly reinforced act.

Anomalous effects. Most anomalies arise when activities have *different* parameter values. We discuss three: superstitious behavior, differential conditionability, represented by typologies, such as the distinction between emitted and elicited behavior, and instinctive drift.

"Superstitious" behavior is typically produced when an animal such as a pigeon is given periodic response-independent food reinforcement. If food delivery is frequent enough, the animal develops various kinds of vigorous stereotypies, despite the fact that food delivery is independent of its behavior. Our model is too simple to account for the associative and temporal properties of this phenomenon, but the model can under restricted conditions produce apparently stereotyped behavior, even with a set of identical activities. The stereotypy is the outcome of a positive-feedback process that resembles a suggestion of Skinner: when a (response-independent) reinforcer is delivered some behavior will be occurring and will be automatically strengthened; if the next reinforcer follows soon enough, the same behavior will still be occurring and will be further strengthened, and so on, until the behavior appears to dominate. More detailed examination shows that our version of Skinner's process is unlikely to be responsible for superstitious behavior, although it does account for the reliable finding of *hysteresis* in operant conditioning.

The distinction between *emitted* and *elicited* behavior parallels the procedural difference between operant and classical conditioning: elicited behavior (salivation is an example) is behavior elicited by a reinforcer but not modifiable by operant conditioning; emitted behavior (lever pressing is an example) is not usually elicited by the reinforcer, but is modifiable by operant conditioning (the rat learns to press the lever to get food). The existence of a typology is usually a sign that we lack understanding of the underlying process. It is interesting, therefore, that the distinction between these two types of behavior emerges in a natural way from our model as a consequence of the complementary relation between the arousal and STM parameters that is forced by stability considerations.

Instinctive drift is perhaps the most striking exception to a contiguity account of reinforcement. Breland & Breland reported several instances that conform to the following pattern: Behavior A (e.g., a raccoon putting a wooden egg onto a chute) is successfully "shaped" by response-contingent food reinforcement; but after a while Behavior A is supplanted by Behavior B ("washing" the egg between the animal's forepaws), which is part of the animal's natural foraging repertoire. Behavior B is inappropriate in this context because it competes with Behavior A on which food delivery actually depends. Behavior A is contiguous with the reinforcer, but Behavior B is ultimately the one strengthened.

A variety of naturalistic interpretations have been offered for these effects (and for the related phenomenon of "superstition"), but they follow from our model on the assumption that "instinctive" behaviors are characterized by very high *a* values (STM persistence). Given a set of activities with moderate *a* and *b* values, and one activity with a very high *a* value (and nonzero *b*), reinforcement of one of the moderate-*a* activities will cause it to predominate initially (because it has a higher *b* value than the "instinctive" activity). But, because increments to the high-*a* activity cumulate more effectively than the (larger) increments to the reinforced activity, it may predominate eventually, even if it is never contiguous with reinforcement.

These effects are illustrated in the cumulative records in the top panel of Fig. 2. Each set of four records shows four activities, three with moderate *a* and *b* values, one with a high *a* value and low *b* value. The Left records show the free-operant (unreinforced) levels of the four activities (the low frequency of the high-*a* activity is a consequence of statistical properties of the model not explained here). The Center and Right panels show the effect of reinforcing one of the low-*a* activities with a probability of 0.25 or 1.00. The increasing reinforcement probability has two effects: it

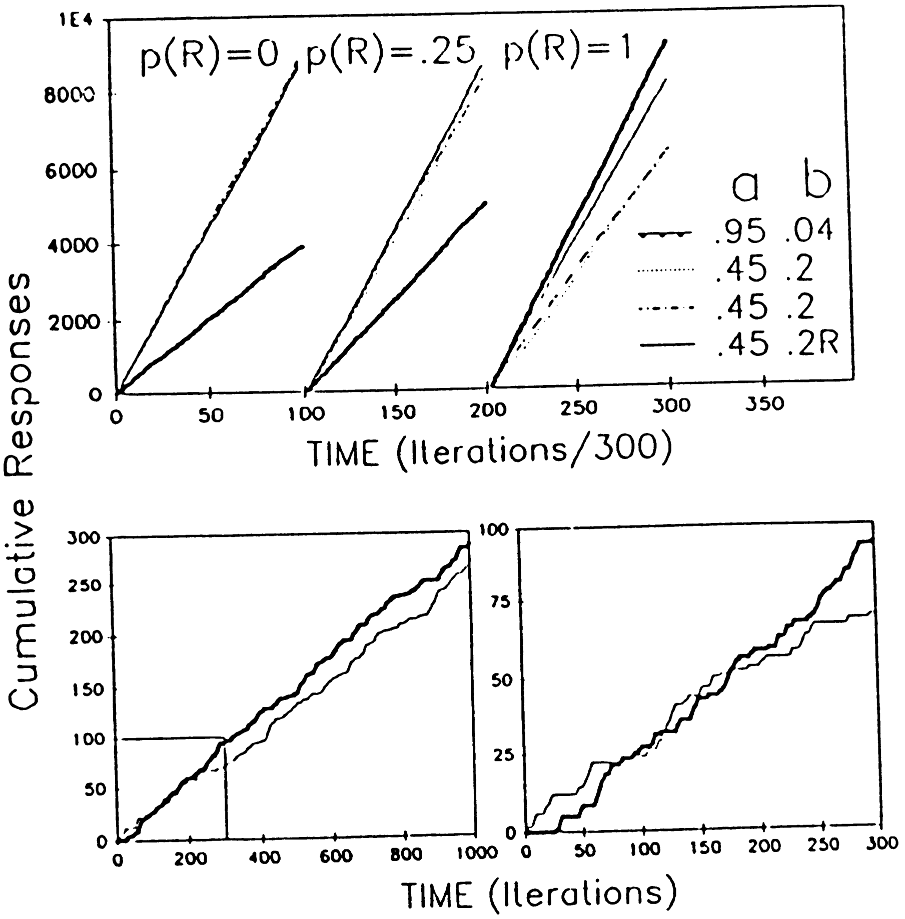


Figure 2 Instinctive drift. Top panel, Left: Cumulative records of four activities in the absence of reinforcement (operant levels): high-a, low-*b* activity is the least frequent (see parameter values in the table). Center: The effect of reinforcing one of the three intermediate-a activities (a = 0.45, b = 0.2R) with probability 0.25. Right: Effect of reinforcing every occurrence of the intermediate activity. Bottom panel: Magnified pictures of the beginning of the record in the Right panel at the top ("instinctive" and reinforced activities only).

causes the level of the reinforced activity ("R") to increase above the level of the other two low-*a* activities; but it causes a disproportionate increase in level of the high-*a* activity, which is predominant when p(R) = 1 (Right). The two bottom panels in Fig. 5 show a magnified picture of initial acquisition in the p(R) = 1 condition, illustrating the transition from dominance by the reinforced, low-*b*, activity to predominance of the "instinctive", high-*a*, activity. If a low *a* value activity is reinforced, it may predominate initially, but will be supplanted by the high-*a* value activity, even if it has a lower "arousability" (*b* value).

Conclusion Both standard and anomalous properties of operant conditioning with positive and aversive reinforcers are consistent with a simple non-associative assignment-of-credit mechanism. Since the operant behavior of vertebrates is always context dependent, our model is obviously only part of the whole story, so it is inappropriate to look for quantitative agreement between facts and predictions. The model also does not deal with anything that depends on long-term memory, such as extinction and reconditioning, stimulus effects, or timing processes. Because the model does not pretend to encompass the whole process of conditioning, it is hard to know what to make of a number of partial exceptions (e.g., the pecking response is both elicited and operantly conditionable, intermittent reinforcement sometimes sustains more behavior than continuous, etc.) -- since they may reflect later stages in the process. Such a process may therefore form the "front end" for the complex neural processing involved in the operant and classical conditioning of higher animals.

Self-Organization of a Linear Multilayered Feedforward Neural Network

R. Stotzka, R. Männer
Physics Institute, University of Heidelberg, Philosophenweg 12, D-6900 Heidelberg, West Germany, (email: stotzka@dhdphy5.bitnet)

Introduction: Hubel and Wiesel /1,2/ analyzed the performance of the functional architecture of the mammalian visual system during the last 30 years. In the layers of the retina and the visual cortex they found many different specialized cells, which are capable to process and extract visual information. These cells perform Laplace filtering for edge detection (ON- or OFF-center neurons) and orientation sensitiveness. Supposing, these structures are able to develop self-organized in the prenatal phase of a mammalian, one has to find an algorithm which describes the formation of layered feature detecting cells. Linsker /3,4/ proposed 1986 a perceptual artificial neural network as a simple model of the layered structure of the visual information processing system. His intention was to show the self-organizing development of special cells in a linear multilayered feedforward neural network without the presentation of special information. We simulated the Linsker model using a reduced, less computational demanding Hopfield-like model /5/. For that model we analyzed the effects of different parameter settings with respect to the outcoming structures.

The Model: A Linsker-type neural network consists of several two-dimensional layers. The neurons in each layer are placed on a regular grid (Fig. 1). Layer 1 is the input layer and represents the receptor cells. The cells in the other layers receive their input from their neighborhood of cells of the previous layer.

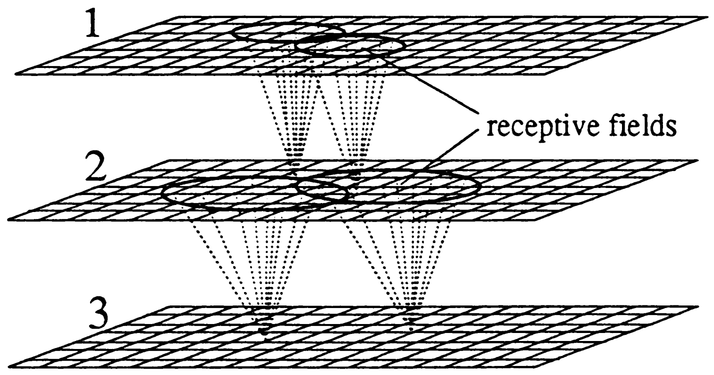


Figure 1: The first three layers of the network are shown. In each layer the neurons are placed on a regular grid. The synapses and the receptive fields of two neurons of layer 2 and 3 are indicated. Layer 1 receives random noise inputs.

There exist only feedforward synaptic connections between adjacent layers and no lateral connections within one layer. Each cell of a layer has N synaptic bonds, which are randomly distributed according to a density distribution, e.g. Gaussian, within a neighborhood (with radius r) of the opposite cell in the previous layer. The size of this receptive field grows from layer to layer. This can be described by r_L/r_M, where M denotes the actual layer and L the layer above. The dynamics of the network can be formulated as follows: the inputs of the cells in layer 1 are random noise signals. At each time step, new uncorrelated signals are presented. A neuron of layer M calculates his activity according to the linear update rule:

$$A^M = a_1 + a_0 \sum_{j=1}^{N} c_j A_j^L \qquad (1)$$

A^M is the activity of the neuron in layer M, A^{L_j} the activity of a neuron L_j of layer L, c_j the synaptic bond from M to L_j, a_0 and a_1 are constants, which are the same for all cells of layer M. The initial values of the synapses c_i, $i=1,2,\ldots,N$, are randomly chosen. In each time step, a synapse i changes its weight according to the Hebb-type learning rule:

$$\Delta c_i = \Theta + b\,(A^M - M_0)(A_i^L - L_0) \tag{2}$$

Θ, M_0, L_0 and $b>0$ are constants. If $b<<1$, it is possible to average over some time steps. In this case, the gradient dc_i/dt can be calculated in analogy to the learning rule (2). The resulting differential equation describes the maturation of a cell:

$$\frac{dc_i}{dt} = k_1 + \sum_{j=1}^{N} (Q_{ij} + k_2)\, c_j \tag{3}$$

Q_{ij} is the covariance of the activity of neurons i and j in the previous layer, k_1 and k_2 are constants dependent on the constants in equations (1) and (2). Synaptic bonds changed by Hebbian learning rules normally increase or decrease without limit. We therefore adopt a measure function, which limits the synaptic values at -0.5 and 0.5. The cell is stable, if the synapses do not change any longer under the dynamics of the system.

Analysis: We are interested which kinds of cells, characterized by the topology of the synapses, develop under different conditions. If the inputs in layer 1 are uncorrelated, the differential equation (3) of synapse i in layer 2 is independent from other synapses $j \neq i$, $j=1,2,\ldots,N$. For $|k_1|>0.5$ and the variance of the input noise $Q_{ii}>-k_2$, all synapses develop homogeneously to 0.5. Now it can be shown that the covariance function, which depends on the distance between neurons i and j in layer 1 and on r_2/r_1, is Gaussian distributed for $N \to \infty$. To interpret the possible effects during the maturation in the following layers, we define an energy function, $\partial E/\partial c_i = -dc_i/dt$, which will be minimized by the set of differential equations (3):

$$E = -k_1 \sum_{j=1}^{N} c_j - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} (Q_{ij} + k_2)\, c_j c_i \tag{4}$$

If $-Q_{ij}<k_2<0$, $Q_{ij}+k_2$ will be positive, if c_i and c_j are bonds to neurons with a small distance in the layer above, and negative, if the distance is large. If $|k_1|$ is small, i.e., if the first term in (4) is negligible compared to the second one, it is possible to minimize the second term of the energy function by choosing c_i and c_j with the same sign for nearby cells and different sign for distant ones. As a result of these competitive conditions we find two connected areas with different sign in the receptive field of a neuron in layer 3. k_1 inclines the energy surface that decides which synapses get a positive and which ones get a negative sign. Because of the symmetry of the Gaussian distribution, a cell with a ON- (or OFF-) center and a OFF- (or ON-) background is expected. But if $|k_1|$ is large, the first term in the energy function dominates. The result is a homogeneous receptive field with positive or negative synapses dependent on the sign of k_1. The covariance function Q_{ij} of the output of center-ON neurons of layer 3 has a Mexican hat form, whose parameters depend on the size of the center.

Determination of synaptic bonds using an Hopfield model: Simulations of the difference equations (1) and (2) to observe the maturation of cells in higher layers are computational expensive. The number of floating point operations per time step required to simulate the development of one cell in layer k with N synapses is proportional to $N^{k-2}\cdot(N+1)+5\cdot N+1$. Normally, approximately 10,000 time steps are needed until a cell is stable. A way to avoid this problem is to use a faster algorithm to find a minimum of the energy function. We apply the Hopfield model /5/, since our energy function has the same structure as the energy function used in /5/. The synapses of one neuron of the Linsker model can therefore be represented by binary Hopfield neurons with values {0.5,0.5}. This allows to simulate the maturation of the synapses of layer k without simulating the previous k-1 layers, because Q_{ij} contains all information about the signals that passed these layers. First, the covariance function Q_{ij} of the layer above is calculated and the synapses are distributed, e.g. with Gaussian density. Secondly, the synaptic bonds of the Hopfield model are calculated according to

$$J_{ij} = Q_{ij} + k_2 \tag{5}$$

Eventually, a random start vector c(0) is iterated with the Hopfield dynamics until c is stable:

$$c_i(t+1) = \frac{1}{2}\,\mathrm{sign}\Big(\sum_{j=1}^{N} J_{ij}c_j(t) + k_1 \Big) \tag{6}$$

This model is able to calculate cells which are near-optimal in respect to the distribution of the synapses, Q_{ij}, k_1, k_2 and r_2/r_1. In our simulations, energy minima are found after $\leq 12 \cdot N$ iteration steps. This is equivalent to only $\leq 12 \cdot N \cdot (2 \cdot N+1)+N^2$ floating point operations.

Results: In the simulations of the Hopfield model we set $k_1=0$, because our point of interest is the topology of the developing cells, and not the sign of the different areas. We examined two kinds of receptive fields in which 450 synapses were distributed:

a) Gaussian; the locations of the synapses were found by selecting random points with Gaussian density in the receptive field. Because only 450 synapses were statistically chosen, the statistical fluctuations are relatively high and therefore there is no obvious maximum at the center of the density distribution.
b) Centered Gaussian; a center with maximum density was constructed by multiplying the Gaussian density function with the linear function $f(r)=1-0.33 \cdot r$ ($r \in]0,3]$, distance to the center of the distribution).

First we examined the cells maturating in layer 3. According to our analysis we took the outputs of the cells in layer 2 Gaussian distributed. In simulations with the centered Gaussian distribution of the synapses we found that center-ON or center-OFF cells develop, if $k_2=-1$ is constant and r_2/r_1 ranges from 1.5 to 8 (Fig. 2.a). The size of the center depends on r_2/r_1, and if $r_2/r_1<1.5$ an homogeneous cell emerges. By using the Gaussian distribution, the density fluctuations lead to asymmetric cell types ($r_2/r_1 >1.5$), normally half-cells (Fig. 2.b).

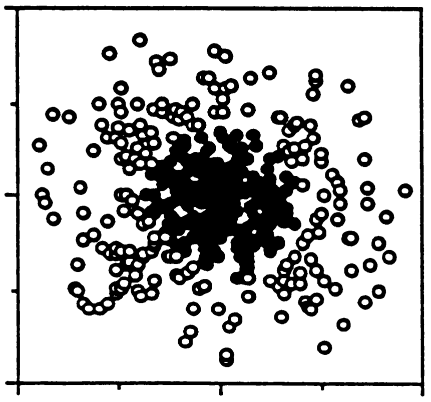
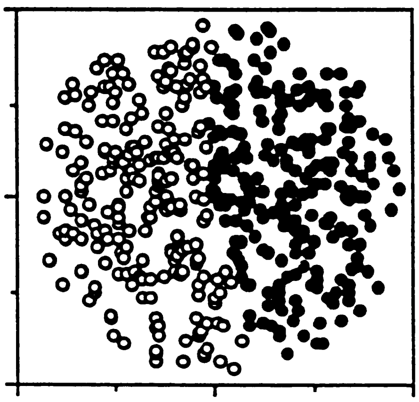



Figure 2.a (left) and 2.b (right): Maturated receptive fields of layer 2 simulated by the Hopfield model. 2.a shows a receptive field with the centered Gaussian distribution, 2.b the usual Gaussian distribution. All synapses developed either to -0.5 (bright circles) or to 0.5 (dark circles). Simulation parameters: $k_1=0$, $k_2=-1$, $r_2/r_1=4$.

Secondly, the emergence of different cell types in layer 4 was observed. Provided that layer 3 consists only of center-ON neurons, the covariance function Q_{ij} of layer 3 has Mexican hat form. Hence, we set $k_2=0$ and observed the maturation of cells by changing r_2/r_1. The receptive field with the centered distribution consists of concentric rings (Fig. 3.a) where the width of the rings depends on r_2/r_1. By using the normal Gaussian distribution, the rings cannot develop around a seed center. Due to the density fluctuations a few areas exist with relatively high density, and the rings try to group around these points. As a result parallel stripes emerge (Fig. 3.b) with the same width as in figure 3.a.

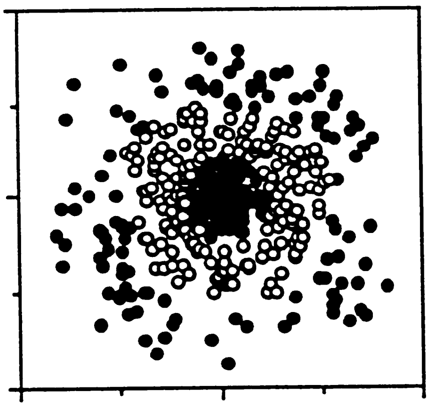
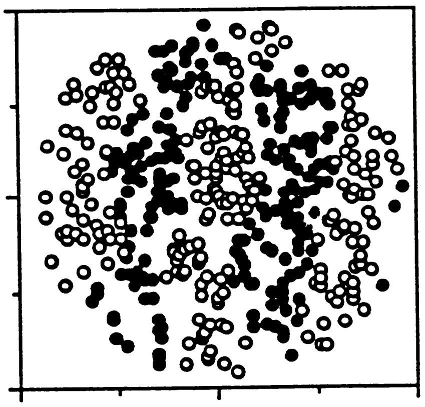



Figure 3.a (left) and 3.b (right): Maturated receptive fields of layer 3 simulated by the Hopfield model. 3.a shows a receptive field with the centered Gaussian distribution, 3.b the usual Gaussian distribution. Simulation parameters: $k_1=0$, $k_2=0$, $r_2/r_1=4$.

Conclusions: We showed by simulations the development of feature detecting cells in a Linsker-type linear feedforward multilayered neural network. The self-organizing emergence of center-ON neurons, half-cells and stripes-cells in response to random input signals has been observed. These cells resemble the cells of the visual system and correspond to minima of Linsker´s energy function. Such minima can be found in very short time using a modified Hopfield model. Simulations with different distributions of synapses in the receptive field resulted in the emergence of orientation selective cells already in the third layer. This is a consequence of the statistical fluctuations of a 2-dimensional Gaussian distribution of N elements.

References:

/1/ D.H. Hubel, T.N. Wiesel, "Brain Mechanism of Vision", Scientific American, Vol. 241. Sept. 1979, pp. 150-162

/2/ D.H. Hubel, "Eye, Brain and Vision", The Scientific American Lib., New York, 1988

/3/ R. Linsker, "From Basic Network Principles to Neural Architecture", Proc. Nat'l Acad. Sci. USA, Vol. 83, Oct.-Nov. 1986, pp. 7508-7512, 8390-8394, 8779-8783.

/4/ R. Linsker, "Self-Organization in a Perceptual Network", IEEE Computer, Vol. 21, No. 3, 1988, 105-117.

/5/ J.J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", P.N.A.S. 79, 2554-2558, April 1982

TEMPORAL-SPATIAL CODING TRANSFORMATION: CONVERSION OF FREQUENCY-CODE TO PLACE-CODE VIA A TIME-DELAYED NEURAL NETWORK

David C. Tam †

Department of Physiology
University of California, Irvine, CA 92717

dt4@nihcudec.bitnet

† *Present address:* ***Laboratory for Cellular and Molecular Neurobiology***
Neural Systems Section, Park Building 5, Rm 431
National Institute of Neurological Disorders and Stroke
National Institutes of Health, Bethesda, MD 20892

Abstract

A multi-layered time-delayed neural network is proposed to convert from a temporal (or frequency) coding scheme of biological neurons to spatial (or place) coding scheme distributed in different neurons, where each neuron encodes a different band-pass filtered frequency (or interval) of firing of the original input. The proposed network represents an implementation of a signal processing scheme for code conversion using time for computing and coding.

1. Introduction

One of the differences between biological neurons and artificial neural elements (or computing nodes) commonly used in artificial neural networks is the coding scheme employed for signal transmission. The signal encoded by the biological neurons in vertebrates as well as most invertebrates for long range communication is transmitted along axons via action potentials. Most action potentials transmitted along axons have characteristic (and stereotypic) pulse height and pulse width even though the pulse shape of the action potential may vary depending on the type of neuron. Although graded potentials (where pulse width and pulse height vary) are observed in most invertebrates and in the retina of the vertebrates, this class of signal transmission will not be discussed in this paper. In contrast, artificial neural elements often employ either binary signal (but not pulse signal) or continuous signal for signal transmission.

Given the stereotypic characteristic of constant pulse height and pulse width of action potentials, the signal transmitted along an axon may be idealized as pulse coded. The time-series of action potentials transmitted along an axon is often called a spike train. The encoding of information via the firing intervals (and consequently firing frequencies) is called *frequency-code* in neurobiology; whereas the encoding of information via the firing in a certain ensemble of neurons is called *place-code*, which is based on the locale of the neurons.

Since both frequency-code and place-code are often observed to be used in the same structure (or sub-network) in the central nervous system, the conversion and interchange of these coding schemes may be necessary for the system to function properly. An idealized multi-layered time-delayed neural network is constructed to elicit insights about plausible mechanisms for conversion of frequency-code to place-code given a spike train as the input and output signal.

Various time-delayed neural network models, particularly those employing Hebbian-type learning rule, have produced some seemingly obvious (yet non-trivial) functional equivalent forms of conventional cross-correlation functions [2];

implemented with reinforcement learning to account for delayed pairing association in classical conditioning [3]; implemented in a hybrid model of nonlinear weight-sum model for detecting correlations among simultaneously recorded spike trains [4]; and implemented with analogous models of sublattice magnetization for encoding and retrieval of temporal sequences [1].

2. The Multi-Layered Time-Delayed Neural Network Model

2.1. Definitions of Input Spike Train and Interspike Interval

A spike train, x(t), can be defined as a time-series of spikes (or delta-functions) with a total of n+1 spikes:

$$x(t) = \sum_{j=0}^{n} \delta(t - \tau_j) \tag{1}$$

That is, at time $t = \tau_j$, there is a spike occurring in the spike train given by the delta-function, which satisfies:

$$\delta(t) = \begin{cases} 1, & t = 0 \\ 0, & t \neq 0 \end{cases} \tag{2}$$

For discrete time, let us assume the time increments are quantized by Δt. Without loss of generality, let us normalize $\delta(t)$ by a factor of Δt (or alternatively, setting $\Delta t = 1$) such that the new $\delta(t)$, which we will use in this paper, becomes:

$$\delta(t - \tau_j) = \begin{cases} 1, & \text{for } \tau_j \text{ at interval } [t, t+\Delta t) \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

The *interspike interval,* I_j, is defined as the time interval between any two adjacent spikes occurring at time τ_j and τ_{j-1}:

$$I_j = \tau_j - \tau_{j-1}, \qquad \text{for } 0 < j \leq n \tag{4}$$

2.2. Time-Delayed Input Relationships of the Neurons in the First Layer

Consider a multi-layered network where the k-th neuron in the first layer has k input lines. Let $x_i(t)$ be the input signal from the i-th input to the k-th neuron in the first layer and $w_{ki}(t)$ the connection weight of the corresponding synapse. The total input, $X_k(t)$, to this neuron is a weighted-sum of all k inputs,

$$X_k(t) = \sum_{i=1}^{k} w_{ki}(t)\, x_i(t) \tag{5}$$

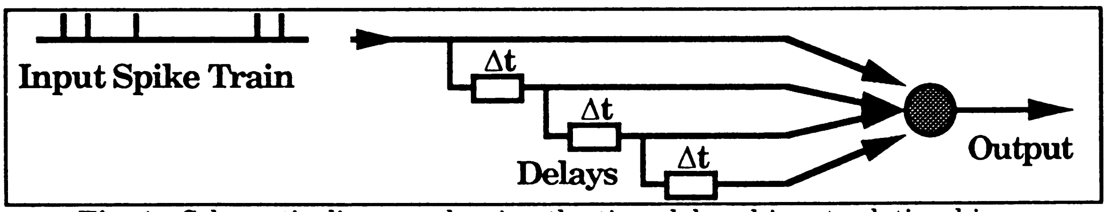


Fig. 1 Schematic diagram showing the time-delayed input relationships

The input relationships of a typical neuron in the first layer is illustrated in Fig. 1. For simplicity, the input to the entire network is given by a single spike train, x(t) as defined by Eqs. (1) and (3), which is then delayed by various time delays, $\tau = i\Delta t$, for i = 1 to k. These various delayed inputs are then connected to the neurons of the first layer, so the i-th input to the first layer neuron becomes:

$$x_i(t) = x(t - i\Delta t) \tag{6}$$

Assuming that $w_{ki}(t) = 1$, a constant for simplicity (for all i), then Eq. (5) becomes:

$$X_k(t) = \sum_{i=1}^{k} x_i(t - i\Delta t) \tag{7}$$

2.3. High-Pass Filtered Output Function of the Neurons in the First Layer

Given the input of Eq. (7) to the k-th neuron in the *first* layer, if the interspike interval of the original input spike train falls within the time-delay window, $I_j \leq k\Delta t$, for $0 < j \leq n$, then

$$X_k(t) = \sum_{i=1}^{k} x_i(t - i\Delta t) > 1 \tag{8}$$

Conversely, if $I_j > k\Delta t$, for $0 < j \leq n$, then $X_k(t) \leq 1$. Thus, if the threshold of the output of the k-th neuron is set at one, then this neuron will fire only when the interspike interval, I_j, of the input spike train is within the time-delay window, $k\Delta t$. That is, the output of this k-th neuron is given by:

$$y_k(t) = \begin{cases} 1, & \text{if } X_k(t) = \sum_{i=1}^{k} x_i(t - i\Delta t) > 1 \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

In other words, the k-th neuron can be considered as encoding only high-pass filtered input frequency, where instantaneous frequency is defined as the reciprocal of interspike interval, i.e., $1/(k\Delta t)$. Thus, the k-th neuron will fire only if the original input spike train contains instantaneous frequencies above this cutoff frequency of $1/(k\Delta t)$. In order to compensate for the effects of phase differences of various delays, the neuron has a refractory period of $(k-1)\Delta t$.

2.4. Elimination of Higher-Order Intervals within the Time-Delay Window by the Second Layer

Although it is customary to describe in frequency domain, it is more appropriate to describe our network in time (interval) domain. If there are more than two spikes occur within the time-delay window, then the neuron in the first layer may be overestimating the cutoff interspike interval, in which case the interspike intervals are, in fact, much shorter than the cutoff interval. To eliminate this effect due to the intervening spikes (which results in estimating higher-order intervals instead of the first-order interspike interval), another layer of neurons (called *first-parallel* layer) is added similar to the first layer except that the output threshold is set at two instead of one similar to Eq. (9). If the difference between the outputs of the first layer and this first-parallel layer neurons is computed by neurons in the *second* layer, it will ensure accurate estimation of true interspike interval within the time-delay window. The input connections to the neurons in the second level are given by an excitatory connection from the first layer, $y_k(t)$, and inhibitory from the first-parallel layer, $y'_k(t)$. Thus, the output of neurons in the second layer, $y''_k(t)$, is given by:

$$y''_k(t) = y_k(t) - y'_k(t) = \begin{cases} 1, & 2 \geq \sum_{i=1}^{k} x_i(t - i\Delta t) > 1 \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

2.5. Band-Pass Filtered Output of the Neurons in the Third Layer

Given the various high-pass filtered signal, a band-pass filtered signal can easily be obtained by the *third* layer neurons, whose input derives from two sources, one excitatory and the other inhibitory from the second layer. That is, the output of a third layer neuron, $y'''_k(t)$, is obtained from the difference between the outputs of k-th and (k-h)th neurons in the second layer:

$$y'''_k(t) = y''_k(t) - y''_{k-h}(t) = \begin{cases} 1, & 2 \geq \sum_{i=k-h}^{k} x_i(t - i\Delta t) > 1 \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

This neuron will fire only on a band-pass filtered input spike train signal where the band-width interval is $h\Delta t$, delimited by the interspike intervals of $k\Delta t$ and $(k-h)\Delta t$ (or between frequencies of $1/k\Delta t$ and $1/[(k-h)\Delta t]$). The firing of a particular neuron indexed by (k,k-h) in this third layer can be considered as encoding the corresponding band-limited interspike intervals of the original input spike train. In other words, the frequency-code (or interspike intervals) of the input spike train is converted to the firing of a distributed set of neurons in the subsequent layers, whose locations (or indices) indicate the frequency-band (or interval-band) of the input signal, thus the signal is encoded *a la* a distributed place-code scheme.

References

[1] Herz, A, Sulzer, B., Kühn, R., and van Hemmen, J. L. "Hebbian learning reconsidered: representation of static and dynamic objects in associative neural nets". *Biological Cybernetics,* Vol. 60, 457-467, 1989.

[2] Tam, D. C. and McMullen, T. A. "Hebbian synapses as cross-correlation functions in delay line circuitry". *Society for Neuroscience Abstract,* Vol. 15, 1989. (in press)

[3] Tam, D. C. and McMullen, T. A. "Classical conditioning, correlation function and synaptic weight modification in delay line circuitry: implications for cerebellar cortical function." *(submitted to Neural Information Processing Systems 1989)*

[4] Tam, D. C. and Perkel, D. H. (1989) A model for temporal correlation of biological neuronal spike trains. *Proceedings of the IEEE International Joint Conference on Neural Networks 1989.* Vol. 1, pp. I-781–786.

The Evolution of Connectivity: Pruning Neural Networks Using Genetic Algorithms

Darrell Whitley and Christopher Bogart
Computer Science Department
Colorado State University
Fort Collins, Colorado 80523
whitley@CS.Colostate.edu

1. Introduction

Defining the connectivity of an artificial neural network so as to enhance learning speed or to use only "necessary" connections is a difficult task that is not well understood. Most researchers rely on simple heuristics, the most common being "if the net does not work, add more hidden units." We evolve the connectivity of a feed forward neural network using genetic algorithms. The nets that are developed display interesting properties; they are smaller, learn faster and the amount of time required for learning is extremely consistent.

The general approach discussed here and previously used by Miller, Todd and Hegde (1989) is to define a net that is as large or larger than necessary to do the job, and then to use a genetic algorithm to define which combination of connections are sufficient to quickly and accurately learn to perform some target task using back propagation. Miller et al. did this for some small nets. However, our experiments with larger problems exposed two difficulties with their approach.

First, there is no explicit mechanism for rewarding nets that use fewer connections. This is a non-trivial problem, since directly rewarding or penalizing a net based on the number of connections used can give a "selective advantage" to nets that are not able to learn; in the extreme case, a net might try to gain reward or avoid penalities by pruning all of its connections. This paper defines a way to reward nets that use fewer connections while at the same time selecting for nets that learn quickly and accurately. Ways of extending this reward system are also discussed.

Second, on larger nets, the amount of time required to find a net that learns quickly and accurately is quite significant, since it means evaluating a population of strings, where each evaluation involves running back propagation on a single net. For a simple net which learned to compute the exclusive-or (Xor) of its two inputs, Miller et al. used enough back propagation cycles that they could reasonably expect to find a net that would be able to learn the task completely. But this is not practical on larger problems because it means training the same net over and over again. A genetic algorithm requires the evaluation of 1) an initial random population and 2) all the offspring that are generated by recombination. Evaluation in this case means running back propagation for some number of times to find out how quickly the net learns. This means that for a net which initially has 50 connections, it typically would be necessary to run back propagation on perhaps 2,000 (and probably more) different nets in order to search for a net with the desired properties.

One alternative is to use a reduced number of back propagation cycles. The difficulty with this approach is that the resulting nets would not be able to completely learn the problem during evaluation. Furthermore, a fast initial reduction in error does not guarantee that the net will continue to learn quickly, or is even able to completely reduce the error for the target problem. To avoid this, we train the net before pruning. While this might seem at odds with the objective of finding a net that learns quickly, having to do back propagation from scratch 2,000 times (or even 100 times) is a false economy. The real issues, we think, are the following. 1) This approach finds much smaller nets that can learn the task. In particular this has important implications for problems where a hardware implementation is the ultimate goal. 2) Pruning has an impact on generalization and noise tolerance (Mozer and Smolensky 1989). 3) Since pruning does introduce error, there is selective pressure in our approach for nets that relearn quickly.

As a side effect of our research, the previous research by Miller, et al. and discussions with Dr. Jerry Jones at the Colorado School of Mines, we have arrived at a conjecture suggested by our empirical results. *It appears that for some nets with a hidden layer, faster learning can be achieved by adding additional direct connections from the input layer to the output layer.* At this point, this suggestion should only be considered a hypothesis and only rigorous testing can confirm or refute this hypothesis, but there is sufficient evidence to warrant further research along these lines. Several hundred runs on Xor and 2-bit adders suggest that nets with direct I/O connections are not only somewhat faster, but also that these nets are not as prone to becoming stuck in local optima.

2. The Problem Encoding and GENITOR

A genetic algorithm manipulates a population of "chromosome-like" problem encodings referred to as "genotypes." The genetic algorithm uses selective pressure so that the "most fit" genotypes in this population are allowed to reproduce via recombination more often that those which are less fit. Although this sounds simple, it can be shown that this yields a robust and efficient search of a problem solution space by sampling hyperplanes in an L-dimensional hypercube, where L represents the length of a binary encoding of the problem.

The problem encoding is relatively simple. Consider the following binary string: 1101001100101101. This string would represent, in the current problem, a net which originally was solved using 16 links. The potential solution represented by this string uses only 9 of those sixteen links. Recombination requires two parents. Consider the string 1101001100101101 and another binary string which we represent using x and y to represent 0 and 1. Let the second string be yxyyxyxxyyyxyxxy. Using two "break-points" recombination occurs as follows:

$$\begin{array}{c} -11010 \times 01100101 \times 101- \\ -yxyyx \times yxxyyyxy \times xxy- \end{array}$$

Swapping the fragments between the two parents produces the offspring 11010yxxyyyxy101 and yxyyx01100101xxy.

Miller et al. defined their operators such that functional substructures are swapped during recombination. Their "crossover" operator swapped all the links leading into some node. This in effect only allows the strings to "break" at a limited number of positions along the encoding. While this makes intuitive sense, theoretically this prevents the genetic algorithm from searching across all possible hyperplanes in the search space. In accord with the schema theory for genetic algorithms, we allowed crossover at any position using a variation on Booker's two point reduced surrogate operator (1987). The empirical results do not allow us to argue that either approach is better at this point, although we found smaller and faster nets for Xor than the nets they typically discovered.

In a standard genetic algorithm the parents are replaced by their offspring after recombination. The idea is that the genetic material of the parents will largely survive and remain in the population, although the parents themselves do not. (See Holland 1975 and Goldberg 1989 for a review of genetic algorithms.) In the *GENITOR* approach the offspring do not replace their parents, but rather a low ranking individual in the population. In this way the algorithm is much more likely to accumulate information about high performance hyperplanes. A theorem has been developed which suggests that this approach does as well or better than a standard genetic algorithm at finding good genetic "building blocks" in the form of schemata (Whitley and Kauth 1988). Another key difference is that the *GENITOR* algorithm is designed to allocate reproductive trials to individuals (ie: an individual problem encoding or "genotype") based on their rank in the population rather than directly using the evaluation function as a fitness measure. When fitness is directly taken from the evaluation function, "selective pressure" can fluctuate so that the algorithm converges too quickly, or the search may stagnate because the selective pressure is inadequate. *GENITOR* abandons fitness values which are directly proportional to performance and instead uses the rank of the "genotype" in the current population to assign a fitness value. Allocating reproductive trials in this way ensures that a constant and effective selective pressure can be maintained no matter how performance is calculated.

Genetic algorithms have also been used to set the weights in small neural networks. Our work in this area indicates that genetic algorithms work well on smaller nets, but have difficulties after the length of the encoding becomes large (Whitley and Hanson 1989). Long encodings are not a problem for neural net pruning for reasons outlined below. Lawrence Davis of BBN has optimized neural nets using a hybrid algorithm that combines recombination and back propagation; the hybrid algorithm out performed either method alone when training a large net (500 connections) for a complex pattern recognition application.

3. The Evaluation Function and Reward Scheme.

The evaluation function is a key ingredient of a genetic algorithm since it is used to assign a "fitness" to the strings in the population. Evaluation in this case involves running back propagation on the net using the connections indicated by the binary encoding. In our approach, the initial net has already solved the problem, so we are actually pruning a developed net and finding which nets are least affected or best able to relearn given a specified number of back propagation cycles. The more links that are removed, the greater the probability that the functionality of the net will suffer. Thus, if the goal is to prune the net, there must be some compensation for nets that use fewer links.

The reward scheme used here is simple but effective. We start with some baseline number of back prop cycles, B. For each link that is pruned from the net, we increase the number of back prop cycles the net is allowed by some delta, D. Thus, if the genetic algorithm defines a net that has N number of links pruned, then we allow (ND + B) back prop cycles. This means that nets with fewer links are given more learning opportunities, but they are not actually rewarded unless they are able to exploit the opportunity. At this point, we are using the weights that have been learned for the already trained fully connected net (we refer to these as the "starting weights") to initialize the pruned nets. Another approach which we feel would be more effective, but which we have yet not implemented, is to use the new weights that are subsequently learned for smaller nets to initialize newly created nets. In other words we would dynamically update the starting weights as the net is being pruned. (They might be taken from the parent nets, for example.) A related idea we have not implemented is to set a criterion level to stop back propagation; if two nets reduce the error to the specified level, then they should be ranked according to the number of links used. This would create additional selective pressure for smaller nets. In the current implementation, the nets are allowed to run the total number of back prop cycles allocated to them, and then are ranked according to which nets have the lowest error. While this approach does create some selective pressure for smaller nets, there is still considerable advantage for those nets that are not too disrupted by pruning. Dynamically moving the starting weights would give more selective advantage to smaller nets, since they would not have to relearn from the original starting weights. In the current tests, we approximated the effect of dynamically updating the "starting weights" by running the genetic algorithm twice. This meant that we could run the algorithm once, find a smaller net, update the starting weights and then run the genetic algorithm using these new weights. This also meant that we could progressively encode the problem using smaller strings--thus the encoding length is not a problem for optimizing net connectivity.

Clearly, the aim of our evaluation function is to direct the genetic algorithm toward performing a kind of pruning. Other pruning schemes have been developed or suggested, although these typically involve pruning activation units rather than links (Mozer and Smolensky 1989). Bits which turn on and off activation units could also be added to our encoding although we have not yet tried this. The attraction of the genetic algorithm for this problem is that it exploits whatever patterns it is able to find. Good mathematical or heuristic methods may eventually prove more efficient; studying the pruning behavior of the genetic algorithm might be one means of developing such heuristics. One reason this might not be the case is that the genetic algorithm is able to exploit the idiosyncrasies of each individual problem. At this point we have no comparative data.

4. Experiments and Results

We have tested our approach by pruning a fully connected two bit adder and Xor. On Xor, we begin with a standard net with two hidden units, with the addition of direct connections between the input layer and the output layer. The genetic algorithm pruned this to the standard net with one hidden unit, which is smaller than the net found by Miller et al. The adder net has four input nodes, four hidden nodes, and three output nodes; in addition, a true node was used for learning the bias, or activation thresholds. To begin with, all input units were connected to all hidden units, and all hidden units were connected to the output units. The true node is connected to all hidden units and output units. In addition to these connections, all input units were given a direct connection to each output unit. Finally, because the minimal adder described by Rumelhart et al. (1986) has connections between members of the hidden layer, we allowed hidden unit 1 to feed to hidden units 2, 3, and 4, hidden unit 2 to feed to hidden units 3 and 4, and hidden unit 3 to feed to hidden unit 4. Thus, this fully connected net has 53 connections.

In the first run the net was trimmed from 53 connections to 37. This smaller net was further trained using back propagation, then was used as a starting point for the genetic algorithm again. On this second run, the genetic algorithm pruned the net further to 29 connections. Figure 1 gives the net developed by the genetic algorithm.

Since the ability of a neural net to learn quickly can be very much depend on what weights it is initialized with, we took the "pruned" net and ran a number of learning experiments from small random initial weights in order to collect statistics about the net's learning ability. We wanted to show that we had not just "gotten lucky" in being able to learn a set of weights for the reduced net. The results were averaged over 50 runs. Surprisingly, the pruned net not only learned faster, but it was very consistent in the number of back propagation cycles its required to learn the problem (TABLE 1). We also noted that the pruned net uses several connections which go directly from the input layer to the output layer. Miller et al. found the same thing in their exclusive-or network. This lead us to also compare a net that is fully connected with direct connections between input and output (TABLE 1, NET B) to a net that is fully connected between the input layer and the hidden layer, and between the hidden layer and the output layer, but which does not allow a direct connection between the input and output layer (TABLE 1, NET A). Our results, averaged over fifty runs, suggest that nets which have links that directly go from the input to the output layer do learn somewhat faster. In hundreds of tests on Xor and the 2-bit adder, we also found that the standard nets without direct I/O connections become trapped in local minima about 5 to 10% of the time. The nets with the direct I/O connections were less likely to fail to converge to a solution. This suggests that they are less likely to become trapped in local minima. The pruned nets never became stuck, which suggests that the genetic algorithm finds nets that that are well suited to the problems they are trying to solve.

TABLE 1: DISTRIBTUION OF TRAINING TIMES FOR A 2-BIT ADDER

Number of Training Repetition	Percentage of Nets That Had Completed Training		
	NET A	NET B	NET C
8,000-9,000	4%	0%	100%
9,000-10,000	4%	0%	
10,000-20,000	10%	62%	
20,000-50,000	34%	32%	
50,000-100,000	24%	2%	
Over 100,000	14%	0%	
Never Converged	10%	4%	

NET A: Standard Fully Connected Net
NET B: Standard Net with Direct I/O
NET C: Pruned Net

SAMPLE SIZE: 50

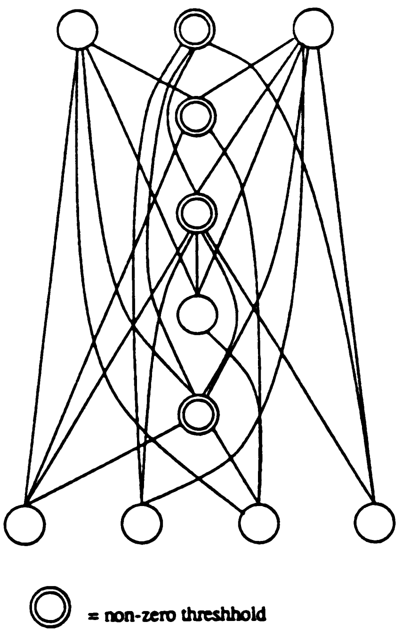


FIGURE 1. THE PRUNED NET FOR THE 2-BIT ADDER

ACKNOWLEDGMENTS

This research was supported in part by a grant from the Colorado Institute of Artificial Intelligence (CIAI). CIAI is sponsored in part by the Colorado Advanced Technology Institute (CATI), an agency of the State of Colorado. CATI promotes advanced technology education and research at universities in Colorado for the purpose of economic development.

REFERENCES

Booker, L. (1987) Improving search in genetic algorithms, in: Lawrence Davis (Ed.), *Genetic Algorithms and Simulated Annealing*. Morgan Kauffmann.

Goldberg, D. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning* Addison-Wesley.

Holland, J. (1975) *Adaptation in Natural and Artificial Systems*. Univ. of Michigan Press, Ann Arbor.

Rumelhart, D., Hinton, G. and Williams, R. (1986) Learning Internal Representations by Error Propagation. *Parallel Distributed Processing, Vol I* Cambridge, MA: MIT Press.

Miller, G., Todd, P. and Hedge, S. (1989) Designing Neural Networks using Genetic Algorithms. *Proceeding of the Third Internation Conference on Genetic Algorithms*. Morgan Kaufmann.

Mozer, M. and Smolensky, P. (1989) Skeletonization: a technique for trimming the fat from a network via relevance assessment. *Advances in Neural Network Information Processing Systems*. Morgan Kaufmann.

Whitley, D. and Hanson, T. (1989) Optimizing Neural Networks using Faster, More Accurate Genetic Search. *Proceeding of the Third Internation Conference on Genetic Algorithms*. Morgan Kaufmann.

Whitley D., and Kauth J. (1988) GENITOR: a different genetic algorithm. *Proceeding of the Rocky Mountain Conference on Artificial Intelligence*. Denver, CO.

Biophysical Model of a Hebbian Synapse

Anthony Zador[1], Christof Koch[2], Thomas H. Brown[1]

[1]*Department of Psychology, Yale University, New Haven CT 06520*
[2]*Divisions of Biology and Engineering, California Institute of Technology, Pasadena CA 91125*

Introduction

A Hebbian synapse is one whose enhancement is caused by a simple conjunction of presynaptic and postsynaptic activity or by a positive correlation between these activities (Brown et al, 1988b; Brown et al, 1990; Sejnowski and Tesauro, 1989). Hebbian synapses have long been postulated to mediate information storage in the mammalian brain (Hebb, 1949). Theoretical studies have shown that powerful forms of learning and self-organization can emerge in networks of simple processing elements whose interconnections are controlled by a Hebbian algorithm (Kohonen, 1984; Hopfield, 1984; Linsker, 1988). Interest in this class of algorithm (reviewed in Brown et al, 1990) was increased by the recent discovery (Kelso et al, 1986) that Hebbian synapses actually exist in the hippocampus, a temporal lobe structure that has long been implicated in the declarative memory system. This Hebbian mechanism in the hippocampus was shown to be responsible for a type of synaptic plasticity called long-term potentiation (LTP). LTP is an increase in synaptic efficacy that can be induced by seconds or less of the appropriate stimulation and that has been shown to last for hours, days, weeks or longer. In accordance with Hebb's postulate for learning, the induction of LTP was shown to depend upon the co-occurrence of presynaptic and postsynaptic activity (Kelso et al, 1986). Enough is now known about the underlying biophysical mechanisms (reviewed in Collingridge and Bliss, 1987; Brown et al, 1988a,b, 1989) to begin constructing realistic models of the process.

The induction of LTP is thought to result from a series of enzymatic reactions triggered by a localized increase in the postsynaptic [Ca^{2+}] in the dendritic spine (Brown et al, 1988a,b; 1989). The Hebbian mechanism is believed to result partly from the unusual gating property of a Ca^{2+}-permeable ionic channel (the NMDA receptor-gated channel) located on the head of the dendritic spine. This channel opens to a state with a significant Ca^{2+} permeability only if two conditions are simultaneously satisfied: (1) the NMDA receptor that is associated with the channel must be bound by a suitable agonist, such as the neurotransmitter glutamate; and (2) the voltage across the membrane containing the channel must be made less negative in order to relieve a blockade of the channel by Mg^{2+}. The receptor-iontophore complex was thus proposed to act as an AND-gate for the co-occurrence of presynaptic and postsynaptic activity (Brown et al, 1988a). Because of the microscopic size of dendritic spines, it has not yet been possible to make direct measurements of some of the biophysical and molecular events that are suspected to control the induction of LTP at Hebbian synapses. Partly for this reason, biophysical models are valuable for gaining insights into some of the underlying dynamics and possible computations. A previous study (Gamble and Koch, 1987) developed a model for examining Ca^{2+} dynamics in spines whose membranes contain classical voltage-dependent Ca^{2+} channels but not NMDA receptor-gated channels. Here we report preliminary simulation results from the first operational biophysical model of a Hebbian synapse.

Methods

We simulated a compartmental model of calcium influx and voltage changes at the Schaffer collateral input to dendritic spines of pyramidal neurons from hippocampal region CA1 (for anatomy, see Brown and Zador, 1990). There were four calcium compartments and two voltage compartments. Calcium entered through NMDA receptor-gated channels located on the spine head, which was approximated by two cylinders (0.25 μm radius and lengths of 0.05 μm and 0.25 μm). It then diffused along the length of the spine neck, which was approximated by two cylinders (0.05 μm radius and lengths of 1.0 μm each). The spine neck was attached to the dendritic shaft, which acted as a Ca^{2+} sink of constant concentration (0.05 μM). In each compartment, Ca^{2+} also interacted with a saturable first-order calmodulin-like buffer and with a saturable first-order Michaelis-Menton Ca^{2+} pump. We used a standard buffer concentration of 55 μM (110 μM in the outermost spine head compartment), a forward rate constant of 0.05 $\mu M^{-1} ms^{-1}$, and a reverse rate constant of 0.5 ms^{-1}. Equation 1 was used for the Ca^{2+} pump:

$$\frac{d[Ca]_P}{dt} = P \frac{[Ca] - [Ca]_r}{[Ca] - [Ca]_r + K_d} , \qquad (1)$$

where d[Ca]p/dt is the change in the free Ca^{2+} concentration due to the pump; [Ca] is an abbreviated notation for the free calcium concentration as a function of time; the resting level of free calcium, $[Ca]_r = 0.05$ µM; P = (Pump Surface Density)·(Membrane Area)·Kp; the maximum pump rate, Kp = 0.2 ms^{-1}; the dissociation constant, $K_d = 0.3$ µM; and the pump surface density is 5×10^{-16} µMol·µm^{-2}. The electrical properties of the spine were modeled by two passive RC compartments coupled by a 175 MΩ resistor representing the spine neck. The spine head compartment consisted of a 28.6 pS leak conductance in parallel with a 0.04 pF capacitance, while the lumped dendritic compartment consisted of a 400 MΩ leak resistance in parallel with a 20 pF capacitance. Both leak conductances were in series with a -80 mV battery representing the (inside negative) resting potential of the cell.

The model depended critically upon the description of the synaptic conductance. The postsynaptic currents were generated by two types of ionic channels. Current generated by the non-NMDA receptor-gated channels (the "non-NMDA current") was represented by Equation 2 (from Brown and Johnston, 1983; Brown et al, 1988a):

$$I(t) = (E_{syn} - V_m)\, g_m\, t \exp(-t/t_p)\ , \qquad (2)$$

where V_m is the membrane potential in mV; $g_m = 0.9$ nS (corresponding to a peak synaptic conductance of 0.5 nS); the synaptic equilibrium potential, $E_{syn} = 0$ mV; and $t_p = 1.5$ ms. Parameters are from experimental data summarized in Brown et al (1988a). Equation 3 (from C. F. Stevens, personal communication) was used to represent the current generated by the NMDA receptor-gated channels (the "NMDA current"):

$$I(t) = (E_{syn} - V_m) g_m \frac{1}{1 + K_n\,[Mg]\exp(-0.06V_m)} [\exp(-\beta_1 t) - \exp(-\beta_2 t)]\ , \qquad (3)$$

with $g_m = 0.2$ nS (corresponding to a peak conductance of 0.17 nS, or one third of the maximum non-NMDA current); $\beta_1 = 0.01$ ms^{-1}; $\beta_2 = 0.47$ ms^{-1} (corresponding to a time-to-peak of 10 ms and a decay time of 100 ms); [Mg] = 3.0 mM; $K_n = 0.33$ mM^{-1}; and $E_{syn} = 0$ mV for both the Ca^{2+} and non-Ca^{2+} components of the NMDA current. The Ca^{2+} component was 2% of the total NMDA current.

Results

The model was explored by simulating both a single stimulation of the synaptic input and a train of five successive stimulations (at 100 or 333 Hz). The resulting changes in $[Ca^{2+}]$ were determined as a function of the presence or absence of a depolarizing voltage clamp (-15 mV) applied to the dendritic compartment. When the voltage clamp was not applied, the $[Ca^{2+}]$ in the subsynaptic compartment reached a peak of 0.16 µM (following 1 synaptic stimulation) or 2.0 µM (following a train of 5 stimulations at 100 Hz). When these same synaptic stimulations were paired with the depolarizing voltage clamp, the peak $[Ca^{2+}]$ in the subsynaptic compartment reached 2.1 µM (following 1 stimulation) or 78.5 µM (with train of 5 at 100 Hz). Based on experimental studies, only the last of these four activity combinations (100 Hz train of 5 excitations presented during an additional imposed depolarization) would be expected to induce LTP in the Schaffer collateral synapses (Kelso et al, 1986; Brown et al, 1989). The actual $[Ca^{2+}]$ required to induce LTP at a single synapse has not been determined experimentally; nor is it known whether the induction of LTP is an all-or-nothing event for a single synapse. In calibrating the model with respect to LTP induction, we have tentatively assumed that--for the synaptic enhancement to occur--the peak transient $[Ca^{2+}]$ must exceed a "threshold" level of at least 10 µM, and possibly a much higher value. With this assumption, the model has thus far only yielded LTP under those conditions in which the experimental results suggest that it should.

We were particularly interested in understanding the spatiotemporal specificity of the Hebbian mechanism (Kelso and Brown, 1986; Kelso et al, 1986; Brown et al, 1988a, 1989). In this regard, the simulations suggested an obvious role for the dendritic spine in enabling spatial or "input specificity." The location of the NMDA receptors on the spine head, rather than on the dendritic shaft, was shown to be important for two reasons. One major effect of the spine was an "amplification" of the $[Ca^{2+}]$ transients. Because of the minuscule volume of the spine, relatively small Ca^{2+} currents cause a large change in the $[Ca^{2+}]$ when compared with the effect of a similar synapse located on a dendritic shaft (represented as a cylinder with a 1 µm radius). In the latter case, the 100 Hz train of 5 stimulations, delivered during the depolarizing voltage clamp, generated a peak $[Ca^{2+}]$ response of only 0.25 µM in the subsynaptic (dendritic) compartment. A second major effect of the spine was the "compartmentalization" of the peak $[Ca^{2+}]$ response to synaptic activation. This effect results largely because of the narrow spine neck, which causes the $[Ca^{2+}]$ transients to be restricted almost completely to the spine region. These two effects--amplification and compartmentalization--help explain how synaptically produced increases in [Ca^{2+}] that are above threshold for inducing LTP can be limited to just the active synapses.

The simulations also helped to elucidate temporal aspects of the Hebbian mechanism. The long decay time course (β_1 = 0.01msec^{-1}) of the NMDA current predicts that synaptic inputs separated by intervals much longer than the membrane time constant (about 15 - 20 ms in these neurons; Brown et al, 1981; unpublished observations) can interact to produced Hebbian modifications. We examined the temporal aspects of the Hebbian mechanism as follows. A 333 Hz train of 5 stimulations was delivered in various temporal relationships to the dendritic voltage clamp (to -15 mV). The [Ca^{2+}] in the spine head was measured as a function of time for several different interstimulus intervals (ISIs). The ISI is defined as the time between the offset of the train of synaptic stimulations and the onset of the voltage clamp. The peak [Ca^{2+}] in the spine head is plotted against the ISI in the inset to Fig. 1. The time courses of the changes in [Ca^{2+}] are plotted for 5 ISIs in the main part of Fig. 1 (the ISI is indicated above each curve). It is clear that ISIs up to 100 ms could in principle engage the modification process if the [Ca^{2+}] threshold for plasticity is about 10 μM, as we assumed earlier. This may be the explanation for the "forward-pairing trace period" that we have discussed elsewhere (Brown et al, 1989; 1990) in regard to the role of this Hebbian mechanism in learning.

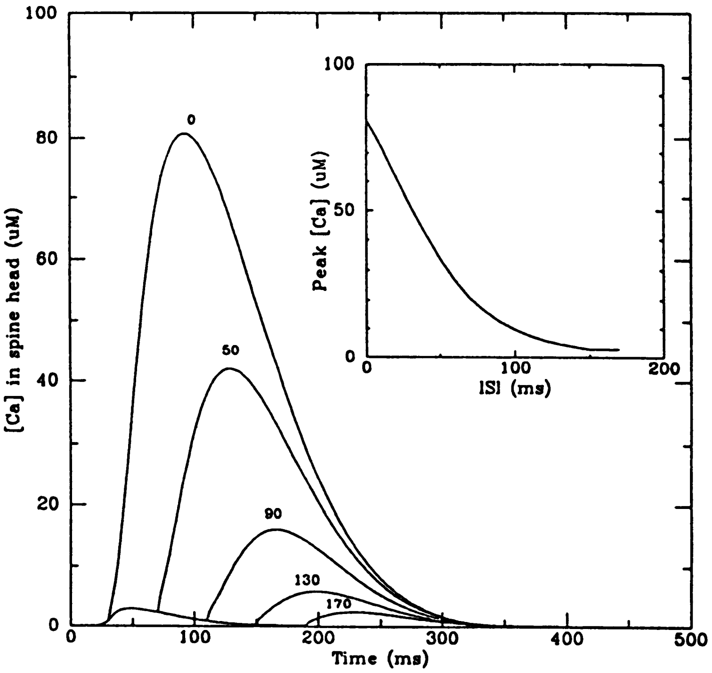


Fig. 1. The calcium ion concentration in the spine head [Ca^{2+}] is plotted as a function of time. A brief train of 5 synaptic stimulations at 333 Hz is presented 20 ms after the start of the simulation. The family of curves illustrate the effect on the [Ca^{2+}] transients of applying a depolarizing voltage clamp (an instantaneous step to -15 mV) in the dendrite as a function of the interstimulus interval (ISI), which is measured from the end of the train to the beginning of the voltage command. Curves corresponding to 5 ISIs (0, 50, 90, 130, or 170 ms) are labeled. The small [Ca^{2+}] transient (unlabeled) that occurs in the first 100 ms shows the effect of the same train of synaptic excitations in the absence of the depolarizing voltage clamp. Comparing this with the curve labeled 0 illustrates the dramatic effect of the depolarization on the peak increase in [Ca^{2+}]. The *inset* is a plot of the peak transient [Ca^{2+}] as a function of the ISI. For all ISIs ≤ 100 ms, a depolarizing step can increase the peak transient to more than 10 μM. At ISIs ≥ 150 ms, the voltage step is completely ineffective.

Discussion

We reported results from the first biophysical model of a well-studied Hebbian synapse, the Schaffer collateral input to the pyramidal neurons of the CA1 region of the hippocampus. These synapses display an "associative" form of LTP (Barrionuevo and Brown, 1983; Kelso and Brown, 1986) whose induction is governed by a Hebbian mechanism that utilizes the NMDA receptor-gated channel (Collingridge and Bliss, 1987; Brown et al, 1988b; 1989). Our biophysical model captures several key features of the known phenomenology of LTP in the Schaffer collateral synapses.

First, it accounts for the fundamental observation that LTP is in fact "associative" in these synapses. Repetitive stimulation of a weak synaptic input to the pyramidal neurons fails to induce LTP in that input unless a strong synaptic input to the same neuron is also stimulated at about the same time (Brown and Barrionuevo, 1983; Kelso and Brown, 1986). In the model described here, the NMDA receptor-iontophore complex accounts for this elemental conjunctive operation. Second, the model accounts for the known spatiotemporal features of associative LTP in these synapses. It is known that the synaptic modifications are input specific and that there is a limited temporal window within which associative interactions can occur (Kelso and Brown, 1986; Kelso et al, 1986; Brown et al, 1989). Two demonstrated properties of the Ca^{2+} dynamics in the dendritic spines--amplification and compartmentalization--help explain the spatial specificity. In regard to the temporal specificity, the model predicts a "forward-pairing trace period" of tens of milliseconds. There is in fact some experimental evidence for this, but the maximum ISI that will support associative LTP is still uncertain (reviewed in Brown et al, 1989). Finally, the model is consistent with the independence of LTP induction from its expression (Muller et al, 1988). This independence is thought to arise from the different roles of two, pharmacologically distinct, classes of receptor-gated ionic channels on the subsynaptic membrane--NMDA and non-NMDA (Collingridge and Bliss, 1987; Brown et al, 1989).

The results have obvious implications for further experimental analysis and for the formation of learning algorithms. It is clear that subtle biophysical details--such as the exact location the the NMDA receptor-gated channels (spine head versus dendritic shaft)--will have major computational consequences. As we develop more complete biophysical descriptions of neuronal signalling and plasticity, it should be possible to abstract some of the essential computations for inclusion into adaptive neural network models.

Acknowledgements

Supported by the Office of Naval Research. We thank Zachary Mainen and Anna Nobre for comments on the manuscript and C. F. Stevens for sharing his evidence regarding Equation 3.

References

Barrionuevo, G., Brown, T. H. 1983. Associative long-term potentiation in hippocampal slices. *Proc. Natl. Acad. Sci. USA* 80:7347-51

Brown, T. H., Chang, V. C., Ganong, A. H., Keenan, C. L., Kelso, S. R. 1988a. Biophysical properties of dendrites and spines that may control the induction and expression of long-term synaptic potentiation. In *Long-Term Potentiation: From Biophysics to Behavior*, Eds. P.W. Landfield and S.A. Deadwyler, pp. 197-260. New York: Alan R. Liss

Brown, T.H., Chapman, P. F., Kairiss, E. W, and Keenan, C. L. 1988b. Long-term synaptic potentiation. *Science* 242: 724- 728.

Brown, T.H., Fricke, R.A. and Perkel, D.H. Passive electrical constants in three classes of hippocampal neurons. *J. Neurophysiol.* 46:812-827, 1981

Brown, T. H., Ganong, A. H., Kairiss, E. W., Keenan, C. L., Kelso, S. R. 1989. Long-term potentiation in two synaptic systems of the hippocampal brain slice. In *Neural Models of Plasticity*, Eds. J. H. Byrne, W. O. Berry, pp. 266-306. New York: Academic Press

Brown, T. H., Johnston, D. 1983. Voltage-clamp analysis of mossy fiber synaptic input to hippocampal neurons. *J. Neurophysiol.* 50:487-507

Brown, T. H., Kairiss, E. W., Keenan, C. L. 1990. Hebbian synapses--biophysical mechanisms and algorithms. *Ann. Rev. Neurosci.* in press

Brown, T. H., Zador, A. 1990. The hippocampus. In *The Synaptic Organization of the Brain*, Ed. G. M. Shepherd, in press. New York: Oxford

Collingridge, G. L., Bliss, T. V. P. 1987. NMDA receptors: Their role in long-term potentiation. *Trends Neurosci.* 10:288-93

Gamble, E., Koch, C. 1987. The dynamics of free calcium in dendritic spines in response to repetitive synaptic input. *Science* 236:1311-15

Hebb, D. O. 1949. *The Organization of Behavior.* New York:Wiley

Hopfield, J. J. 1984. Neurons with graded response have collective computational properties like those of two-state neurons. *Proc. Nat. Acad. Sci.* USA 81:3088-3092

Kelso, S. R., Ganong, A. H., Brown, T. H. 1986. Hebbian synapses in hippocampus. *Proc. Nat. Acad. Sci.* USA 83:5326-5330

Kohonen, T. 1984. *Self-Organization and Associative Memory.* Berlin/Heidelberg: Springer Verlag

Linsker, R. 1988 Towards an organizing principle for a layered perceptual network. In *Neural Information Processing Systems*, Ed. D. Andersen. New York:American Institute of Physics, pp. 485-494

Muller, D., Joly, M., Lynch, G. 1988. Contributions of quisqualate and NMDA receptors to the induction and expression of LTP. *Science* 242:1694-97

Sejnowski, T. J., Tesauro, G. 1989. The Hebb rule for synaptic plasticity: Algorithms and implementations. In *Neural Models of Plasticity; Eds.* Byrne, J. H., Berry, pp. 94-98. New York: Academic Press

An Improved Competitive Learning Algorithm Applied to High Level Speech Processing.

Pedro L. GALINDO
Computer Science Faculty
Universidad Politecnica de Madrid

Thierry MICHAUX
Computer Science Faculty
Universidad Politecnica de Madrid
and
Université de la Sorbonne Paris III

In this paper, we summarize the last results of our work on the development of Competitive Learning neural networks.

We compare the performance of our algorithm with that of Stephen Grossberg ART1, and demonstrate how his method can be improved by modifying the attentional parameter dinamically during learning.

We illustrate the aplication of our model to natural language, and prove that spanish lexicon can be considered as an adaptive non arbitrary system.

1. INTRODUCTION.

Competitive Learning has been a subject of an unexpected small interest over the recent years. However, competitive models exhibit some unique capabilities, as unsupervised learning, self-organization, inmediate good results.

There are three main models that apply Competitive Learning, those of Rumelhart(1985), Carpenter and Grossberg (1987) and Fukushima (1988)

We have only considered in our studies the two first models, which have some claims in Natural Language processing.(Grossberg and Stone, 1986)

Rumelhart´s model is a simple, very fast competitive model. However the results gotten are strongly dependant on the initial conditions chosen, like the number of nodes of competition in each pool, and the number of pools.

Grossberg´s model ART1 gathers real self-organization characteristics: an effective use of full memory capacity, self-regulated optimization in search and sensibility, self-controlled growing. Its most important property is its capacity to preserve the stability-plasticity balance, when a familiar or unfamiliar pattern is presented. The coarseness of the categories learnt is regulated by a vigilance parameter (μ).

2. ART1 (FAST LEARNING) : THE MODEL.

A basic description of ART1 has appeared in Carpenter μ Grossberg (1988). A slightly simplified and modified version has been presented in Carpenter and Grossberg (1987): ART1, fast learning. This is our interpretation of this model.

ART1 is composed of two layers.

The first layer, F1, called input layer, receives the external signals from input patterns.

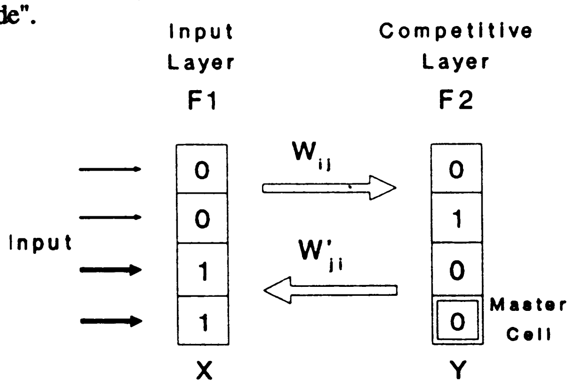
The second layer, F2, is designed as a competitive layer. There is always just one node active, the one that receives the largest input, meanwhile the others are inhibited. It contains a special node, called "Master node", that creates new nodes when it reaches an active state. At the beginning it is the unique node in F2.

Both layers are fully interconnected by assimetric connections. Competition is obtained by massive inhibitory connections between nodes in F2.

Let I the input pattern presented at F1. Let X the pattern of activation of F1. Let $|X|$ the number of active nodes in F1.

Signals from nodes in F1 are modulated by the weights of connections from F1 to F2.

Let Y the pattern of activation of F2, calculated in a competitive way. Hence, the node which receives the largest signal reaches an active state, and others get inactive. This node will be called "winner node".



I = Input
X = Activation at F1
Y = Activation at F2.
W ij = Connection Weights from F1 to F2.
W'ji = Connection Weights from F2 to F1.
μ = Attentional Parameter.

FIGURE 1. ART1 MODEL.

If the winner node is the "Master node", a new node is created in F2, being the new "Master node". The winner node can be considered as a new category. Learning is made by modifying the weights between nodes in F1 and the winner node.

If the winner node is not the "Master node", signals from nodes in F2 are modulated by the weights of connections from F2 to F1.

At that time, F1 receives, both, input signal I, and signals from F2. So, a new pattern of activation is created in it, X'.

Let $| X' |$ the number of new active nodes in F1.

If $| X' | / | X_i | < \mu$ the winner node in second layer is reset, so signals from F2 to F1 are inhibited, and the activation in F1 is X again. This produces a new iteration where a <u>new</u> node in F2 is chosen to win the competition.

If $| X' | / | X | >= \mu$ reset is prevented, and learning is made by modifying the weights between nodes in F1 and the winner node in F2.

This means that the proportion $| X' | / | X |$ must exceed the attentional parameter (μ) to prevent reset at second layer and make learning possible.

So, when μ is low, coarse categories are learned. When μ is high, fine categories are learned.

3. DYNAMIC MODIFICATION OF μ : THE IMPROVEMENT.

3.1 In ART1, the value of μ is constant during learning. This produces that:

A) The categories learnt have the same level of generality. They are coarse OR fine, general OR particular, depending on the value of μ.

B) The categories represent clusters, that classify input patterns. A CLASSIFICATION is made.

C) The search of winner node is just the search of the nearest category to input pattern.

3.2 Our main idea was to vary the value of μ during learning, from low to high values. The objective was to try to decompose the input patterns in their basic subunits.

The results were:

A) The categories learnt are both, coarse AND fine, general AND particular, at differents levels of accuracy.

B) These categories allow to define ALL the "subunits" in which each input pattern can be divided depending on the input set. Hence, a DECOMPOSITION is made, NOT A CLASSIFICATION.

C) The search of winner node, makes a Morphological Analisys of input patterns, because search is made from general to particular subunits.

D) The analisys made is only dependant on the global set of input patterns, in a non-supervised way, just considering the context.

3.3 Different modifications of μ have been made, and we obtained that sigmoid function gives the best performance for the same process time. In our simulations, we used a variation of μ as:

$$\mu(t) = \frac{1}{1 + \exp(- t)} \qquad \text{so,} \qquad \mu(t + dt) = \frac{1}{1 + \exp(- dt) * (1 - \mu(t)) / \mu(t)}$$

4. APLICATION TO NATURAL LANGUAGE : THE CONTEXT.

Two theories exist about the structure of Natural Language.

* The traditional one conceive Natural Language as an arbitrary, sequential and symbolic system.

* The recent one has tried to prove that it was parallel, distributed and adaptive.

In our research we were most interested in the relation between the morphological and the phonological structure. The relation between those levels has always been considered as highly arbitrary.

Our objective was to show that using a Competitive Learning model it was possibe to define a non-arbitrary sub-level with non-arbitrary subunits.(Michaux and Galindo,1989)

We applied the model to extract the structure of regular spanish verbs. The same process is currently applied to the whole spanish lexicon.

5. MORPHOLOGICAL DECOMPOSITION: THE RESULTS.

The goal of the simulation was to extract the inherent structure of the lexicon. To do that WITH NO TRICKS like wickelfeatures (MC Clelland and Rumelhart, 1986) we needed to codify words in the right way, without any structural information implicit in the codification.

We made the simplest possible codification, a matrix of binary inputs with as many rows as the length of the longest word at the input, and one column for each letter of the alphabet. So, a bit 1 in first row, fifth column means that the first letter of the input word is 'e'.

FIGURE 2. CODIFICATION OF THE WORD "GROSSBERG".

The input universe was a collection of eight spanish verbs, conjugated in present, past and future for first, second and third person of singular and plural, aligned on the stress (Galindo and Michaux, 1989).

The contrast of the performances between a simulation of ART1 for a value of $\mu = 0.5$ with the model proposed, when μ just reachs 0.5. is quite important. With the model proposed verbs are devided in morphological subunits

ART1 (μ=0.5)

```
     a o
      m s
  temia
   viv
     a a
  vivi
   a a
   ir
  lati
    e   n
    re
    aba
     is
latir
    ra
    cen
    pic
    dur
    late
    tem
    reis
```

MODEL PROPOSED (μ=0.5)

```
     o          em            reis
   i            aba         temia
   a           tap          vivi
   e            iamos         viv
  dur           eis           teme
  re             mos            amos
   a a        ara           cenar
   ia         ira           vivir
 a a            iais        durar
  ra         temer            cena
    m s         ias              aban
    is        are           latir
  vi            abamos      picar
   e n        ire           tapar
lati            emos             abas
  pic           ais           tapa
  cen           ian           picais
  lat           abais         duraba
                              tapaban
```

FIGURE 3. COMPARISON OF ART1 AND MODEL PROPOSED.

6. SOME MORPHOLOGICAL ANALYSIS : THE PROOF.

The previous paragraph show how our model can devide the verbs in their morphological componants only wit a $\mu = 0.5$. If we let the model stabilize with $\mu = 0.9$ we get the following results.

We will consider two examples with $\mu = 0.9$. the inputs are the verbs *picabais* and *durare*.

INPUT = P I C A B A I S

	1	A	1º The stress vowel, class of the verb
	2	A A	2º Correlation of the stress Vowel with the next vowel
	3	A B A	3º Detemination of the tense: imperfect
	4	P I C	4º Determination of the stem
	5	A B A I S	5º Composition of tense + person (2º of plural)
	6	P I C A B A	6º Composition of stem + tense (imperfect)
RES	7	P I C A B A I S	7º Word recognized = stem + tense + person

INPUT = D U R A R E

	1	E	1º The stress vowel
	2	R E	2º Correlationof the stress vowel with the previousc one tense = future, person = 1º of the singular
	3	A R E	3º Composition class of the verb + tense + person
	4	D U R	4º Determination of the stem
RES	5	D U R A R E	5º Word recognized = stem + tense + person

We need to remind here that the system has no linguistic information at all. It doesn´t know what is a letter, a vowel, a consonant, a verb, a tense, or a person. The system only has determined a certain number of classes that he considers successively before stating what is the input. The most astonishing is that the system, without any kind of linguistic information, considering the different possibilities, makes a morphological analysis and compose the stem, the tense and the person like in a conjugation. The system has made decomposition of the verbs in morphological subunits and has learnt their conjugations.

7. CONCLUSIONS : THE FUTURE.

To summarize we can say that:

- The model proposed makes a DECOMPOSITION of each input pattern in its parts, NOT JUST A CLASSIFICATION.
- Categories learnt represent STRUCTURAL PROPERTIES of the input patterns.
- Each category is a subunit in which the input patterns can be devided.
- The performance of the model proposed is much better than the previous ones, since it extracts ALL THE CONTEXTUAL STRUCTURES, not only a small part.
- The search process of winning-resetting goes from general to particular, and can be viewed as a MORPHOLOGICAL ANALYSIS of each input pattern.
- The process of analysis is learnt in an UNSUPERVISED, CONTEXT-DEPENDANT way.
- The model has scaled a level in abstraction, from phonology to morphology.

Succesive applications of the model to lexicon would permit some extention from morphology to syntax, from syntax to semantics, ...

REFERENCES.

CARPENTER,G.A. and GROSSBERG, S. (1987), A massively Parallel Architecture For a Self-Organizing Neural Pattern Recognition Machine. Computer Vision, Graphics and Image Processing, 37, pp 54-115, 1987.

CARPENTER, G.A. and GROSSBERG, S. (1988), The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network. Computer, Vol 21, Nº3, pp77-90, 1988.

FUKUSHIMA, K. (1988). Neocognitron, A hierchichal Neural Network Capable of Visual Pattern Recognition. Neural Networks, Vol 1., pp. 119-130.

GALINDO, P. and MICHAUX, T. (1989) Implementation of Competitive Models for High Level Speech Processing. NEURO-SPEECH. Edinburgh. May 17, 18, µ 19th 1989.
To appear in:
Speech Communication, Special issue on Neural Networks Guest Editor: R. Moore

GROSSBERG, S. and STONE, G. (1986). Neural Dynamics of word Recognition and Recall: Attentional Priming, Learning and Resonance. Psychological Review, Vol. 93, N.1, pp 46-74. 1986

MICHAUX, T. and GALINDO, P. (1989). High Level Speech Processing By Competitive Neural Networks: From Psychology to Simulation.
To appear in
Fogelman Soulié, F. (under press)
Neurocomputing: Algorithms, Architectures and Applications. Berlin. Springer Verlag.

RUMELHART, D. and ZIPSER, D. (1985) Feature Discovery by Competitive Learning. Cognitive Science, 9, 75-112, 1985.

MC CLELLAND, J, and RUMELHART, D. On learning the past tense of english verbs in MC CLELLAND, J, and RUMELHART, D. Parrallel Distributed processing Vol,2 . Cambridge, MA. M.I.T. Press.

MOTOR PROGRAMS AND SENSORIMOTOR INTEGRATION

J.C. HOUK, A. BARTO, L.N. EISENMAN, J. KEIFER, S.P. SINGH, T. SINKJAER, and D. VYAS

Northwestern University , Dept. of Physiology, 303 E. Chicago Avenue, Chicago, IL 60611; University of Massachusetts, Dept. of Computer & Information Science, Amherst, MA 01003

An experimental analysis of neural signal transmission through the cerebellum and rubrospinal pathway (Houk and Gibson 1987) has led to the hypothesis that central motor programs are produced by an array of adjustable pattern generators in the cerebellum (Houk 1987). The present paper summarizes theoretical and experimental work that was stimulated by this concept.

We are in the process of developing an adaptive sensorimotor network that will be based on the anatomy and physiology of the cerebellorubrospinal circuit. Our first step has been to implement a simulation model of a single adjustable pattern generator capable of producing simple motor programs (Houk et al. in press). In this model, the fundamental driving force for the generation of a motor program is positive feedback in a loop between the cerebellum and the red nucleus. Inhibition from cerebellar Purkinje cells is postulated to sculpt motor programs out of this tendency for sustained discharge. Motor programs are stored by adjusting the synaptic weights of parallel fiber inputs onto Purkinje cells using training signals transmitted by climbing fibers.

Assuming that a set of synaptic weights has already been learned, the retrieval and execution of a motor program works as follows. In a preparatory period, specific patterns of parallel fiber signals turn Purkinje cells on and off in a bi-stable manner; in this way a spatial pattern of Purkinje cell states is preselected before the movement begins. A motor program is started by a trigger signal (sensory cue) that initiates regenerative positive feedback in the loop. The amplitude and time course of the program is then governed by the inhibitory input to the loop from Purkinje cells which serves to regulate the intensity of the positive feedback. This regulatory mechanism is controlled in a unique, quasi-feedforward manner that avoids problems of instability inherent to negative feedback systems. The feedforward operations are specified by the preselected pattern of Purkinje cell states and by an internal loop that transmits an efference copy signal representing anticipated limb position. The limited feedback operation uses sensory input that monitors actual limb position to switch Purkinje cells to their on-states, thus terminating the program.

Recently we have begun to explore how adjustable pattern generators might be trained to use force signals in combination with position signals to permit adaptation to different loading conditions. This extension of the simulation model will allow us to explore mechanisms that the brain might use for the formation of internal models of the loading characteristics of the environment.

A more global theory of sensorimotor integration has been proposed by including loops between the motor cortex and cerebellum in the model (Houk 1989). The motor cortex is considered to function as a combinatorial map linking a large number of potential sensory cues to a large number of potential actions. Positive feedback in corticocerebellar loops is thought to bind the combinatorial map in motor cortex to motor programs stored in the cerebellum. As in the rubrocerebellar model, execution of a program is triggered when a sensory input succeeds in initiating regenerative loop activity. The spatial and temporal features of the program are then controlled by inhibitory inputs to the loops from Purkinje cells in the cerebellar cortex. This model provides a new perspective for the interpretation of certain psychological phenomena such as reaction time and stimulus-response compatibility. The model also suggests a potential neural mechanism for the reported mental rotation of neuronal population vectors in the motor cortex (Georgopoulos et al. 1989).

We have begun to test the cerebellar pattern generator concept experimentally using an *in vitro* brainstem/cerebellum preparation from the turtle (Keifer and Houk 1989a). This isolated neuronal network preserves the synaptic connections between the cerebellum, red nucleus and reticular formation. Brief electrical stimuli elicit long-duration bursts of red nucleus discharge that appear to represent fictive motor programs. Our results suggest that positive feedback contributes importantly to the generation of burst discharge (Houk and Keifer 1989) and further support a role for several types of excitatory amino acid receptor (Keifer and Houk 1989b). An activity-dependent fluorescent dye has been used to visualize the neural circuit generating motor programs (Houk et al. 1989).

While many features of the proposed theory of sensorimotor integration remain speculative, mechanistic aspects of the model are testable in the *in vitro* network and functional aspects can be evaluated by computer simulations coupled with microelectrode recordings in awake animals. Experiments guided in this manner by specific models may help us to understand how central motor programs are represented in the brain.

Acknowledgement. This work was supported by ONR contract N00014-88-K-0339 and NIH grant R01-NS21015.

REFERENCES

Georgopoulos, A.P., Lurito, J.T., Petrides, M., Schwartz, A.B., and Massey, J.T. Mental rotation of the neuronal population vector. *Science*. **243**: 234-236, 1989.

Houk, J.C. Cooperative control of limb movements by the motor cortex, brainstem and cerebellum. In: *Models of Brain Function*. Edited by R.M.J. Cotterill. New York, NY: Cambridge University Press, 1989.

Houk, J.C. Model of the cerebellum as an array of adjustable pattern generators. In: *Cerebellum and Neuronal Plasticity*. Edited by M. Glickstein, C. Yeo, and J. Stein. New York, NY: Plenum Press, 1987, p. 249-260.

Houk, J.C. and Gibson, A.R. Sensorimotor processing through the cerebellum. In: *New Concepts in Cerebellar Neurobiology*. Edited by J.S. King. New York, NY: Alan R. Liss, Inc., 1987, p. 387-416.

Houk, J.C. and Keifer, J. Model of cerebellum as an array of adjustable motor pattern generators: microelectrode studies using the isolated turtle brainstem-cerebellum. In: *Neural Networks: From Models to Applications*. Edited by L. Personnaz and G. Dreyfus. Paris: IDSET, 1989, p. 121-130.

Houk, J.C., Keifer, J., and Vyas, D. Activity patterns in the cerebellorubrospinal pathway of the *in vitro* turtle hindbrain revealed with activity-dependent uptake of fluorescent dye. *Soc. Neurosci. Abs*. **15**: 612, 1989.

Houk, J.C., Singh, S.P., Fisher, C., and Barto, A.G. An adaptive sensorimotor network inspired by the anatomy and physiology of the cerebellum. In: *Neural Networks for Control*. Edited by W.T. Miller, R.S. Sutton, and P.J. Werbos. Cambridge, MA: MIT Press, in press, Chapter 14.

Keifer, J. and Houk, J.C. An in vitro preparation for studying motor pattern generation in the cerebellorubrospinal circuit of the turtle. *Neuroscience Letters*. **97**: 123-128, 1989a.

Keifer, J. and Houk, J.C. Burst generation in red nucleus is blocked by excitatory amino acid receptor antagonists in the *in vitro* turtle brainstem. *Soc. Neurosci. Abs*. **15**: 390, 1989b.

COMPUTATION OF PATTERN PRIMITIVES IN A NEURAL NET FOR ACOUSTICAL PATTERN RECOGNITION.

Paul Mueller

Department of Biochemistry and Biophysics,
University of Pennsylvania, Philadelphia, PA 19104-6059 USA.

This talk deals with fundamentals of pattern analysis in neural systems and their application to the analysis and recognition of acoustical patterns. For further details see ref. 1,2 and3.

Pattern properties.

Patterns can be defined as distributions of energy in space and time. The distributions form quantities of the three domain variables: Energy (E), Space(S) and Time (T) which exist as a seven dimensional manifold (E_1, S_{0-3},$T_{0,1}$). Spatial and temporal variables such as position, length, duration, etc. are generated by variations of energy in space or time, defining boundaries and positions of spatial objects or temporal events.

Any pattern can be described by and decomposed into a set of low order relations among these variables and may involve single or multiple domains and dimensions. The first order variables and low order relations are called pattern primitives.

Some degree of decomposition into primitives is indispensible for the analysis, recognition and understanding of patterns. In biological sensory systems it is implemented extensively at the early stages of processing. For example, in the retina and primary visual cortex multidomain and multidimensional primitives such as contrast, position, length, width, orientation and curvature of boundaries, edges, lines or areas are computed and represented by the activity of different neurons or neuron populations.

Relational primitives have both a quantitative and a logic content. Statements such as "larger than", "before", "longer than", etc. describe relations between domain quantities and have both a quantitative content (how much is A larger than B) and a binary truth value. They are commonly called elementary propositions[3] and are not to be confused with Boolean statements such as "not", "and", "or" and their derivatives.

Because of the dual aspects of the primitives, circuits that perform pattern analysis must not only be able to recognize, i.e., make logic (binary) decisions about the presence or absence of a particular pattern or primitive but must also provide a measure and resentation of its quantitative aspects. Because of their threshold, neurons can combine logic and linear operations within the same unit. For example, a unit that computes the difference of two input values, A and B, fires only if A minus B exceeds the threshold indicating a positive truth value for statement "A larger than B" and at the same time transmits the quantitative value of the difference between A and B. This mode of **transparent computation** makes neurons ideal elements for pattern analysis.

Representations, Measures and Transforms

The quantities to be analyzed, be they variables of a single domain such as length or duration or more complex relations must be measured and expressed in a computable form. In neuron assemblies the most common and efficient representation are in the form of the potential (E) and- or a position in space. Not only are different energy forms entering the system transformed and expressed as potentials (or firing rates), but quantities of space and of time as well as other primitives and more complex features such as motion, orientation etc. are usually represented either as values of the output potential or mapped into positions (i.e., activity of individual neurons) within a group of units. For example, the orientation of edges or lines is mapped into different neurons at different spatial positions in the visual cortex.

Although the point (S_1) representation has the largest dynamic range, in principle all other dimensions of the space or time domains could serve as representation. In some cases it may be advantageous to transform one type of representation in another; there is a complete set of transforms between representations in all domains and dimensions. The transformation of the output amplitude of one neuron into the spatial extension of activity within a set of other neurons is a simple example. Most of these transforms are arbitrary, others, such as the transform of time into potential are computationally very useful and often observed in the biological system. Transform operations are not restricted to single domain variables, but can be performed also on two-domain primitives such as dE/dS, dE/dt and dS/dt as well as on more complex features or even entire patterns.

Neural circuits for extraction, representation and representational transformation of low order primitives are relatively simple and can be designed in a straightforward manner without the use of learning procedures. A circuit that transforms the rate of motion of activity (dS/dt) from one neuron to another into the output level of a third neuron such that the potential is only proportional to the direction and rate of motion and not to the value of E in one or both of the input neurons involves only a few neurons and a limited number of connections.

Domain Normalization and Invariance

Patterns often contain features that form an invariant core while others are irrelevant and variable. The spatial pattern of activity within a neuron assembly may be independent of the amplitudes and temporal variations of this activity; a neural circuit that analyzes or recognizes only the spatial aspects of the pattern should be able to do so independent of variations in the other domain quantities. This requires that the spatial pattern be normalized with respect to potential and time. Conversely, the temporal aspects of a three-domain pattern may have to be separated from variations in the potential and space domains.

Application to acoustical pattern analysis.

The early stages of the higher vertebrate auditory system transform acoustical signals into patterns of neural activity distributed in space and time and decompose the relations between these variables into their pattern primitives. We have constructed an electronic analog neuron net for phonetic speech recognition in which a section of the net performs this pattern decomposition as an initial step for subsequent processing[3]. At the input to the net sound is decomposed into its different frequencies by a set of band pass filters. The rectified filter output is fed into several hundred neurons, each of which is tuned to a particular pattern primitive such as fundamental frequency bands associated with speech sounds - so called formants, the amplitude changes at particular frequencies, and temporal changes of frequency. The performance of this section of the net is modeled after the dorsal cochlear nucleus of higher vertebrates and the different functions of individual neurons are essentially identical to those of representative units in the dorsal cochlear nucleus[5], although the number of units is much smaller.

A second set of neurons decodes the output patterns of the first set in such a way that individual neurons respond to different phonemes within a word and in conjunction with a computer terminal provide a real-time phonetic printout of speech.

The temporal aspects of patterns are particularly important for acoustical patterns and particularily for speech and this discussion deals mainly with those aspects of the initial pattern decomposition that involve time as an explicit variable[1,2,3].

The first of the temporal pattern primitives is the change of energy (E) with respect to time ($\pm$dE/dT) at constant spatial position (frequency). Positive and negative values of dE/dT are represented by outputs from separate neurons ("ON" and "OFF" units). This computation is achieved by a combination of direct and delayed (low pass filtered) inputs of opposite polarity from frequency tuned units to the "ON" or "OFF" units. For the "ON" units the excitatory connection is direct whereas the inhibitory input is low pass filtered. The "OFF" units receive inputs of opposite polarity.

These "ON" and "OFF" units represent not only the temporal boundaries of events, i.e., beginning and end of a particular sound of a certain frequency, but also provide a measure of time from the temporal boundary by transforming time into an exponentially decaying potential, the latter aspect being particularly useful for computing the duration of sounds or silences at a later stage.

The second temporal primitive extracted by the net is the change of frequency that occurs for example during formant transitions associated with certain diphones. Since each frequency band is represented by the activity of a separate neuron group, changes of frequency appear as "motion" of activity in neuron space, i.e., $\pm$dS/dT. We define this motion as the decay of activity at one position and increase of activity at a neighboring position and compute the rate and direction of motion (increase or decrease of frequency) by neural "AND" gating of the outputs from neighboring "ON" and "OFF" units. This "AND" gating is achieved by implementing a

"NOT(NOT)" function of the "ON" and "OFF" units through separate representation of, and inhibition by their complementary output[1]. The motion-tuned neurons are sensitive to direction, amplitude and rate of motion. The range of rate sensitivity is determined by the time constants of the "ON" and "OFF" units.

A third temporal primitive represented in the net is the temporal sequence of activity between selected neurons. This feature, although related to motion, does not include the rate of transition of activity from one position to another as a variable. Thus a neuron that is tuned to a particular sequence of activity in two neurons will fire independent of the delay between the two events. This is achieved by latching the output of the "OFF" representation of the first unit and "AND" gating it with the "ON" representation of the second unit.

Finally, there are units that are tuned to a specific duration of activity. This is implemented by summing low pass filtered inhibition from the input unit and low pass excitation from its "OFF" unit to a duration-tuned unit. If the time constant of the inhibition is larger than that of excitation, the range of durations the unit is tuned to is determined by the difference of the time constants, the synaptic gains and the threshold.

It should be noted that in our net motion, sequence and duration as well as more complex temporal pattern primitives are computed through hierarchical inputs from "ON" and "OFF" units and that such units are also found at the early stages of biological neural systems.

REFERENCES:

1. **Mueller, P., Martin, T. and Putzrath, F.** General Principles of Operations in Neuron Nets with Application to Acoustical Pattern Recognition. **Biological Prototypes and Synthetic Systems**, Vol.1, E.E. Bernard and M.R. Kare editors, p. 192-212, Plenum Press, New York, 1962.
2. **Mueller, P.,** Principles of temporal pattern recognition in artificial neuron nets. **Artificial Intelligence**, S-142, 1963, The Institute of Electrical and Electronic Engineers, Inc., New York.
3. **Whitehead, A.N. and Russel, B.** Principia Mathematica, Vol.1. Cambridge U. 1910.
4. **Mueller, P. and Lazzaro, J.,** A machine for neural computation of acoustical patterns with application to real time speech recognition, **AIP Conference Proceedings,** 151:321-326, 1986.
5. **Kaltenbach, S.A. and Saunders, Y.A.** Spectral and Temporal Response Patterns of Single Units in Chinchilla Dorsal Cochlear Nucleus, **Exp. Neurology,** 96:406, 1987.

Supported in part by grants from ONR (#N00014) and NSF (EET87-16685)

MODELING OF SPATIAL TRANSFORMATIONS IN VESTIBULAR REFLEX SYSTEMS

Barry W. Peterson, Northwestern University Medical School, Chicago, IL 60611.

Summary *Experimental and theoretical approaches to understanding how the CNS converts patterns of sensory input into patterns of motor activity are described. A key is understanding how sensorimotor transformations can take place in the non-orthogonal, multidimensional coordinate systems that occur in nature. Tensor network theory can describe such transformations and generate predictions that can be tested experimentally. Experiments have confirmed the theory's predicted input/output transformations for the vestibulocollic reflex but more detailed predictions of neuronal substrates of the vestibuloocular reflex (VOR) have to be modified to account for specialized connections that have evolved in this system. Adaptive plasticity of the VOR poses a challenge for future modeling efforts.*

Introduction. The goal of the experimental-theoretical efforts described here is to understand how the brain implements sensorimotor transformations in real biological coordinate systems that are determined by the anatomy of sense organs and musculo-skeletal linkages in the body. Unlike the familiar orthogonal coordinate systems utilized by engineers, these biological coordinate systems are typically non-orthogonal and have an arbitrarily large number of dimensions that correspond to the axes of sensitivity of sensors or pulling action of muscles. In considering such coordinate frames, it is important to realize that a given invariant, such as a stimulus or the response that it elicits, may be represented in two distinctly different ways. Using the nomenclature adopted by Pellionisz and Llinas (1979,1980) in their tensorial modeling, these representations are termed covariant and contravariant. A sensory stimulus will typically be represented in a *covariant or projection form.* As illustrated in Fig. 1A, this is found by projecting the invariant (eg the end point of a vector representing a rotation applied to the head) upon each of the axes in the coordinate frame. This is exactly what happens in the vestibular labyrinth where the response of each semicircular canal (SCC) to a head rotation is equal to the projection of that rotation upon the axis of maximum sensitivity of that canal. Note that the covariant representation is always unique regardless of the coordinate frame.

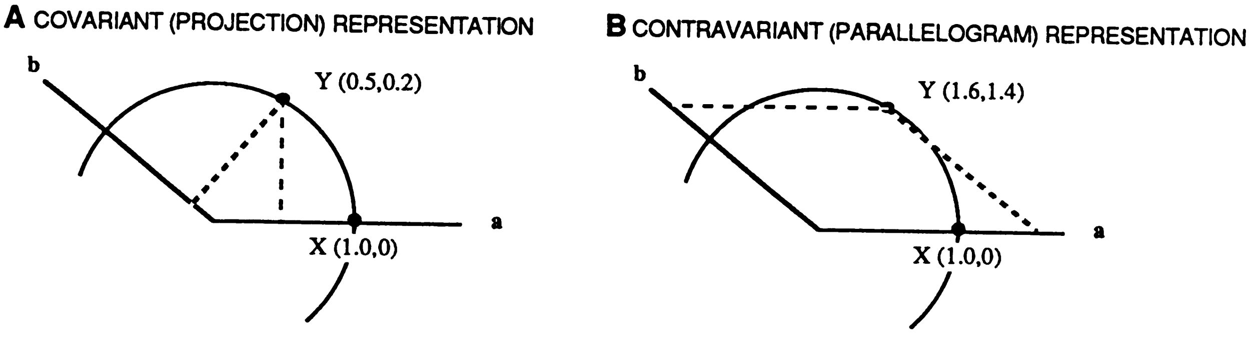

A COVARIANT (PROJECTION) REPRESENTATION
b
Y (0.5,0.2)
a
X (1.0,0)
B CONTRAVARIANT (PARALLELOGRAM) REPRESENTATION
b
Y (1.6,1.4)
a
X (1.0,0)

FIGURE 1. Representations in non-orthogonal coordinate frames. Two points X and Y are shown represented in a 2-dimensional coordinate frame with 135° angle between axes. A. Covariant representation is formed by projecting X and Y on axes **a** and **b**. B. Contravariant representation is the components along axes **a** and **b** which, when added vectorially, produce X and Y. The actual values of coordinates of X and Y are shown in parentheses.

A motor response will typically be represented in a *contravariant or parallelogram form.* As illustrated in Fig. 1B, this corresponds to those components along each axis which when summed vectorially produce the invariant (eg the required net torque on a given body segment). Note that this representation is not unique if the number of axes exceeds the number of degrees of freedom of the final response. In the figure the use of two axes in a two-dimensional space results in a unique representation but if a third axis were added, a given point could have an infinite number of contravariant representations. One problem to be understood by modeling is how the brain chooses a response in such cases where the number of effectors exceeds the number of degrees of freedom. The figure illustrates another feature of contravariant representations in non-orthogonal coordinate frames – the component along a given axis is typically not maximal for responses that are aligned with that axis (eg **Y** has a larger component along **a** than does **X**). Thus we should not expect a muscle to be maximally activated when movement occurs along the line of its best pulling action.

Motor Commands Producing Head Movement. To explore how the CNS deals with the complexity of the spatial transformations discussed above, my colleagues and I have examined the patterns of motor activity that underlie head and eye movements. The former are especially interesting since there are 30 muscles that act to move the head whereas fluoroscopic studies (Vidal et al., 1986) have shown that normal head movements involve perhaps 6-10 degrees of freedom. Correspondingly, Keshner et al. (1986) found that when cats are trained to follow a moving water tube, each animal uses a different pattern of muscle activity to generate the same set of head trajectories. A different picture emerges, however, when one analyzes muscle patterns that underlie the vestibulocollic reflex (VCR), a reflex that stabilizes the head by generating rotations that compensate for rotational perturbations sensed by the SCCs. In this case there is no significant difference between the motor patterns employed by different animals (Baker et al., 1985; Banovetz et al., 1987). This stereotypy suggests that CNS processing is organized around some optimizing strategy. Modeling efforts described below have shown that this strategy corresponds quite closely to that suggested in a theoretical paper by Pellionisz (1984).

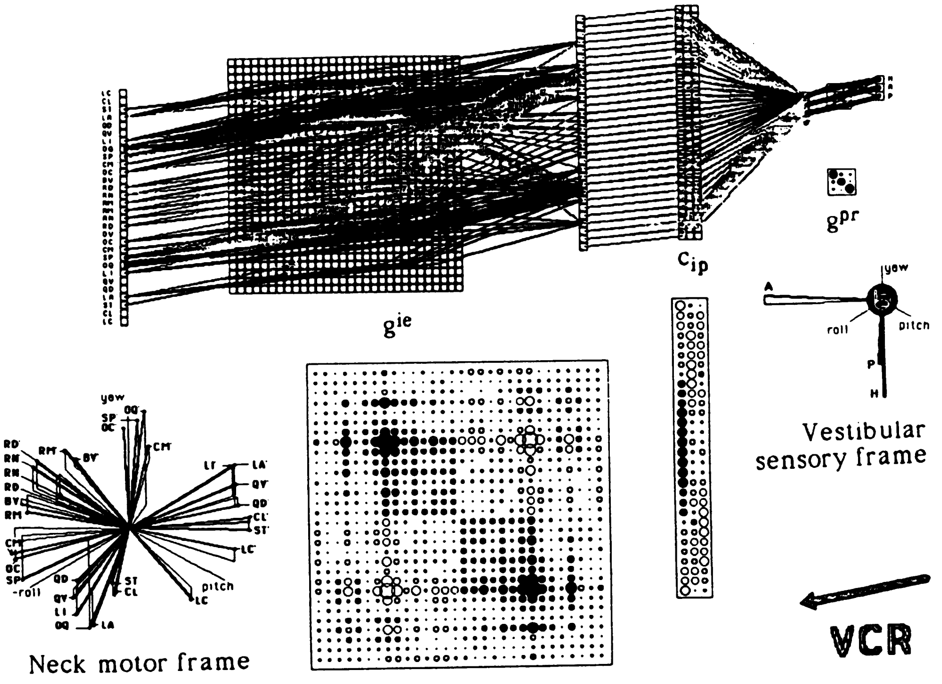


FIGURE 2. Tensor network model of VCR in the cat. The sensorimotor transformation in the model occurs in 3 stages: g^{pr}, sensory metric tensor, c_{ip}, sensorimotor embedding tensor and g^{ie} neck motor metric tensor. These tensorial operations are expressed in the reference frames of the VCR by matrices, which are here represented by patch diagrams in which positive and negative components are shown by filled and open circles whose areas are proportional to the size of the component. The CNS can implement these matrices by neuronal networks as indicated in the upper part of the diagram where size and strength of connections are indicated by line thickness and shading (black for excitation, grey for inhibition). From Pellionisz and Peterson (1988).

Tensorial Model of the VCR. We know that the VCR must transform an input (representing the rotation the experimenter applies to the head and body) that is covariantly coded in the coordinate frame of the SCCs into an output (activation of the 30 neck muscles) that is contravariantly coded in the coordinate frame of the neck muscles. The model (cf: Pellionisz and Peterson, 1988) implements this transformation in 3 stages shown in Fig 2. These are:

1. Sensory metric tensor. This transforms the incoming covariant signal in the SCC coordinate frame into a contravariant representation in that frame. Since the 6 canals act as 3 pairs coding movement in 3-dimensional space, the 3X3 matrix required for this transformation may be found by simply inverting the matrix consisting of cosines of angles between the 3 SCCs.

2. Sensorimotor embedding tensor. This projects the 3 independent components of the contravariant SCC signal upon the axes of the 30 neck muscles. It is just a 3X30 matrix of cosines of angles between canal sensitivity axes and muscle pulling directions.

3. Neck motor metric tensor. The covariant neck motor signal resulting from 2 must be transformed into the contravariant representation required at the output of the VCR. As at stage 1, what we require is the inverse of the matrix [M] of cosines of angles between axes of the coordinate frame. Here, however, this is a 30X30 matrix which has no unique inverse. Therefore the model utilizes the Moore-Penrose Generalized Inverse (MPGI) as proposed by Pellionisz (1984). Mathematically (cf: Albert, 1972) the components of the MPGI are obtained from the outer (dyadic matrix) product of the eigenvectors of [M]. Biologically, it could be generated in a plastic neural network by the meta-organization process described by Pellionisz and Llinas (1985). From the point of view of mechanics, the MPGI is an efficient choice since it tends to minimize co-contraction of muscles with opposing actions thus preventing wasteful expenditure of energy.

As shown in Fig. 3, the model succeeds in predicting the motor pattern of the VCR within experimental error. This indicates that the choice of the MPGI as an optimizing principle approximates the kinematic selection process embodied in the VCR circuitry. The model also suggests what some of the intermediate signals present in that circuitry could be. For instance, even though the patterns of activity of neck muscles are contravariant and therefore do not exhibit signals that are maximal along the pulling directions of neck muscles, the model suggests that covariant signals, which would be maximal along those directions should be found in the CNS. We are now searching for such signals in the vestibular nuclei where SCC afferents project upon neurons that send them onward to the cervical spinal cord and other brainstem centers. It is of course possible that other intermediate signals could appear if CNS processing is constrained by other factors as it is in the case of the vestibuoocular reflex (VOR), which is discussed next.

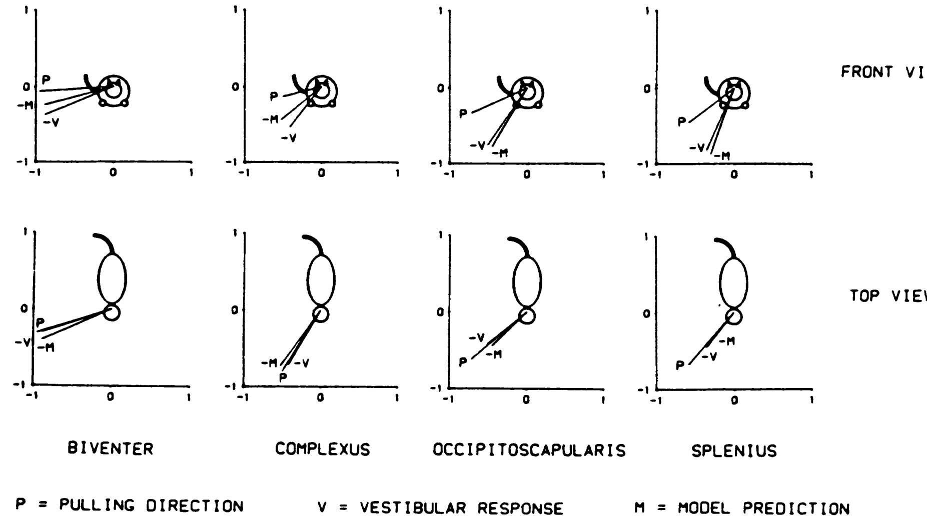


FIGURE 3. Comparison of pulling directions (P), directions of head rotation that excite maximal VCR activation (V) and model predictions of V (M) for four neck muscles. In each case rotation directions are expressed as unit vectors (use the right hand rule to find rotation corresponding to each vector). The V and M vectors have been reflected (to -V and -M) to facilitate comparison with P vectors.

Spatial Transformations in the VOR. The VOR is an attractive subject for studying spatial transformations since a large amount of its output is generated by a very simple 3-neuron arc which includes only SCC afferents, motoneurons and a single interneuron, the vestibuloocular relay neuron (VORN). It is also simpler than the VCR since there are only 6 extraocular muscles rather than 30 neck muscles. Recent studies have thoroughly characterized the fashion in which the 3-neuron arc implements the spatial transformations required in the VOR of decerebrate (Peterson et al., 1987) and alert (Perlmutter et al., 1988) cats.

Our experimental work was spurred by a tensorial model of the VOR put forth by Pellionisz and Graf (1987). Since they wished to model the action of the 3-neuron arc, the three transformations of the typical tensorial model (like that of the VCR model described above) were reduced to two by combining stages 1 and 2 into a single matrix. The major spatial transformations of the VOR are between the axes of the vertical SCCs and the oblique and vertical rectus (VR) eye muscles, which are misaligned by and degrees respectively. In the model these directional shifts occur in the initial transformation, which should be between SCC afferents and VORNs. (Such a transformation would occur if more than one SCC acted on a given VORN as had been reported by Baker et al., 1984). A difficulty was that electrophysiological data suggested that a large part of the transformation between SCCs and VR muscles should arise from the fact that VORNs project to VR motoneurons on both sides of the brain rather than on one side as in the case of oblique and horizontal rectus motoneurons. This projection pattern may have arisen as the CNS adjusted to evolutionary changes from lateral to frontal eyes. Experimental work has provided a definitive answer. The model is right in predicting that the transformation required for activation of oblique muscles occurs prior to VORNs but is not correct in predicting that the transformation between SCCs and VR muscles occurs there. This appears to rely on the aforementioned bilateral projections of VORNs. Thus it appears that an imperative introduced by evolutionary change has overridden the predicted pattern of transformation for this portion of the VOR.

Motor Learning in the VOR. The VOR poses another interesting challenge for neural modeling. Its input/output transformation can be markedly altered by motor learning. My colleagues and I have found that the direction of the vestibuloocular eye movement evoked by a head rotation can be changed by a few hours of pairing of head rotation in one direction with motion of the visual world in an orthogonal direction (Harrison et al., 1986; Baker et al. 1986). Such pairing causes the VOR, tested in total darkness, to acquire, with a time constant of ~30 min, a new component in the direction of the visual world motion. This means that terms in the transformation matrices describing the VOR can undergo adaptive change under the guidance of visual feedback. These changes show amazing specificity. For instance, if the pairing is done at a single frequency, the adaptation will be tuned to that frequency with the learned component of the VOR falling off like a band-pass filter when tested at higher or lower frequencies. If the subject is trained while lying right ear down, the adaptive response will be maximal in that position and much smaller in the right ear down position even though the SCC activation produced by the horizontal rotation stimulus is identical in the two cases (Baker et al, 1987). One can even train the VOR to produce oppositely directed learned components in the two positions (Baker et al., 1988)! A challenge for the future is to devise models that can exhibit such context and frequency dependent learning in the sort of simple networks that are present in the VOR.

References.

Albert, A. (1972). Regression and the Moore-Penrose Pseudoinverse. Acad. Press, New York.

Baker, J., Goldberg, J., Hermann, G. and Peterson, B.W. (1984). Optimal response planes and canal convergence in secondary neurons in vestibular nuclei of alert cats. Brain Res. **294**: 133-137.

Baker, J., Goldberg, J., and Peterson, B. (1985). Spatial and temporal response properties of the vestibulocollic reflex in decerebrate cats. J. Neurophysiol. **54**: 735-756.

Baker, J., Harrison, R.E.W., Isu, N., Wickland, C. and Peterson, B. (1986). Dynamics of adaptive change in vestibulo-ocular reflex direction. II. Sagittal plane rotations. Brain Res. **371**: 166-170.

Baker, J. F., Perlmutter, S. I., Peterson, B. W., Rude, S. A. and Robinson, F. R. (1988). Simultaneous opposing adaptive changes in cat vestibulo-ocular reflex direction for two body orientations. Exp. Brain Res. **69**:220-224.

Baker, J., Wickland, C. and Peterson, B. (1987). Dependence of cat vestibulo-ocular reflex direction adaptation on animal orientation during adaptation and rotation in darkness. Brain Res. **408**:339-343.

Banovetz J.M., Rude S.A., Perlmutter S.I., Peterson B.W., Baker J.F. (1987) A Comparison of Neck Reflexes in the Alert and Decerebrate Cats. Soc Neurosci. Abstr. **13**:1312

Harrison, R.E.W., Baker, J.F., Isu, N., Wickland, C.R. and Peterson, B.W. (1986). Dynamics of adaptive change in vestibulo-ocular reflex direction. I. Rotations in the horizontal plane. Brain Res. **371**: 162-165.

Keshner EA, Baker J, Banovetz J, Peterson BW, Wickland C, Robinson FR, and Tomko DL (1986). Neck muscles demonstrate preferential activation during voluntary and reflex head movements in the cat. Soc. Neurosci. Abstr. **12**: 684.

Pellionisz A. J. (1984). Coordination: A vector-matrix description of transformations of overcomplete CNS coordinates and a tensorial solution using the Moore-Penrose generalized inverse. J. Theor Bio **101**: 353-375.

Pellionisz A. J. and Graf, W. (1987). Tensor network model of the "three-neuron vestibulo-ocular reflex-arc" in cat. J. Theoret Neurobiol. **5**: 127-151.

Pellionisz A. J. and Llinas, R. (1979). Brain modeling by tensor network theory and computer simulation. The cerebellum: Distributed processor for predictive coordination. Neuroscience **4**: 323-348.

Pellionisz A. J. and Llinas, R. (1980). Tensorial approach to the geometry of brain function. Cerebellar coordination via metric tensor. Neuroscience **5**: 1125-1136.

Pellionisz A. J. and Llinas, R. (1985). Tensor network theory of the meta-organization of functional geometries in the CNS. Neuroscience **16**:245-274.

Pellionisz, A. J. and Peterson, B. W. (1988). A tensorial model of neck motor activation. In: Peterson, B.W. and Richmond, F.J. Control of Head Movement. Oxford Univ. Press, New York, pp. 178-186.

Perlmutter, S.I., K. Fukushima, B.W. Peterson, and J.F. Baker (1988) Spatial properties of second order vestibuloocular relay neurons in the alert cat. Soc. Neurosci. Abstr. **14**:331.

Peterson, B.W., Graf, W. and Baker, J.F. (1987). Spatial properties of signals carried by second order vestibuloocular relay neurons in the cat. Soc. Neurosci. Abstr. **13**:1093.

Peterson. B. W., Pellionisz, A.J., Baker, J.F. and Keshner, E. A. (1989). Functional morphology and neural control of neck muscles in mammals. Amer. Zool. **29**: 139-149.

Vidal, P.P., Graf, W. and Berthoz, A. (1986). The orientation of the cervical vertebral column in unrestrained awake animals. I. Resting position. Exp. Brain Res. **61**:549-559.

Computer Simulation of a Macular Neural Network

Muriel D. Ross*, Judith Dayhoff**, Dale Mugler***
*NASA-Ames Research Center, Moffett Field, CA 94035
**Judith Dayhoff & Associates Inc, Mountain View, CA 94043
***University of Akron, Akron, OH 44325

Our previous research demonstrated that gravity-sensing endorgans of the inner ear are organized as weighted neural networks for parallel distributed processing of linear acceleratory information. This conclusion is based upon computer-assisted, three dimensional (3-D), animated reconstructions of terminal/receptive fields of nerves ending in the sense organs, called maculas. Findings obtained from this research and from known physiology are now being implemented in dynamic, symbolic computer models to simulate a functioning system, and to learn more about the principles of organization of biological neural networks.

Our model currently encompasses only three macular nerves with their terminals and receptive fields (Figure 1). Once completely implemented, the software will permit us to simulate thousands of processing elements in any grouping desired. In order to understand the basis of our model, a brief summary of macular functional organization follows.

Sensory maculas are comprised of two types of hair cells, type I and type II, and supporting cells. A type I hair cell synapses only with the expanded nerve terminal (calyx) that surrounds its body and neck. From one to five type I cells may be enclosed by a single calyx. Type II hair cells lie outside calyces. They distribute their output to 1-4 neighboring calyces and to nerve and calyceal collaterals. Type II hair cells, calyces and unmyelinated preterminals are also postsynaptic to efferent nerve endings that are collaterals of calyces and nerves.

Hair cells have tufts of 60-80 thread-like processes (stereocilia) arranged hexagonally in rows of graded height and a single kinocilium that lies between the two tallest stereocilia. The location of the kinocilium functionally polarizes the hair cell. Neighboring hair cells are not identically polarized, and polarity also reverses direction along a traceable curved line (striola). Hair cells depolarize maximally to a stimulus in the direction of the kinocilium and hyperpolarize to a lesser degree to stimuli in the opposite direction (AJ Hudspeth and DP Corey, Proc. Natl. Acad. Sci. U.S.A. 74:2407, 1977). They do not respond to stimuli at right angles to the tuft (SL Shotwellet al., Ann. N.Y. Acad. Sci. 374: 1, 1981). The hexagonal organization likely optimizes signal detection, as is the case in man-made detectors (DP Petersen and D Middleton, Inform. and Control, 5:279, 1963). Stereocilia and kinocilia differ in thickness and in height from site to site on a macula so that neighborhoods of hair cells differ in threshold properties and in range of sensitivity to incoming stimuli. The entire macular array, then, has an overall morphology comparable to man-made phased array detectors (E Brookner, Sci. Amer. 252; 94-102,1985), and responses to changes in direction or rate of acceleration are rapid.

The sac-like membranous labyrinth above the macula is filled with endolymph, a weak gel. Interposed in this gel is a layer of crystallite particles (otoconia) that are unequally distributed in a thickened part of the gel, the otoconial membrane. The otoconial membrane is attached to the kinocilia and the tallest stereocilia by strands of gel-like material, but is relatively free to move. The maculas are attached to the bone of the skull by connective tissue. It is commonly accepted that, during accelerations to the head, the macula follows the motion of the skull but the otoconial layer lags behind, due to inertia. This results in bending of the stereociliary tufts in the direction of lag. It is possible that this concept is too simplistic, however, because of variance in distribution of otoconial mass and because of the physics of the gel and the geometry of the macula. It is more likely that wave patterns generated in the gel by linear accelerations to the head are amplified or dampened by the otoconial layer in specific ways; and that analysis of the resulting, complex waveforms by the sensory hair cells is dependent upon tuft and macular geometries.

The hair cells communicate their information to vestibular nerves that have three major kinds of terminal patterns: M-type, in which there is a single calyceal terminal at the end of a myelinated nerve; M/U-type, in which there is a short, unmyelinated preterminal segment that may bifurcate; and U-type, in which the preterminal unmyelinated segment is often long and three branches are typical. Receptive fields consist of all the type I and type II hair cells synapsing with the terminal(s) of a nerve and its collaterals. The fields tend to be rounded when there is a single calyx, oblong when there are two branches, and highly elongated when there are three branches. No two fields are identical in detail.

The neural network, although continuous, is more complexly organized in some places than in others. Near the striola, all three kinds of nerve patterns are present; in an area toward the border there are only U-type nerves with many collaterals. Lengthening of the unmyelinated preterminal segment increases delay time before propagation of an action potential can begin. This finding and another, that hair cell synapses differ in number and size and are spatially distributed, strongly suggest that spatio-temporal factors are important to macular information processing.

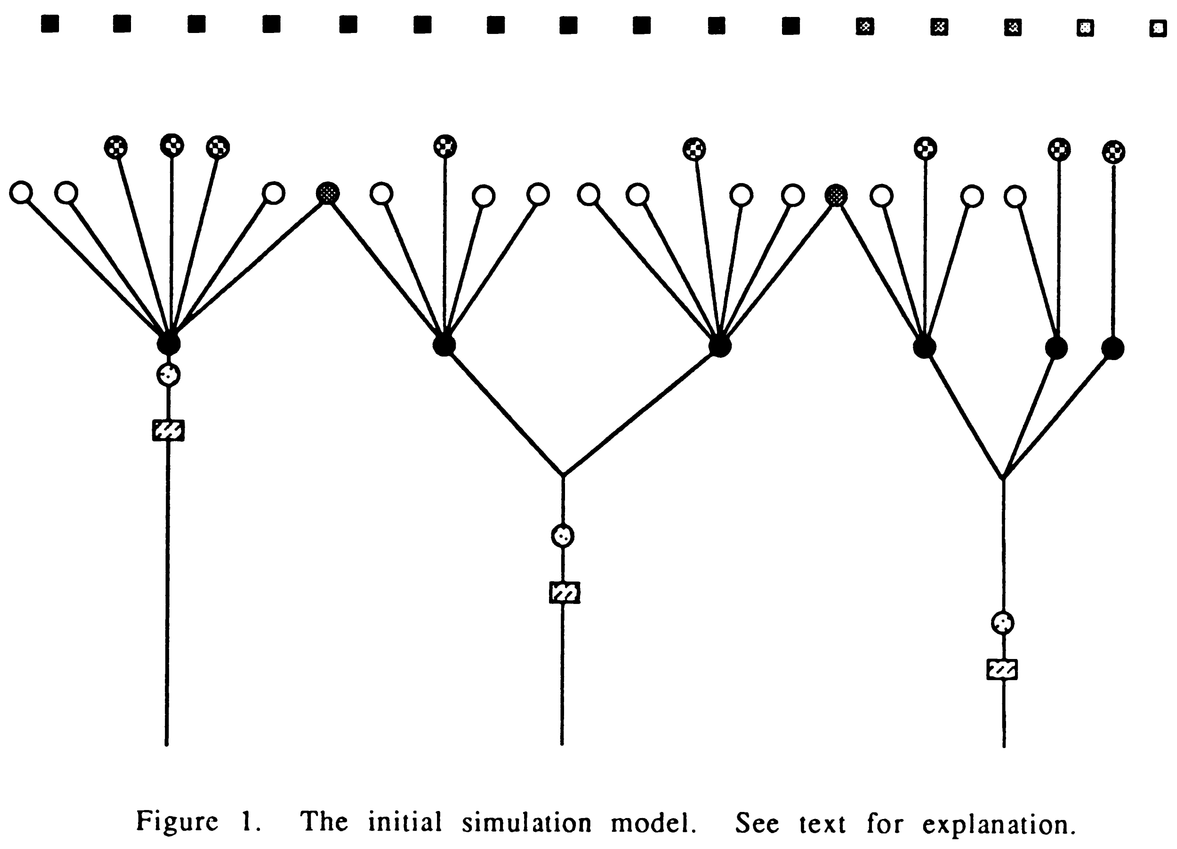


Figure 1. The initial simulation model. See text for explanation.

The full meaning of the elegant organization of the endorgans still eludes us. Maculas are geometrically organized to detect the full range of linear accelerations in their general plane and also rotation, some range of angular stimulation, and velocity. They are clearly segmented, possibly for feature extraction, and wiring patterns appear to be determined through constrained randomess. Our simulations examine these and other concepts and will be sharpened by new experimental results.

No effort has been made to duplicate the complexities of the supramacular material (gel and otoconial membrane) in our initial model, but a sequence of input vectors is simulated, with time stepped forward incrementally. The simulations are run on an IRIS 4D turbo high performance workstation. Currently, our model is two dimensional, and stimuli are presented from left to right (or vice versa) and randomly. Stimuli are simulated by activation of the squares of the first tier of our model (Figure 1). The stimulus as well as responses of all active elements are represented through color-coding (here, by shading), with blue (lightest square, right) representing inhibition or hyperpolarization and red (darkest square, left) representing maximal activation or depolarization. Green is neutral. The full range of colors in this range also depict membrane activation levels for receptor cells, calyces and spike initiation zones. Nerve impulse initiation is animated at the nerve fiber.

Type I hair cells are in the second tier of processors (checkered circles) and type II hair cells, in the third tier (Figure 1). Type II hair cells are clear circles except for cells, shown cross-hatched, that are distributing information to neighboring calyces. The two tiers reflect hair cell positions inside and outside calyces. The number of hair cells is an average of those actually observed in receptive fields of the nerve types employed in the model. Physical features of the tufts and their directional tuning are not portrayed but are assigned values in the software for simulation. Each line connecting a hair cell to a calyx is also assigned a scalar value that reflects the strength of that connection. The value varies from cell to cell because the number and positions of the synapses vary. The connection can also be assigned a value indicating that the hair cell output is excitatory or inhibitory to the calyx.

The fourth tier contains the calyces (solid circles). Calyces are processing elements that, in our model, sum excitatory and inhibitory inputs from the hair cells. The information is passed to a spike initiation zone, or encoding site, shown as a small, dotted circle on the stem axon (see Figure 1). This zone is assigned a threshold value which must be overcome for the axon to fire. Axonal discharge is indicated by color coding of the rectangle with dashes and separately as impulse trains. Our initial model consists of a simulated M-type nerve (left), an M/U-type (center), and a U-type (right).

The current model, then, includes membrane activation levels, receptor cell polarizations, and temporal decay of membrane potentials, along with interconnection strengths and topologies. Currently, each nerve has only a single spike initiation zone, appropriately located relative to the calyx for each type of nerve. A delay is required to take into account the distance between calyces and their spike initiations zones. Later simulations will test the consequences of multiple encoding sites, one for each calyx. Also, our present simulations do not yet include calyceal collaterals (synapses from a calyx back to a receptor cell) or efferent regulation. We have examined appropriate equations for incorporating each of these phenomena in future simulations.

Our aim here is to test the model with the elements presently incorporated in our simulation software, and to add other elements sequentially, to learn how each influences nerve activity. The software is written in C and is composed in a way to facilitate changing parameters and redesigning the layout and interconnection topology of the network. This freedom to change parameters has allowed us to test responses of our simple model to the following conditions: 1) all receptor cells are excitatory; 2) type I receptor cells are excitatory and type II cells are inhibitory; and 3) hyperpolarization (inhibition) of type II receptor cells that are programmed to have weak inhibitory connections to calyces.

A series of initial experiments were run in which we observed a variety of different connection strengths and different input stimuli under the above conditions. In each case we were able to examine the temporal dynamics of the resulting activation patterns. We observed a difference in responsiveness of the sensory neurons between conditions 1 and 2 (with type II cells excitatory versus inhibitory). When all cells are excitatory, the neurons saturate rapidly. The simulation remains more active and stable when type II hair cells are weakly inhibitory to the calyx. From condition 3, we observed an activating response before the strongly depolarizing signal reaches the hair cells. This response proved to be the result of well-known disinhibition (inhibition of an inhibitory unit), which is net excitation. Under all these preset conditions, random stimulation did not result in as many impulse trains as did directionally controlled stimulation. Nevertheless, some version of multiple stimuli will have to be incorporated into our model to mimic current concepts that complex waveforms are the stimuli to the hair cells.

DEFAnet - a deterministic approach to function approximation by neural networks

Wolfgang J. Daunicht
Abt. Biokybernetik, Inst. f. Phys. Biologie
Heinrich-Heine-Universitaet Duessseldorf
D-4000 Duesseldorf, F.R. Germany

Introduction

The approximation of a given continuous real function with support in the n-dimensional hypercube by means of neural networks has attracted the interest in both theory and applications. It has been shown that a feedforward network with a limited number of units in the hidden layers can not only approximate, but even represent exactly any continuous function (Kolmogorov 1957), if the only restrictions to the units' output functions are that they be monotonous and continuous. It is also known that feedforward networks with a single type of nonlinear output function can approximate a given function arbitrarily well (Lapedes and Farber 1988), even with a single hidden layer (Cybenko 1989). However, a lot of problems remain unresolved, e.g. what size of network is sufficient to approximate a given function with a given accuracy, what can be said about its inter- and extrapolation properties, and what can be said about the convergence of learning. Therefore in practical problems such as the implementation of the inverse kinematics of a manipulator in a neural network (Guez and Ahmad 1988), the structure of the network employed is often chosen somewhat arbitrarily. The present paper proposes a deterministic approach to these problems. Based upon the choice of a certain interpolation rule, a neural network concept is developed that allows to determine the topology of the network required to approximate any function and allows to find the weights of the synapses by direct calculation as well as by training.

Concept of DEFAnet

The concept of DEFAnet is based on the construction of a grid in the n-dimensional hypercube by subdividing it into n-dimensional rectangular cells by means of l_ν hyperplanes orthogonal to the ν-th axis ($\nu = 1, \ldots, n$), and the assumption that the goal function f is defined and known at least at the corners of each cell, the grid points. The choice of the interpolation rule that is to be implemented by the network is given by the condition, that all first partial derivatives are constant inside a cell and along the edge of a cell. An example of a function following this interpolation rule is given in the 2-dimensional case as an interpolation of the XOR truth table (Fig 1).

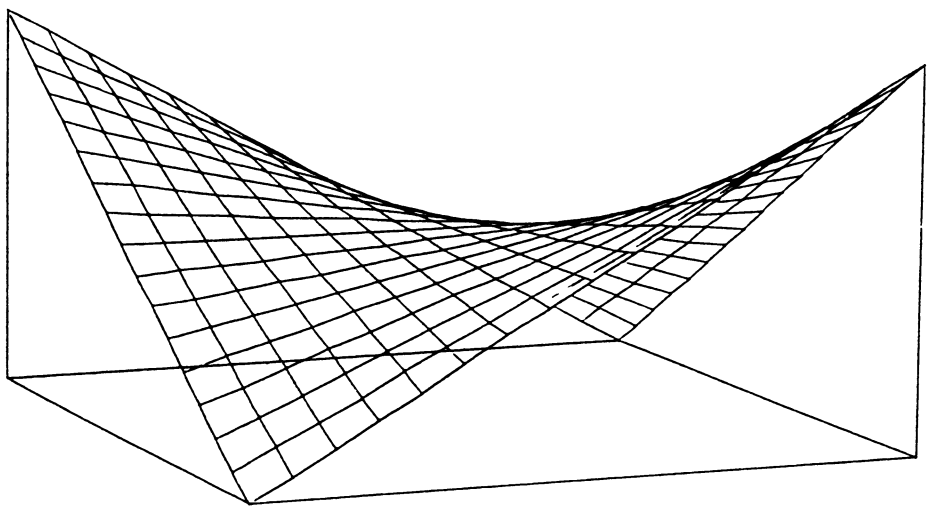


Fig. 1. Example of a function following the interpolation rule in a single 2-dimensional grid cell (XOR surface)

The neural network is constructed in such a way as to follow the interpolation rule - given any combination of function values at the corners of each cell - only by modifying the synaptic weights of the last layer. To this end it is required to have all kinds of products of relative input signals (taken to the power of 0 or 1) available, so that appropriate linear combinations of them may be formed. If this is to be achieved by a network with neurons summing rather than multiplying their inputs, a 4-layer feedforward network is required. Unfortunately, using neurons with limited non-negative output (such as natural neurons), it is not possible to generate a product directly using logarithmic output functions, summation, and an exponential function, as the logarithm of numbers less than one would generate negative signals of unlimited amplitude. However, the use of output functions that can be described as shifted logarithmic functions yields linear combinations of products that can still be employed while following the interpolation rule.

Construction and size of a DEFAnet

Based on the DEFAnet concept outlined above, the following network is proposed. The first layer consists of n pure fan-out units. The neurons of the second layer have monotonous output functions which increase logarithmically over the range between two adjacent grid hyperplanes from 0 to 1. The ranges of such increase correspond to the sizes and locations of the grid cells. To subdivide a hypercube into $\prod_{\nu=1}^{n}(l_\nu - 1)$ cells requires

$$z_{(1-2)} = \sum_{\nu=1}^{n}(l_\nu - 1)$$

synapses as well as neurons in the second layer, as each second layer neuron receives input from only one first layer neuron. It suffices to let the synapses be excitatory, non-plastic and their weight be 1.

The output functions of the third layer neurons increase proportional to the exponential function and saturate at 1. To provide all required linear combinations of products the third layer consists of $\prod_{\nu=1}^{n} l_\nu$ neurons. The number of required non-zero synaptic connections between the second and third layer is

$$z_{(2-3)} = \sum_{\nu=1}^{n} \frac{l_\nu - 1}{l_\nu} \prod_{\rho=1}^{n} l_\rho$$

Here the synapses are excitatory and non-plastic; their weights depend on the third layer neuron to which they belong, but not on the goal function.

The output function of the only fourth layer neuron is linear and not limited. If vector rather than scalar functions are considered, i.e. the output dimension m is greater than 1, this is the only layer to increase the number of units. The number of synapses is

$$z_{(3-4)} = m \prod_{\rho=1}^{n} l_\rho$$

as all third layer neurons are connected to all fourth layer neurons. The synapses between the third and the fourth layer are considered to be plastic and to have either sign. Given the output functions of the second and third layer neurons, the values of the last synaptic layer can be determined by n-fold application of a set of recursive formulae. The size of a DEFAnet and its topology of connections is completely determined by the dimensions of the input and output signals and by the resolution of the grid along each axis. An example of a DEFAnet topology is given in Fig. 2

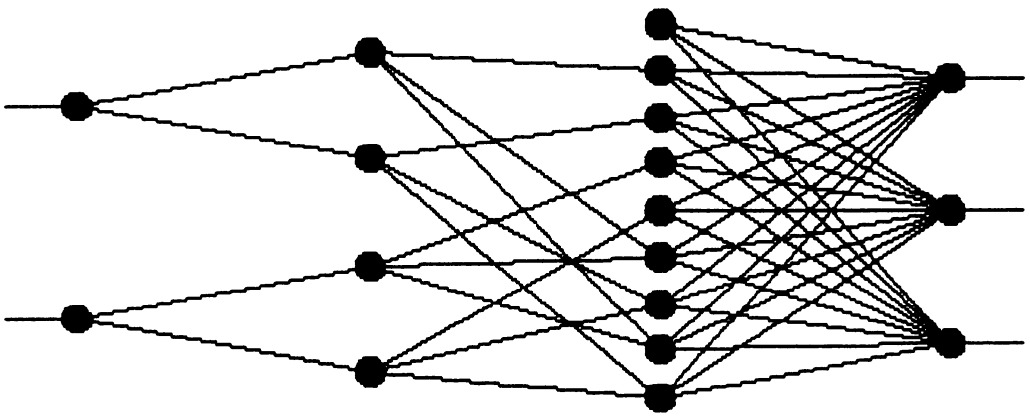


Fig. 2. Example of a DEFAnet structure for $n = 2, l_1 = 3, l_2 = 3$, and $m = 3$

Discussion

The shape of the output functions of the hidden layer neurons is derived from the interpolation rule. If the interpolation rule is chosen to be less restrictive, other output functions may be used with similar results. However, the interpolation rule chosen here shows a number of useful properties, e.g. that the network function at the center of a cell equals the mean of the function values at the corners.

In networks with output functions independent of the goal functions the achievable accuracy of the approximation must depend on the size of the network. In DEFAnet the number of neurons and synapses required can be calculated from the resolution of the grid in the input space. It may be pointed out that cells added to the grid may require as little as one additional third layer neuron per cell in order to have the n-dimensional interpolation rule holding inside the whole cell.

It is obvious, that the weights of the plastic synapses can be found by learning, and as the last neuron (layer) is linear, problems of convergence do not occur. A wide variety of rapid learning rules may be employed, e.g the Delta rule; backpropagation of errors is not required. Under certain conditions it may be possible to solve self-optimization problems with DEFAnet by introducing forgetting mechanisms (Werntges and Daunicht 1988).

In all essential aspects DEFAnet is consistent with the properties of biological neurons. Of course, one cannot expect the input space to be subdivided into non-overlapping grid cells or the interpolation rule to be followed exactly in natural networks, but it becomes clear, what size of a network suffices to implement arbitrary continuous functions in a nervous system. In fact, the size is considerably less than that indicated in articles dealing with the existence of such networks (see e.g. Lapedes and Farber 1988). It may be noted that the units of some layers need not to correspond to neurons. E.g., in sensory-motor systems, it is conceivable that the first layer is represented by sensory organs.

A DEFAnet has been successfully tested in a simulation to approximate an exact solution to the inverse kinematics of a redundant manipulator with a 3-dimensional input space, 10^3 grid cells and a 4-dimensional output space.

Supported by Der Bundesminister für Forschung und Technologie

References

[1] Cybenko, G. Approximation by superpositions of a sigmoidal function. Math. Control, Signals and Systems, 1989, (submitted)

[2] Guez, A. and Ahmad, Z. Solution to the inverse kinematics problem in robotics by neural networks. IEEE Intern. Conf. Neur. Networks, San Diego, July 1988, II, 617-624

[3] Kolmogorov, A.N. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. Dokl. Akad. Nauk. USSR, 1957, 114: 953-956 (Russ.); AMS Transl., 1963, 28: 55-59

[4] Lapedes, A. and Farber, R. How neural nets work. Technical Report LA-UR-88-418, Theoretical division, Los Alamos National Laboratory, 1988

[5] Werntges, H.W. and Daunicht, W.J. Effects of forgetting on the self-optimization of redundant sensory-motor control networks. Abstr. 1st Ann. INNS Meeting, Boston, 1988, 366.

INTERNAL REPRESENTATION OF SPACE IN NEURAL NETWORKS OF PRIMATES AND OTHER SENSORIMOTOR MAPPING MACHINES

Rolf Eckmiller, Div. Biocybernetics, Heinrich-Heine-Universität Düsseldorf, Universitätsstraße 1, D-4000 Düsseldorf 1 (F.R. Germany)

Introduction

Primate brain research at the systems level emphasizes various sensimotor brain functions such as:
visually and/or vestibularly induced eye movements,
auditory induced vocalization ("speak what you hear"),
visually induced arm pointing ("point at where you look"), or
auditory induced gesticulation and/or locomotion.
These brain functions serve to map spatio-temporal events (trajectories) in a given sensory space (e.g. Euclidian space of visual target locations within arm's reach or space of sound frequencies and frequency patterns) onto 'corresponding' spatio-temporal events for a related motor system. These sensorimotor mapping operations sometimes immediately follow sensory stimulation in a reflectory fashion (e.g. vestibulo-ocular reflexes), whereas others are capable of storing the spatial and temporal parameters of the sensory events over an extended period of time before a decision is being made regarding the time of movement and the selected specific motor system. The various functionally separate areas within the thalamus, neocortex, and cerebellar cortex are unlikely to perform the required mapping algebraic-analytically (e.g. by solving a set of differential equations). The available evidence suggests that the underlying mathematical mapping problem is being imbedded in the geometry and time-dependent, layered neural net topology of the different communicating brain regions.

Various neurophysiological, behavioral, and theoretical studies (2,3,6,8,9,11,15,17,18,20,22,23) have been performed regarding the internal space representation in the brain by means of visually induced pointing experiments in both trained macaque monkeys and human subjects. Especially the parietal cortex (1,10), the cerebellum (16), and various regions of the precentral cortex (12,13,19) received attention.

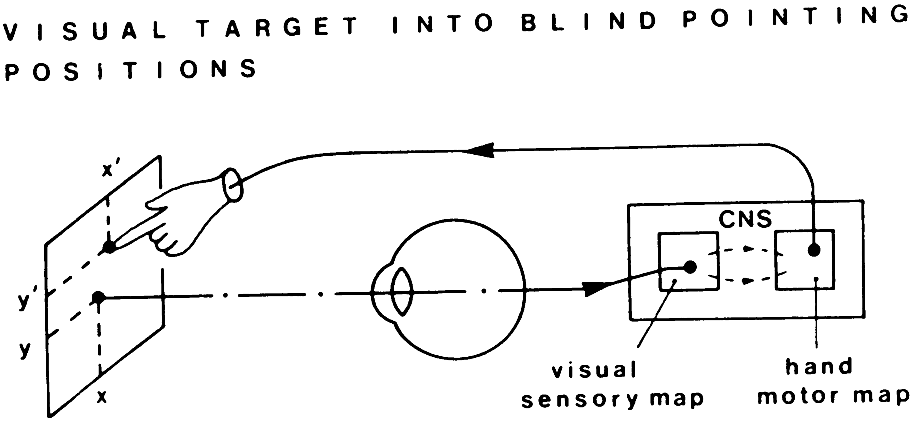


Fig.1: Schema for a typical sensorimotor mapping operation.

*) Supported in part by grants from BMFT and DFG to R.Eckmiller.

Pointing to visual targets within the grasping space without seeing the pointing arm (blind pointing) was recently studied in trained monkeys (7,10,14,25) and human subjects (4,5,21,24,26), addressing several aspects of this sensorimotor mapping operation (Fig.1). This paper describes new results on blind pointing in normal human subjects.

Results

Fig.2 indicates the relation between visual targets (open symbols) and corresponding blind pointing positions (filled symbols; average of three measurements) of the right hand. Each measurement sequence included blind pointing (repeated three times) to seven randomly presented target positions at straight ahead (open star), 10 deg (open circles), 20 deg (open triangles), and 30 deg (open squares) eccentricity along a horizontal, vertical, or oblique (+/- 45 deg) line. The data in Fig.2 are plotted as viewed by an external observer (30 deg R on the left). The left half of Fig. 2 refers to the oculomotor paradigm, which required the subject to look at (fixate) a given target (red LED) on the inner surface of a hemispherical screen (radius: 23cm; thickness: 2.5cm styrofoam). The screen with a set of LEDs (only the illuminated LED was visible) was positioned in front of the subject's eyes with the center of the hemisphere at the nasal root.

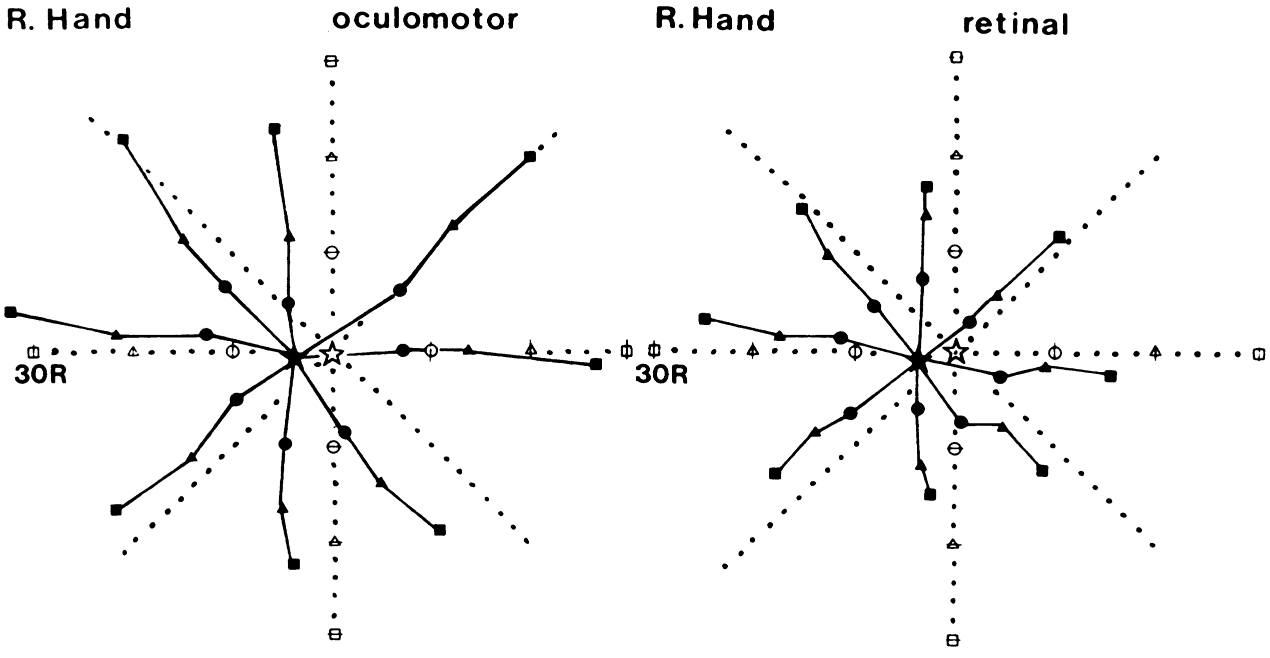


Fig.2 Blind pointing with oculomotor(left) and retinal(right) paradigm.

In the oculomotor paradigm the spatial coordinates of the target are represented as cross points of the two lines of gaze, i.e., by the oculomotor control signals (efference copy) of both eyes. The different control signals can be assumed to form a neural eye position map, which is closely related to the postulated internal space map (9,21,23). Please, note the significant (though highly reproducible and characteristically different for different subjects) differences between target and corresponding blind pointing positions. One can see differences in blind pointing gain (e.g. 30 deg downward target position yields only about 22 deg downward pointing position), translatory shifts, and even rotatory shifts in this typical sensorimotor mapping operation (driver reaching for cigarette lighter; pilot reaching for a panel button without looking).

The right half of Fig.2 gives data of the same subject during the retinal paradigm: a green LED at straight ahead location had to be fixated constantly to avoid any eye movements, while a red LED appeared at one of the visual target locations (as in the oculomotor paradigm). The subject was asked to point at this extrafoveally presented target from the outside of the screen. In the retinal paradigm, visual target coordinates are represented as positions on a retinotopic map. The spatial information of this sensory map yields an entirely different mapping result. Fig.2 (right half) clearly indicates that the blind pointing gain was significantly reduced in the retinal paradigm relative to the oculomotor paradigm. In some other of the 10 studied subjects, the retinal paradigm yielded a larger gain than the oculomotor paradigm for reasons not yet understood. It is noteworthy that all subjects typically made the final decision regarding direction an amplitude prior to a given pointing movement in that the initially reached pointing position was only very rarely corrected. In other words, subjects 'knew' already before the pointing movement that they would be satisfied with the match of target and pointing position.

Conclusions

1) The internal representation of space may be different for purposes of perception (sensory map) versus action (motor map) and is dependent on the updating input (sensory or motor).
2) Although movements are internally generated, their precision is dependent on continuous 'sensory updating', which was not available (with regard to vision) during the blind pointing task. The sensory updating of internal space representations is also necessary for other sensorimotor mapping tasks as demonstrated for example by the poor motor performance of vestibulo-ocular reflexes in the dark or vocalization without auditory feedback.
3) Autonomous vehicles and redundant robots under neural net control require a powerful module for internal space representation to assure sufficiently precise multi-modal sensorimotor mapping operations in real time in the absence of algebraic-analytical solutions.

References

(1) R.A. Anderson; G.K. Essick,R.M. Siegel; The encoding of spatial location by posterior parietal neuron, Science, Vol. 230, pp. 456-458, 1985
(2) M.A. Arbib; Schemas for the temporal organization of behaviour, Human Neurobiol., Vol. 4, pp. 63-72, 1985
(3) M.B. Berkinblit, A.G. Feldman, O.I. Fukson; Adaptability of innate motor patterns and motor control mechanisms, Behavioral and Brain Sciences, Vol. 9, pp. 585-638, 1986
(4) B. Biguer, C. Prablanc, M. Jeannerod; The contribution of coordinated eye and head movements in hand pointing accuracy, Exp. Brain. Res., Vol. 55, pp. 462-469, 1984
(5) O. Bock, R. Eckmiller; Goal-directed arm movements in absence of visual guidance: evidence for amplitude rather than position control, Exp. Brain Res., Vol. 62, pp. 451-458, 1986
(6) B. Bullock, S. Grossberg; "Neural dynamics of planned arm movements: emergent invariants and speed-accuracy properties during trajectory formation", In

Neural Networks and natural intelligence, S. Grossberg (ed.), MIT Press, Cambridge, pp. 553-622, 1988

(7) D. Domann, O.Bock, R. Eckmiller; "Interaction of visual and non-visual signals in the initiation of smooth pursuit eye movements in primates", Behav. Brain Res., Vol. 32, pp. 95-99, 1989

(8) R. Eckmiller, J. Beckmann, H. Werntges, M. Lades; Neural kinematics net for a redundant robot arm, Proc. Int. Joint Conf. Neural Networks, Vol. II, pp. 333-339, 1989

(9) R. Eckmiller; Neural nets for sensory and motor trajectories, IEEE Control Systems Magazine, Vol.9, pp. 53-60, 1989

(10) S. Faugier-Grimaud, C. Frenois, F. Peronnet; Effects of posterior parietal lesions on visually guided movements in monkeys, Exp. Brain Res., Vol. 59, pp. 125-138, 1985

(11) T. Flash, N. Hogan; The coordination of arm movements: An experimentally confirmed mathematical model, J. Neurosci., Vol. 5, pp. 1688-1703, 1985

(12) A.P. Georgopoulos; On reaching, Ann. Rev. Neurosci., Vol. 9, pp. 147-170, 1986

(13) A.G. Georgopoulos; Neural integration of movement: role of motor cortex in reaching, Faseb Journal, Vol. 2, pp. 2849-2857, 1988

(14) R. Held, J.A. Bauer; Development of sensorially-guided reaching in infant monkeys, Brain Research, Vol. 71, pp.265-271, 1974

(15) G. Hinton; Parallel computations for controlling an arm, J. Motor Behavior, Vol. 16, pp. 171-194, 1984

(16) J.C. Houk, A.R. Gibson; Sensorimotor processing through the cerebellum, In: New Concepts in Cerebellar Neurobiology, J.S. King (ed.), Alan Liss Inc., pp. 387-416, 1987

(17) M. Kawato, Y. Uno, M. Isobe, R. Suzuki; Hierarchical neural network model for voluntary movement with application to robotics, IEEE Control Systems Magazine, Vol. 8, pp. 547-561, 1988

(18) M. Kuperstein; An adaptive neural model for mapping invariant target position, Behavioral Neuroscience, Vol. 102, pp. 148-162, 1988

(19) R. Lemon; The output map of the primate motor cortex, TINS, Vol. 11, pp. 501-506, 1988

(20) L. Massone, E. Bizzi; A neural network model for limb trajectory formation, Biol. Cybern., Vol.xx, pp. 1-9, 1989

(21) D. Ott, R. Eckmiller; "Dynamic adaption of the blind pointing characteristic to stepwise lateral tilts of body, head,-and trunk", Behav. Brain R., Vol. 30, pp. 99-110, 1988

(22) A. Pellionisz; Tensor geometry: A language of brains & neurocomputers. Generalized coordinates in neuroscience & robotics, In: Neural Computers, R. Eckmiller, C.v.d.Malsburg (eds.), Springer, Heidelberg, pp. 381-391, 1988

(23) E. Saltzman; Levels of sensorimotor representation, J. Math. Psychol., Vol. 20, pp. 91-163, 1979

(24) J. F. Soechting, F. Lacquaniti, C. A. Terzuolo; Coordination of arm movements in three-dimensional space. Sensorimotor mapping during drawing movement, Neuroscience, Vol. 17, pp. 295-311, 1986

(25) E. Taub, I.A. Goldberg, P. Taub; Deafferentation in monkeys: Pointing at a target without visual feedback, Exp. Neurology, Vol. 46, pp. 178-186, 1975

(26) B.T. Volpe, J.E. LeDoux, M.S. Gazzaniga; Spatially oriented movements in the absence of proprioception, Neurology, Vol. 29, pp. 1309-1313, 1979

CODING OF THE DIRECTION OF REACHING BY NEURONAL POPULATIONS

Apostolos P. Georgopoulos

The Philip Bard Laboratories of Neurophysiology, Department of Neuroscience, The Johns Hopkins University School of Medicine, Baltimore, Maryland, USA.

The generation and control of reaching is a function of several motor structures. Some principles of coding of the direction of reaching by motor cortical neurons and neuronal populations have now become evident and will be discussed in this presentation. First, large populations of neurons in the motor cortex are engaged with reaching. Second, this engagement is early, starting approximately 60 ms following the onset of a visual target. Third, the time course of the cell recruitment in the active population is very similar for equal-amplitude reaching movements directed to different targets. Fourth, the intensity of cell discharge is modulated with the direction of reaching. A cell discharges at highest intensities with reaching in a particular direction (the cell's "preferred direction") and at progressively lower intensities with reaching movements in directions that are farther and farther away from the preferred one. Typically, the discharge rate is a cosine function of the angle formed by the direction of a particular reaching movement and the cell's preferred direction. Fifth, an unambiguous, distributed code for the direction of reaching exists in neuronal populations in the motor cortex. The motor command for the direction of reaching is regarded as composed of cell vectors each of which points in the cell's preferred direction and has length proportional to the change in cell activity associated with that particular reaching movement. The outcome of this population code can be visualized as a vector (the "neuronal population vector") that points in the direction of the upcoming reaching. This population vector is an accurate and robust predictor of the direction of reaching in space; is resistant to cell loss; it can be estimated reliably from about 100 cells; and it predicts well the direction of reaching well before the reaching begins (i.e. during the reaction time), and even during an instructed delay period. Finally, when a mental transformation is required for the generation of a reaching movement in a different direction from a reference direction, the population vector can provide useful information concerning the nature of the cognitive process by which the required transformation is achieved. (Supported by USPHS grant NS17413 and ONR contract N00014-88-K-0751.)

REFERENCES

Georgopoulos AP, Schwartz AB, Kettner RE (1986) Neuronal population coding of movement direction. Science 233: 1416-1419.

Georgopoulos AP, Kettner RE, Schwartz AB (1988) Primate motor cortex and free arm movements to visual targets in three-dimensional space. II. Coding of the direction of movement by a neuronal population. J. Neurosci. 8: 2928-2937.

Georgopoulos AP, Lurito JT, Petrides M, Schwartz AB, Massey JT (1989) Mental rotation of the neuronal population vector. Science 243:234-236.

RELATIONSHIP OF VISUAL SPATIAL MAP AND SACCADIC MOTOR MAP IN SALAMANDER

Gerhard Manteuffel
Brain Research Institute (FB-2), University Bremen
D-2800 Bremen 33, Fed. Rep. Germany

The eyes of salamanders are fixed in the head such that they are positioned obliquely to each other and to an orthogonal coordinate system given by the horizontal plane of the head and the respective vertical. The retinae of both eyes therefore represent two independent coordinate systems. Thus, an object located within the binocular field that is determined horizontally by the convergence angle of the eyes, is represented differently in each coordinate system.

In the optic tectum, the ipsilateral and contralateral topographies are arranged so that excitations from the ipsilateral and the contralateral eye correspond (i.e. are at the same site) if an object within the binocular field is at a certain distance that is given by the topographic relation and can be considered the system's horopter. This relationship is established by an intertectal transfer mechanism (probably via the nucleus isthmi) whereby excitation at a particular site in one tectal hemisphere is carried to a particular site in the other hemisphere. Thus, the z-coordinate of the environment (i.e. the distance of objects) is established by a combination of the two-dimensional retinotopic maps in both tectal hemispheres.

It is assumed that the bilateral visual layers of the tectum, coupled according to the properties of intertectal transfer, trigger sites in the respective motor layers lying immediately beneath. Therefore, a more posterior excitation in one tectal hemisphere will result in a more anterior excitation of the other hemisphere. However, direct excitation from the contralateral eye is known to be stronger than the indirect input from the ipsilateral eye. As a result, under binocular conditions the contralateral excitation may suppress the ipsilateral one by means of lateral inhibition, providing both are not at the same site, so that each visual hemisphere will have only one output. Under monocular conditions, however, the ipsilateral input survives such that both tectal hemispheres then have an output depending on intertectal transfer (meaning that any object is then seen located on the horopter).

The output of the tectum has been analysed by neuroanatomical tracing (Naujoks-Manteuffel and Manteuffel, 1988). When horseradish peroxidase was applied at bulbar levels, a specific distribution of tectal output cells was found. Cells are generally more numerous on the ipsilateral than on the contralateral side, but on both sides cell density increases from the rostral pole of the tectum up to an isthmic level and then decreases rapidly. The distribution is also non-homogeneous in a medio-lateral direction. On the ipsilateral side (relative to the injection) efferent cells are clustered medially and laterally and are sparse intermediately. On the contralateral side, however, cells are sparse medially and increase in number toward the lateral margin.

In salamanders, gaze direction is adjusted by the actions of the bilateral epaxial and hypaxial neck muscles driven by their respective motoneuron pools. These muscles allow essentially two-dimensional head movements within a coordinate system relative to the trunk.

Bearing in mind the visual topography and assuming that information transfer from the visual map to the motor map occurs predominantly within vertical columns, it is tempting to speculate that the lateral and medial tectal output groups are premotoric to the hypaxial and epaxial muscles, respectively. Since dendrites of tectal output cells arborize considerably over some 100 micrometers, they can receive input from more than one visual column. As a result, the intermediate visual zone of the tectum, that is responsive to stimuli close to the visual horizon, will have an efference to both the pre-hypaxial and the pre-epaxial output neurons.

The crucial statements of this hypothesis are that the premotor output map of the tectum is arranged in the coordinates of the neck muscles and that the strength of muscle contraction is determined by the number of activated (recruited) premotor cells. It can be assumed that both the ipsilateral and the contralateral descending pathways from each tectal hemisphere will finally reach the contralateral motoneurons. As the contralaterally descending tecto-bulbar tract is located more medially than the ipsilateral one, it will provide a major input to the medial part of the nucleus reticularis medius (nRM), whereas the ipsilateral tract can be assumed to synapse mainly on the lateral portion of this nucleus. The lateral portion of the nRM, however, projects to the contralateral spinal cord, whilst the medial portion projects ipsilaterally.

The bilateral visual layers of the tectum may act locally (i.e. at the site of activation) and independently on the respective motor maps. This action must be considered as a triggering one so that any activity at a certain site on the visual map recruits the respective premotor neurons, whereby the strength of activity in the visual layer determines whether or not a threshold intercalated between both maps can be surpassed. The rationale for this assumption is given by the fact that although tectal neurons are excited more vigorously by some stimuli than by others, this excitation determines only the probability of a saccadic movement and not its precision (i.e. the turning angle). It is likely that the threshold is variable and controlled by "motivational afferents" arising from the amygdala and other forebrain structures that project to the tectum.
According to this hypothesis, both tectal sides convey independent outputs that may, however, differ in strength: a more posterior excitation at the visual map of one tectal hemisphere will result in a stronger premotor activity than in the other premotor map triggered by a visual excitation located more rostrally (if the stimulus is not positioned straight ahead). The two output channels (consisting of two sub-channels: the epaxial and the

hyaxial one) are then considered to inhibit each other at some site, so that activation finally occurs only in the more active one.

A computer simulation of the presented hypothesis on the design of the saccadic system reveals the typical head movements and lines of approach to a prey found in binocular or monocular salamanders. Most impressive, the strange approach of monocular salamanders toward prey is precisely reproduced. Such animals do not walk straight ahead to an object presented at a short distance exactly in front of them, but first deviate toward the side of the seeing eye. Somewhat later, when approaching closer, they curve sideways and eventually reach the object (Roth, 1987). This can be explained by the type of intertectal transfer between the visual maps of both sides. At the starting point, the tectum contralateral to the eye receives a relatively caudal excitation and the transferred signal to the other tectal side occurs rostrally. Hence, the output of the contralateral tectum is stronger and evokes turning away from the object (toward the seeing eye). During approach, the object will stimulate more and more temporal positions of the retina and this results in an increasingly rostral position of excitation on the contralateral tectum. As a result of intertectal transfer, the other tectal side (ipsilateral to the eye) will then be stimulated more and more caudally and will therefore have an increasing output. From the point where the ipsilateral tectal output is stronger than the contralateral one, the animal turns to the side of the blinded eye and, thus, toward the object.

REFERENCES:

Naujoks-Manteuffel,C., and Manteuffel,G. (1988): The origins of descending projections to the medulla oblongata and rostral medulla spinalis in the urodele Salmandra salamandra (Amphibia). J. comp. Neurol.: 273, 187-206.

Roth,G. (1987): Visual behavior in salamanders; Studies of brain function, vol. 14, Springer, Berlin.

On the Role of Input Representations in Sensorimotor Mapping

Lina Massone and Emilio Bizzi

Dept. of Brain and Cognitive Sciences
Massachusetts Institute of Technology

Abstract

This paper emphasizes the role of input representations in sensorimotor mapping. We present experiments in training a sequential network that generates aiming movements with three different stimuli representations and we describe the corresponding behavior as far as generalization and learning are concerned.

1 Introduction

In this paper we aim at emphasizing the role of input representations in sensorimotor mapping. We will do it by describing some experiments performed during training of a sequential network that generates aiming movements of a redundant limb towards targets specified as sensory stimuli. The network is fully described in [Massone and Bizzi 1989, Massone and Bizzi in press]; its main features will be briefly summarized in Section 2. Section 3 will address the particular problem of input representations and will show that the network can exhibit quite different behaviors depending on the adopted representation of the input stimuli.

2 The Network

We represented the aiming task by means of a sequential network of the type proposed by Jordan [1986]. The network is composed of two arrays of input units (plan units and state units), one array of hidden units and one array of output units. Due to recurrent connections from output units to state units, this network is able to produce sequences of output signals. The state units provide then a time-varying input to the layered network that learns the sequences. The other input to the network derives from the plan units, which are activated by the external stimuli. The activation of the plan units remains constant within a given sequence but varies between sequences to allow different sequences to be learned by the same network.

In our case, the output units drove a redundant three-joint limb which moved from a fixed initial posture to a target. The limb was schematically modeled with four pairs of antagonist muscles: the shoulder flexor and extensor, the double joint flexor and extensor, the elbow flexor and extensor,

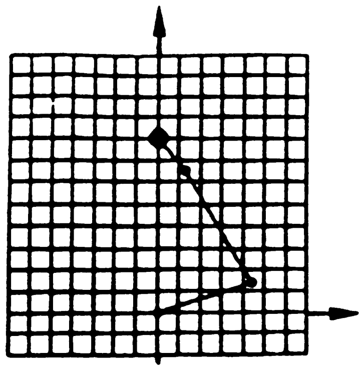


Figure 1: *A portion of the limb workspace discretized with a 15x15 grid. The limb is in the starting posture.*

the wrist flexor and extensor. Muscles were represented as springs according to a model that is described later in this section. Each output unit activated a muscle; output units were considered as motorneurons. Hence, the time sequence generated by the network was encoded in muscle space.

Plan units contained a representation of the sensory stimulus. A portion of the limb workspace was discretized with a 15x15 pixel grid as shown in Figure 1; Figure 1 also shows the initial posture of the limb for all aiming movements. The stimulus was encoded as a narrow gaussian distribution centered on one of the 225 pixels; any pixel could become the target of the aiming movement. All units in the network had continuous activation functions.

The network's task involved generating a trajectory of the limb from the starting posture toward one of the targets. Hence, the network performed a sensory-motor transformation.

Aiming movements were assumed to be planar. We used a bell-shaped velocity profile for training trajectories, which is a recurring feature of movements performed by biological systems. The duration of movements was assumed to be constant. Consequently, the network was asked to generalize not only the path of the limb towards the target but also the corresponding velocity profile.

The network was trained through supervised learning. To compute the sequence of muscle activations that correspond to a trajectory of the limb from the initial to the final posture we used a model which represents muscles as tunable springs characterized by a set of integrable functions between length and tension at steady state [Mussa Ivaldi et al. 1988]. This model makes it possible to compute:

- the xy coordinates of the end-point position of the limb, given the activation of the muscles. (This computation is a well-posed problem.)
- the muscle activations given the xy coordinates of the end-point. (This is an ill-posed problem, solved by applying a minimum potential-energy constraint.)

We employed the inverse transformation (from end-point position to muscle activation) to compute the training set. The direct transformation (from muscle activation to end-point position) was used during the testing phase.

Experiments performed during learning and on the trained network showed that: (i) the task could be learned by a three-layer sequential network; (ii) the network successfully generalized in trajectory space and adjusted the velocity profiles properly; (iii) the same task could not be learned by a linear network; (iv) after learning, the internal connections became organized into inhibitory and excitatory zones and encoded the main features of the training set; (v) the model was robust to noise on the input signals; (vi) the network exhibited attractor-dynamics properties; (vii) the network was able to solve the motor-equivalence problem.

3 Input Representations

The network described in Section 2 was trained with three different representations for the input stimuli, namely a local representation and two distributed ones. Only one out of the three gave good results from the point of view of the generalization properties of the network. It is worth noting that in our case building a stimulus representation means translating the values of the 225 pixels into activation values of the plan units. In the remainder of this section we will describe the three representations as well as the corresponding network's behavior and we will try to provide explanations for these results.

3.1 Local Coding

Local coding of the stimulus means associating one plan unit to each pixel in the workspace. In this case the network had 225 plan units. The activation of each plan unit was simply the value of the corresponding pixel (all values were properly normalized between 0 and 1, 0 being the value for absence of stimulus, 1 being the value at the gaussian peek, i.e. at the target.)

With this local coding the network could learn the task (the training set contained about 15 trajectories) but its performance from the point of view of generalization was bad. The limb moved in the right direction but it missed the targets of a significant amount and the velocity profile was not bell-shaped. The network basically behaved as a look-up table.

3.2 XY Coding

XY coding refers to a distributed representation introduced in [Hinton et al. 1986]. It requires one plan unit for each row and one plan unit for each column of the workspace, 30 units in our case. Each pixel is represented by a pair of units (one row unit and one column unit) encoding its position in the workspace. This representation is known to have a few drawbacks, the main one being the ambiguity arising when more than one pixel has to be represented. In fact this representation fails to encode explicitly "what goes with what". In our case, this particular problem does not apply as

- only one stimulus at the time is represented;
- although each stimulus activates more than one pixel (because of the gaussian distribution) only one of such pixels (the target) has a value equal to 1.

Consequently, each pattern of activation across the plan units uniquely identifies one target.

With the XY coding the network could learn the task, but, surprisingly, its generalization capabilities were even worst than in the case of local coding. Besides missing the target and not being able to adjust the velocity profile, the limb often moved in the wrong direction. A possible explanation for this misbehavior came from an analysis of the muscle activations in the training set. We found that for different portions of the workspace the sequences of muscle activation were totally different in structure. Let's consider, for example, two targets on the same row, the first one in the top-right portion of the workspace and the second one in the top-left portion; the sequences of muscle activation that lead the limb to those targets did not share any common structure. This fact implies that the notion that two targets are on the same row not only does not help the network to understand the task, but can be highly misleading. The network builds an internal representation for the task that is "wrong".

3.3 Coarse Coding

We achieved good generalization by coarse coding the stimuli as follows. Each plan unit had a receptive field over the array of pixels; each receptive field contained nine pixels and was partially overlapped with neighbor fields. The result was a 7x7 array of plan units. The activation of each plan unit was the sum of the values of the pixels that belonged to its receptive field. The resulting values were then properly normalized. Besides achieving good generalization on both path and velocity profile, learning was faster (it took about 2/3 of the trials needed in the case of XY coding.) More detailed information about learning and generalization with coarse coding can be found in [Massone and Bizzi 1989, Massone and Bizzi in press]. This result shows that what is important for the limb to reach a particular target is the behavior in a neighborhood of that target, rather then the behavior in portions of the workspace that are far apart. An other nice feature of coarse coding is that overlapping receptive fields can partially overcome the arbitrariness of the workspace discretization by translating a spatial discretization into continuous activations of plan units. Furthermore, this coarse coding can be considered a good approximation of the receptive fields of sensory neurons in the case, for example, of skin stimulation.

4 Conclusions

In this paper we showed the importance of building good representations of the input stimuli for sensorimotor transformations. We presented experiments in training a sequential network that generates aiming movements with three different stimuli representations and we described the corresponding behavior as far as generalization and learning are concerned. Coarse coding [Hinton et al. 1986] turned out to be the best representation for this particular motor task.

References

Hinton GE, McClelland JL, Rumelhart DE (1986) Distributed Representations, in Parallel Distributed Processing, McClelland JL, Rumelhart DE Eds., MIT Press, Cambridge (MA).

Jordan MI (1986) Attractor Dynamics and Parallelism in a Connectionist Sequential Machine, Proc. 8th Annual Conf. of the Cognitive Science Society, Hillsdale, N.J.: Erlbaum.

Massone L, Bizzi E (1989) Generation of Limb Trajectories with a Sequential Network, Proc. Int. Joint Conf. on Neural Networks, June 18-22, Washington D.C.

Massone L, Bizzi E (in press) A Neural Network Model for Limb Trajectory Formation, Biol. Cybern.

Mussa Ivaldi FA, McIntyre J, Bizzi E (1988) Theoretical and Experimental Perspectives on Arm Trajectory Formation: A Distributed Model for Motor Redundancy, in Biological and Artificial Intelligence Systems, E. Clementi and S. Chin Eds.,pp. 563-577, Escom.

ACKNOWLEDGEMENTS

The authors wish to thank Michael Jordan for his constant support and valuable suggestions, and for making available the basic software which implements sequential networks. Thanks also go to Joe McIntyre and Ferdinando Mussa Ivaldi for providing the model of muscles. This work has been supported by the Office of Naval Research Grant N00014/88/k/0372 and the Sloan Foundation.

Learning Spatiotemporal Patterns in A Neural Network with Lateral Inhibitory Connections

Noboru Murata, Kenji Doya and Shuji Yoshizawa

Department of Mathematical Engineering and Information Physics
Faculty of Engineering, University of Tokyo

address : 7-3-1, Hongo, Bunkyo-ku, Tokyo 113, Japan
e-mail address : mura%sat.t.u-tokyo.ac.jp@relay.cs.net

1. Introduction

Animals have superior ability of conducting series of well organized motions. The ability is considered to be supported by some flexible neural mechanisms which memorize and control spatiotemporal patterns. In several lower animals simple types of such neural networks have been observed. These networks are called 'motion pattern generators'. Some mathematical models of the networks were proposed and it was shown that they can memorize some simple temporal patterns (see Suzuki et. al., 1972; Doya and Yoshizawa, 1989, for example). However, these networks do not explain complex motions of higher animals, because their ability is restricted to generating relatively simple patterns and not enough to regenerate the memorized pattern in another temporal structure. In this paper, a neural network model is proposed, which memorizes spatial and temporal patterns in separate parts of the network so that the scheduling of the spatiotemporal pattern regeneration is possible. The capability of the model is confirmed by computer simulations.

2. Neural Network Model

2.1 Structure of the Network

The network has three layers: input, hidden and output layers. Stimuli to the network are given on the input layer. Autonomous excitations are supposed to exist on the hidden layer. The dynamics of the excitation is changed according to the value of the weighted sum of the input stimuli through input-to-hidden connections. Hidden-to-output connections transform the excitation patterns on the hidden layer into desired patterns on the output layer. This two layered structure gives an easy way of memorization and regeneration of spatiotemporal patterns (Fig.1a).

The following properties are required for the hidden layer. The excitation pattern on the hidden layer changes continuously, and the excitations at different time are distinguished from each other. The transition from one excitation to the other takes some duration which is controlable by external inputs. As a realization, a nerve field with lateral inhibitory connections is adopted and travelling local excitation on the nerve field is used.

The nerve field consists of two layers of excitatory and inhibitory neurons. Each excitatory or inhibitory neuron is connected to excitatory and inhibitory neurons in its neighborhood. In addition, each neuron of the field receives input stimuli from outside. Figure 1b shows these connections. Based on the computer simulation of this kind of nerve fields, Wilson and Cowan (1973) reported that following three types of local excitation patterns occur depending on feature of external stimuli and parameters of the field: transient local excitation around an intensely stimulated locus, lasting local excitation around an intensely stimulated locus and travelling local excitation across the field. Amari (1977) analyzed these phenomena mathematically. Only the third phenomenon is considered in the followings.

2.2 Fundamental Equations of the Network

In the followings, vectors $s_A(t)$, $u_A(t)$ and $y_A(t)$ denote external stimulus, inner state and output vectors, respectively. The j-th component of these vectors represents the value of the j-th A-layer neuron at time t. Matrix $W_{A\,B}(t)$ denotes the connecting weight matrix, whose (j,k) component represents weight of connection from the k-th B-layer neuron to the j-th A-layer neuron at time t. Subscripts i, ex, in and o represent input layer, hidden excitatory layer, hidden inhibitory layer and output layer, respectively.

The input layer consists of n_1 neurons. Output function of the input neuron is assumed to be the identity function. Thus,

$$y_i(t) = u_i(t) = s_i(t) \quad \text{(input to the network).} \tag{1}$$

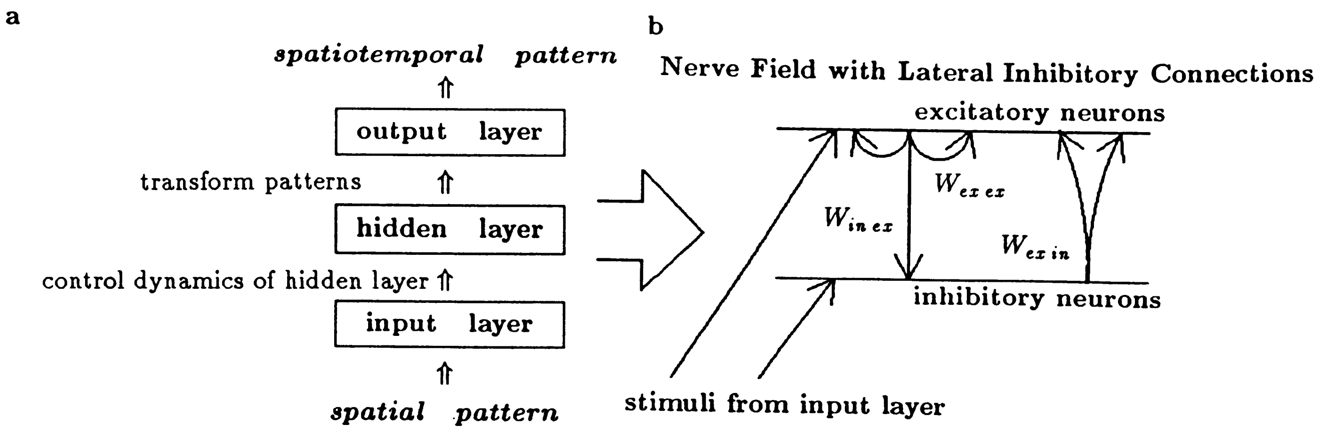


Fig. 1. The structure of the neural network model.
a : The layered structure. **b** : Connections of hidden layer.

Hence, output vector $y_i(t)$ represents state of the input layer.

The hidden layer consists of n_2 excitatory and n_2 inhibitory neurons. Values of external stimulus vectors are caluculated by equations

$$s_{ex}(t) = W_{ex\,i}(t)y_i(t), \quad s_{in}(t) = W_{in\,i}(t)y_i(t), \tag{2}$$

where $W_{ex\,i}(t)$ and $W_{in\,i}(t)$ are $n_2 \times n_1$ weight matrices of input-to-hidden connections. The components of weight matrices $W_{ex\,ex}$, $W_{ex\,in}$, $W_{in\,ex}$ and $W_{in\,in}$ are time-independent and defined as follows:

$$\begin{aligned} (W_{ex\,ex})_{jk} &= \begin{cases} w_{ex\,ex} & \text{if } |j-k| < \sigma_{ex\,ex} , \\ 0 & \text{otherwise,} \end{cases} & (W_{in\,ex})_{jk} &= w_{in\,ex}\delta_{jk} = \begin{cases} w_{in\,ex} & \text{if } j = k , \\ 0 & \text{otherwise,} \end{cases} \\ (W_{ex\,in})_{jk} &= \begin{cases} w_{ex\,in} & \text{if } |j-k| < \sigma_{ex\,in} , \\ 0 & \text{otherwise,} \end{cases} & (W_{in\,in})_{jk} &= 0, \end{aligned} \tag{3}$$

where $\sigma_{ex\,ex}$ and $\sigma_{ex\,in}$ are constants which decide regions of interactions. Dynamics of the hidden layer are

$$\begin{aligned} \tau_{ex}\frac{du_{ex}(t)}{dt} &= -u_{ex}(t) + W_{ex\,ex}y_{ex}(t) - W_{ex\,in}y_{in}(t) + s_{ex}(t) - h_{ex}\mathbf{1}, \\ \tau_{in}\frac{du_{in}(t)}{dt} &= -u_{in}(t) + W_{in\,ex}y_{ex}(t) + s_{in}(t) - h_{in}\mathbf{1}, \end{aligned} \tag{4}$$

$$y_{ex}(t) = \mathbf{1}[u_{ex}(t)], \quad y_{in}(t) = \mathbf{1}[u_{in}(t)], \tag{5}$$

where h_{ex} and h_{in} are thresholds of the hidden-layer neurons, vector $\mathbf{1}$ is an n_2 vector, whose components are all 1. Function $\mathbf{1}[\cdot]$ is output function of the hidden-layer neurons, and

$$i-\text{th component of } \mathbf{1}[x] = \begin{cases} 1 & \text{if } x_i > 0 , \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

The output layer consists of n_3 neurons. Inputs to the neurons are weighted sum of outputs of the hidden-layer excitatory neurons.

$$s_o(t) = W_{o\,ex}(t)y_{ex}(t), \tag{7}$$

where $W_{o\,ex}(t)$ is a $n_3 \times n_2$ connecting weight matrix. Output function of the output-layer neuron is the identity function, then input and output of the output-layer neuron are identical. Namely,

$$y_o(t) = u_o(t) = s_o(t). \tag{8}$$

2.3 Equations of Learning

Weight matrix $W_{o\,ex}(t)$ transforms spatial patterns on the hidden layer at time t into spatial patterns on the output layer. The learning rule for $W_{o\,ex}$ is defined by

$$r_{o\,ex}(t) = d(t) - y_o(t), \quad \tau_{o\,ex}\frac{dW_{o\,ex}(t)}{dt} = r_{o\,ex}(t)[y_{ex}(t)]^T, \tag{9}$$

where $d(t)$ denotes the desired output, and τ_{oex} is a small positive constant. This rule is the so-called continuous-output perceptron learning.

Weight matrix $W_{ex\,i}(t)$ decides the stimuli to the hidden-layer excitatory neurons, and controls the velocity of the travelling local excitation. Since propagation velocity of the excitation in the hidden layer increases with intensity of the stimuli to the neurons, $W_{ex\,i}(t)$ is modified according to phase relation between desired output and actual output. Modification rule for by $W_{ex\,i}(t)$ is given by

$$r_{ex\,i}(t) = g_\alpha(\nu(t))y_{ex}(t), \quad \tau_{ex\,i}\frac{dW_{ex\,i}(t)}{dt} = r_{ex\,i}(t)[y_i(t)]^T, \tag{10}$$

$$g_\alpha(x) = \begin{cases} -1 & x < -\alpha, \\ 0 & -\alpha \le x \le \alpha, \\ 1 & \alpha < x, \end{cases} \tag{11}$$

$$\tau'\frac{d\nu(t)}{dt} = -\nu(t) + \{d(t-\delta t)\cdot y_o(t) - d(t)\cdot y_o(t-\delta t)\}, \tag{12}$$

where $\tau_{ex\,i}$, τ', δt and α are small positive constants. Here, $\nu(t)$ detects phase relation and $\nu(t)$ is positive if phase delay.

Weight matrix $W_{in\,i}(t)$ decides stimuli to the hidden-layer inhibitory neurons. If the stimuli to the inhibitory neurons increase, they become more excitable and their inhibitory effect is strengthened and, as a result, local excitation area of the excitatory neurons becomes narrower. According to this property, the following modification rule for $W_{in\,i}$ is used:

$$r_{in\,i}(t) = g_n(\, N - \sum_{j=1}^{n_2} y_{ex\,j}(t)\,)y_{in}, \quad \tau_{in\,i}\frac{dW_{in\,i}(t)}{dt} = r_{in\,i}(t)[y_i(t)]^T, \tag{13}$$

where $N \pm n$ is the desired width of the local excitation area, and $\tau_{in\,i}$ is a small positive constant.

3. Computer Simulation

The capability of the model is confirmed by computer simulations. The following parameters were selected: $n_2 = 200$, $n_3 = 1$; $\tau_{ex} = \tau_{in} = 1.0$, $h_{ex} = h_{in} = 1.0$, $w_{ex\,ex} = 0.30$, $w_{ex\,in} = 0.66$, $w_{in\,ex} = 1.0$, $\sigma_{ex\,ex} = 5$, $\sigma_{ex\,in} = 2$; $\tau_{o\,ex} = 100.0$, $\tau_{ex\,i} = 50.0$, $\tau_{in\,i} = 25.0$, $\tau' = 0.5$, $\delta t = 0.5$, $\alpha = 0.04$, $\varepsilon = 0.3$, $N = 14$, $n = 1$.

3.1 Exercise I.

Figure 2a shows the output after learning. Weights $W_{o\,ex}(t)$ and $W_{in\,i}(t)$ are modified by equations (9) and (13), but weight $W_{ex\,i}(t)$ not changed. Here, $n_1 = 1$, $(W_{ex\,i}(0))_{j1} = 0.6$, $(W_{in\,i}(0))_{j1} = 0.3$ for all j. The dotted and the solid curves are the actual and the desired outputs, respectively. Figure 2b shows, on the other hand, the output after modifying $W_{ex\,i}(t)$ and $W_{in\,i}(t)$ by equations (10) and (13), and keeping $W_{o\,ex}(t)$ constant.

3.2 Exercise II.

This exercise shows the change of output according to input to the network. First, for input $y_i = (0.5, 0.5)^T$, $W_{o\,ex}(t)$ and $W_{in\,i}(t)$ are modified so as to memorize the output pattern shown in Fig.3a. Next, $W_{ex\,i}(t)$ and $W_{in\,i}(t)$ are modified so as to generate the desired output patterns shown in Fig.3b and 3c for inputs $y_i = (1,0)^T$ and $y_i = (0,1)^T$, respectively. In this case, $n_1 = 2$, $(W_{ex\,i}(0))_{jk} = 0.6$, $(W_{in\,i}(0))_{jk} = 0.3$ for all (j,k). Figure 4 shows outputs from the network for unexperienced inputs. It is seen that the network generates various temporal patterns according to the input.

4. Conclusion

A neural network model was proposed, which memorize spatiotemporal patterns and regenerate them with various temporal structures using dynamic excitation patterns on a nerve field with lateral inhibitory connections. The present network is a prototype model to explain the flexibility of acquiring and planning complex motions of higher animals.

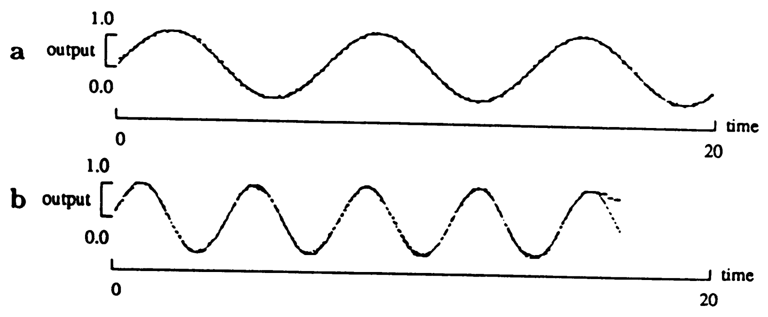


Fig. 2. Exercise I. Memorizing patterns.
The solid curve is the desired output $d(t)$, the dotted curve is the actual output $y_o(t)$.
a : Learning $W_{o\,ex}(t)$ and $W_{in\,i}(t)$. b : Learning $W_{ex\,i}(t)$ and $W_{in\,i}(t)$.

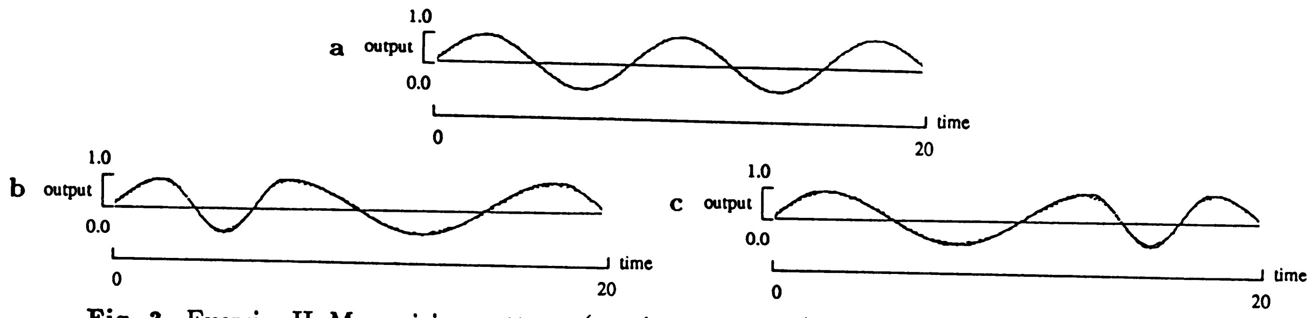


Fig. 3. Exercise II. Memorizing patterns (two input neurons).
a : Learning $W_{o\,ex}(t)$ and $W_{in\,i}(t)$ for input $y_i = (0.5, 0.5)^T$. b : Learning $W_{ex\,i}(t)$ and $W_{in\,i}(t)$ for input $y_i = (1, 0)^T$. c : Learning $W_{ex\,i}(t)$ and $W_{in\,i}(t)$ for input $y_i = (0, 1)^T$.

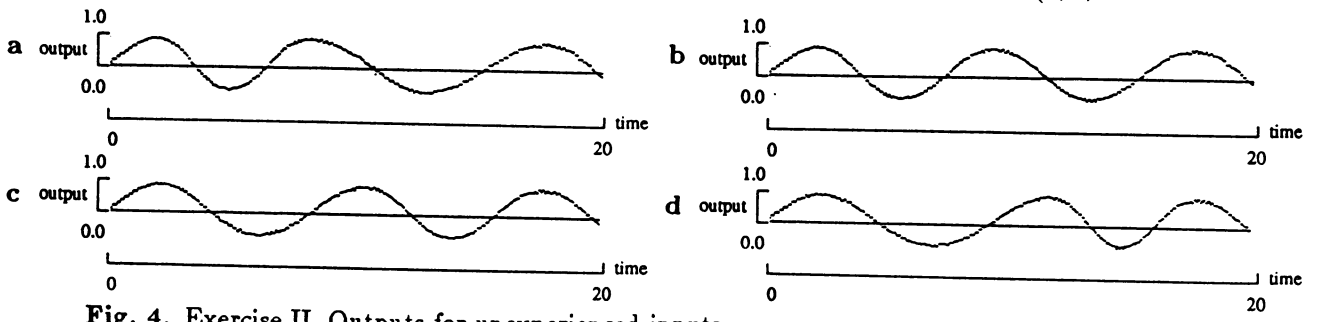


Fig. 4. Exercise II. Outputs for unexperienced inputs.
a : Output for input $y_i = (0.8, 0.2)^T$. b : Output for input $y_i = (0.6, 0.4)^T$. c : Output for input $y_i = (0.4, 0.6)^T$. d : Output for input $y_i = (0.2, 0.8)^T$.

References

Albus,J.S. (1981): Brains, Behavior, and Robotics, McGraw-Hill.

Amari,S. (1977): Dynamics of Pattern Formation in Lateral-Inhibition Type Neural Fields, *Biol. Cybern.*, **27**, 77–87.

Doya,K. and Yoshizawa,S. (1989): Memorizing Oscillatory Patterns in the Analog Neuron Network, *Proc. of International Joint Conference on Neural Networks*, **I**, 27–32.

Kishimoto,K. and Amari,S. (1979): Existence and Stability of Local Excitations in Homogeneous Neural Fields, *J. Math. Biol.*, **7**, 303–318.

Suzuki,R., Katsuno,I. and Matano,K. (1971): Dynamics of "Neuron Ring ". Computer Simulation of Central Nervous System of Starfish, *Kybernetik*. **8**, 39–45.

Tokura,T. and Morishita,I. (1977): Analysis and Simulation of Double-Layer Neural Networks with Mutually Inhibiting Interconnections, *Biol. Cybern.*, **25**, 83–92.

Wilson,H.R. and Cowan,J.D. (1973): A Mathematical Theory of the Functional Dynamics of Cortical and Thalamic Nervous Tissue, *Kybernetik*, **13**, 55–80.

Collective Oscillations in Neuronal Networks: Functional Architecture Drives the Dynamics

Daniel M. Kammen[†], Philip J. Holmes[*], and Christof Koch[†]

[†] Computation and Neural Systems Program, California Institute of Technology, 216–76, Pasadena, California 91125.

[*] Fairchild Visiting Professor, California Institute of Technology. Permanent address: Dept. of Theoretical and Applied Mechanics, Cornell University, Ithaca, New York 14853.

Abstract:

Stimulus specific frequency and phase locked oscillations in cells tuned to similar orientations but widely separated on the cortical surface (up to 7mm) have been reported in visual cortical areas (Gray *et al.* 1989). We have analytically and in simulation explored the basis for this activity by examining two neuronal architectures, representing two extremes along a continuum that can generate oscillations with the observed properties in a robust and rapid manner. Using arguments from the theory of dynamical systems we show that under certain conditions neural networks with global, feedback connections can generate oscillations among the stimulated neurons with zero phase-lag. The phase-lag among neurons coupled to their neighbours in a chain-like geometry, on the other hand, will vary with the distance between them and the phase of substantially separated groups will drift relative to each another. It therefore appears that feedback connections are primarily responsible for these oscillations.

Recent electrophysiological evidence supports the view that the firing patterns of groups of neurons in the mammalian cortex exhibit stimulus-induced oscillations. Such $40-60\ Hz$ oscillations have long been reported in the rat and rabbit olfactory bulb and cortex on the basis of both single- and multi-unit as well as EEG activity (Freeman 1978). Similar oscillations have been described in visual, auditory and motor cortex (gray *et al.* 1989, Ecknorn *et al.* 1989). In the cat visual cortex, the firing rates of cells many *mm*'s apart can be highly synchronized—with no noticeable phase-shift—as long as they have similar orientation tuning. These oscillations are not stimulus-locked. Since cells in mammalian cortex are not thought to act as Central Pattern Generators (CPG) the question as to the origin and coordination of these oscillations arises.

We are interested in exploring the anatomical basis of this process at a functional level in the visual cortex. In earlier work on CPG's (Cohen, *et al.* 1982), it was argued that, in studying coupling effects among large populations of bursting neurons one can ignore the details of individual oscillators and represent each one simply by a single variable: its *phase*. Letting $\theta_j(t)$ be the phase of the j^{th} oscillator in a collection of $N+1$ oscillators leads to models of the form:

$$\frac{\partial \theta_j}{\partial t} = \omega_j + f_j(\theta_0, \theta_1, \ldots, \theta_N) \tag{1}$$

where ω_j is the firing frequency of the j^{th} oscillator and the coupling functions, f_j, are continuous and periodic in each phase variable. In eq. (1) ω_j represents the properties of individual oscillators, and in particular its level of excitation, while the functions f_j encode the types and architecture of the coupling.

We consider two extreme connection geometries: in case (A) we take a one dimensional array of oscillators with nearest neighbor couplings, interpreting each unit as a hypercolumn in the visual cortex; while in (B) each unit is coupled to a common comparator which feeds back a function of the average phase. In both cases it is convenient to replace (1) by its continuum limit, in which the phase of the j^{th} oscillator in a chain of length $N+1$ at time t is described by the variable $\theta(x,t)$, $0 \leq x = j/N \leq 1$. The models are of the form:

$$(A) \quad \frac{\partial \theta}{\partial t} = \omega(x) + \frac{1}{N}\frac{\partial f}{\partial x}\left(\frac{1}{N}\frac{\partial \theta}{\partial x}\right) \tag{2}$$

$$(B) \quad \frac{\partial \theta}{\partial t} = \omega(x) + f(\theta - \int_0^1 \omega(s,t)ds). \tag{3}$$

The two architectures are shown in Figure 1 along with an input pattern consisting of two "spots" of size δ and intensity α. This stimulus is similar to that used by Gray *et al.* (1989) in recent experiments on cats.

Analysis of Model A

If the excitation is too "noisy" or the chain too long we cannot expect to find frequency locked solutions to model (A). This makes good sense; in fact we find that the "excited" units will "break away" and fire at a rate appreciably higher than that of the unstimulated oscillators. Ermentrout and Koppel (1986) have studied this effect in the case that the driving frequencies spatially have a linear gradient and show that there is a critical value of the gradient above which locking is lost and the chain breaks into domains which fire at different rates. Our simulations reveal that the same phenomenon of *frequency plateaus* occurs with excitations of the form presented in Figure 1.

We find that even relatively strong nearest neighbor coupling cannot adequately account for the tight, $\sim 0°$, phase differences observed between active units separated by large numbers of inactive (unstimulated) units.

Analysis of Model B

We first observe that, while we have presented model A as a linear chain, there is no intrinsic *geometry* in the coupling architecture: the placement of the excited units does not affect phase conditions, as it does in model A.

We have proven analytically and demonstrated in simulation that for small input amplitudes that excited units will exhibit phase-coupled collective oscillations that are *decoupled* from the background neurons. Not only do the excited units fire at the same rate , but they remain *exactly* in phase *regardless of their geometrical arrangement or separation.* For larger amplitude input the excited cells "break away" both in frequency and phase from the unstimulated neurons. The analysis easily generalizes to several excitation spots, but if there are different excitation levels α_i then phase differences will appear.

It has been argued on theoretical grounds by von der Malsburg (1986) that temporal synchronization of groups of neurons labels perceptually distinct objects, subserving figure-ground segregation. We will present a hybrid of models (A) and (B) that can perform such segregation and mimics the architecture of the early visual pathway.

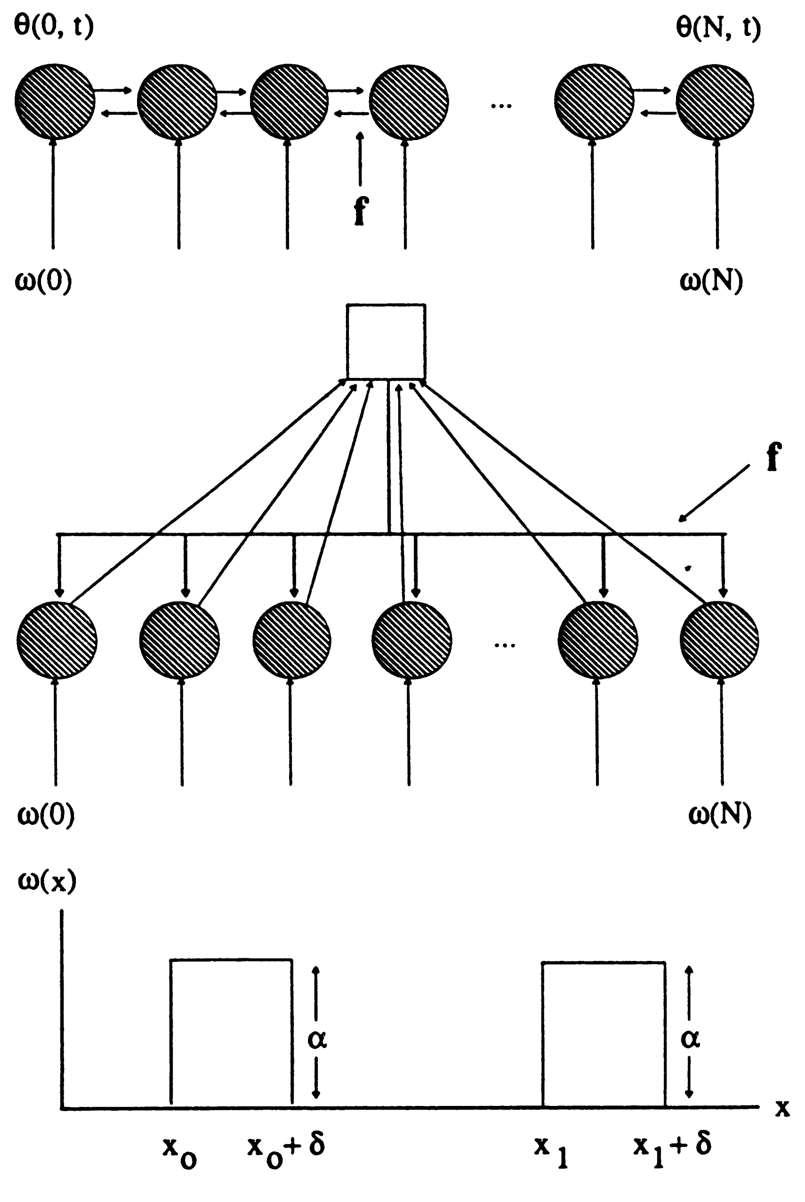


FIGURE 1. (Kammen, et al.)
TOP: network geometry (A)
MIDDLE: network geometry (B)
BOTTOM: input frequency (driving input) to the networks. See text for details.

References

A. M. Cohen, P. J. Holmes and R. H. Rand, (1982). *J. Math. Biol.*, **13**, 345.

Eckhorn, R., Bauer, R., Jordan, Brosch, M., Kruse, W., Munk, M. and Reitboeck, H. J. (1988). *Biological Cybern.*, **60**, 121.

W. J. Freeman (1978). *Elect. Clin. Neurophys.* **44**, 369 (1978).

C. M. Gray, P. König, A. K. Engel and Singer, W. (1989). *Nature*, **338**, 334.

N. Kopell and G. B. Ermentrout, (1986). *Comm. Pure Appl. Math.* **39**, 623.

Ch. von der Malsburg and W. Schneider, (1986). *Biol. Cybern.* **54**, 29.

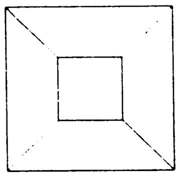
A MULTILAYER NEURAL NETWORK MODELLING THE PERCEPTUAL REVERSAL OF AMBIGUOUS PATTERNS

F. Masulli, M. Riani, and E. Simonotto
Dipartimento di Fisica - Universita' di Genova
Via Dodecaneso 33, 16146 Genova, Italy

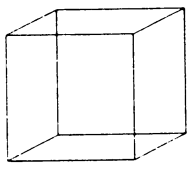
The capability of a multilayer neural network (based on a modified BSB model) for reproducing the stochastic dynamics of the perceptual reversal of ambiguous figures is assessed. Computer simulation results, as well as experimental data, are well fitted by a Gamma distribution.

1. Introduction

When an ambiguous pattern, such as the Necker cube or the Mach pyramid (Fig.1), is observed, the same visual input can elicit two different interpretations, giving rise to a cyclic perceptual alternation of such competitive interpretations. This repetitive cognitive behaviour can be regarded as the basic feature of the perceptual alternation phenomenon.



Mach Pyramid



Necker Cube

Fig. 1

During a prolonged observation of an ambiguous drawing [1,2], a stationary phase is reached in which both percepts appear with some regularity, and the perceptual durations of the competitive interpretations are well represented by a Gamma distribution, with a mean time normally ranging from few to about ten seconds . The analytic form of a Gamma distribution is:

$$p(t)\,dt = \frac{b^n t^{n-1} \exp(-bt)}{\Gamma(n)}\,dt \qquad (1)$$

where $\Gamma(n)$ is the Euler-Gamma function.

In this paper, we propose a multilayer neural network model that is able to describe the main characteristics of the perceptual alternation phenomenon [3] and, in particular, the stochastic distribution required.

2. A single layer model of perceptual alternation

In a previous work [3] we described a single-layer neural network (SLN) model of perceptual alternation that was based on the "Brain State in a Box" (BSB) model proposed by Anderson and coworkers [4,5]. The SLN is the basic building block of the multilayer model that we present in the next section. Here we summarize and discuss the principal characteristics of the single-layer network.

The recognition processes related to an ambiguous pattern can be modelled by an autoassociative neural network in which the features characterizing both alternative interpretations must be coded in the activities of the network "neurons". So the state vector $\vec{f}$ of the network can be seen as composed of two subvectors, $\vec{f}_A$ (the first l components of $\vec{f}$) and $\vec{f}_B$ (the last $m - l$ ones), associated with the features of the two alternative interpretations, A and B, of the ambiguous pattern. Furthermore, such interpretations exclude each other, and can never be present together; hence, it is important that the excitation of the subvector $\vec{f}_A$ should exert an inhibiting influence on the subvector $\vec{f}_B$, and vice versa.

The connection matrix, C, is obtained by learning, through some trials, the two competitive interpretations, A and B, using a generalized Hebb rule. The matrix contains two square blocks, E_{AA} and E_{BB}, representing the positive autoconnections of each subvector ($\vec{f}_A$ or $\vec{f}_B$) to itself, and two rectangular blocks, I_{AB} and I_{BA} ($I_{AB} = I_{BA}^T$), which are the inhibitory connections of $\vec{f}_B$ to $\vec{f}_A$, and vice versa, that is:

$$C = \begin{pmatrix} E_{AA} & -I_{AB} \\ -I_{BA} & E_{BB} \end{pmatrix} \tag{2}$$

In this way, the SLN is able to reinforce the features of only one interpretation, while the features of the other are weakened; hence, when a constant input, $\vec{G}$, representing a static ambiguous pattern is presented, the system can reach a stable state (i.e., a corner of the box in BSB model), corresponding to one of two alternative percepts , in which every neuron of the related subvector is firing at its maximum rate . At this point, in order that the system may simulate an experimental cyclic behaviour, we assume that, once the system has reached a corner, a habituation process becomes effective for a fixed time lag, giving rise to a continuous decrement in the components of that subvector, which leaves the corner. As a result, the subvector representing the alternative part of the state vector becomes dominant and further decreases the activity of the other subvector.

The dynamical evolution of the i-th component of the state vector (i.e. the activation value of the corresponding neuron), from time t to time $t + \tau$, can be expressed by the equation:

$$f_i(t+\tau) = LIMIT\left[\left(\sum_{j=1}^{m} C_{ij} f_j(t) + \beta G_i\right)\left(1 - \sigma_i(t)\right)\right] \tag{3}$$

where $LIMIT$ is a function limiting the values of the state vector components to those ranging between zero and one, C_{ij} is the element of the connectivity matrix and βG_i is the i-th component of the stimulus multiplied by a constant parameter β and $\sigma_i(t)$ stands for the habituation process. The effect of this process is to lower the input sensitivity of the neurons. Accordingly, $\sigma_i(t)$ is usually zero but when the $LIMIT$ function becomes active, it assumes the value $\sigma = \sigma_o$ ($\sigma_o \in (0,1)$) for a fixed time lag.

A plausible value of the unit time τ is about a tenth of a second. In fact, to use continuous values of the neurons' activation, that is their firing rate, one must integrate the instantaneous activity of neurons over a suitable time interval (τ).

Computer simulations of the network behaviour [3] have shown the existence of a stable limit cycle in which the two percepts, A and B, alternate periodically; furthermore, this network exhibits considerable robustness to noise. In fact, the addition of a biologically plausible synaptic noise does not change significantly the temporal evolution of the SLN.

3. The multilayer model and its stochastic dynamics.

In order that the system may exhibit a stochastic behaviour, we designed a multilayer neuronal network (MLN) with a single-layer network as basic element. Such improvement involves inserting in the model the probable redundancy of the neural assemblies acting as "recognizers" in the brain [6]; then, an ambiguous input stimulus, $\vec{G}$, can be shifted among such parallel recognizers, as a consequence, for instance, of eye movements.

We designed a two layers network. The lower layer is made up of r SLNs working in parallel, without any interconnections. The probability, $p(t)$, that the input $\vec{G}$ is present in the k-th SLN of the lower layer can be expressed as:

$$p(t) = p(ON/ON)p(t-\tau) + p(ON/OFF)(1 - p(t-\tau)) \tag{4}$$

where $p(ON/ON)$ is the transition probability from the state ON (input present) to the state ON and $p(ON/OFF)$ is the transition probability from the state OFF (input absent) to the state ON . We chose such transition probabilities that, on average, the stimulus is present in only one SLN of the lower layer for every time step. When the stimulus is not present in the k-th SLN, the input vector is the null vector.

The upper layer consists of a single basic unit (SLN). The input to the i-th neuron, $G_i^U(t)$, is the sum of the activities of the corresponding neurons in the lower layer:

$$G_i^U(t) = \sum_{k=1}^{r} f_i^k(t) \tag{5}$$

We assumed that the perceptual interpretations, A and B of the ambiguous pattern resulting from this MLN are associated with the temporal evolution of the upper layer; then the system perceives A (or B) if the sum $\varphi_A(t)$ of the activities of the neurons of the subvector $\overrightarrow{f_A^U}$, normalized to one, is greater than the corresponding sum $\varphi_B(t)$ for the subvector $\overrightarrow{f_B^U}$(or vice versa $\varphi_B(t) > \varphi_A(t)$).

A preliminary test of the MLN behaviour , was performed, via computer simulations, using the following parameter values : $m = 10$; $r = 10$; $\beta = 0.9$; $\sigma_o = 0.6$ or $\sigma_o = 0.65$ or $\sigma_o = 0.7$ (for a time lag of 30 times τ). The parameter l could assume two values: $l = 6$ and $l = 5$ (perfect symmetry between A and B). If $l = 6$, $G_i = 0.02$ for $i \leq 6$ and $G_i = 0.03$ for $i \geq 7$; if $l = 5$, $G_i = 0.02$ for every i value. The connection matrix C was symmetrical with $C_{ii} = 0$.

When white noise (of the order of 0.4 of the connection value) is added to the connection matrix C, the MLN exhibits a stochastic dynamical behaviour. After a short transient period, a stationary phase is reached in which the two percepts alternate (Fig 2a), and the durations of each percept, (i.e. the time interval in which $\varphi_A > \varphi_B$ or vice versa) are distributed around their mean values according to a stochastic distribution that is well fitted by a Gamma distribution (Fig. 2b).

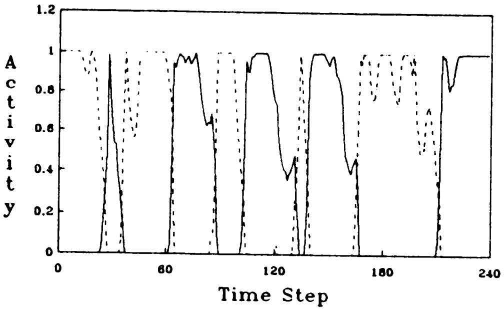


Fig. 2a

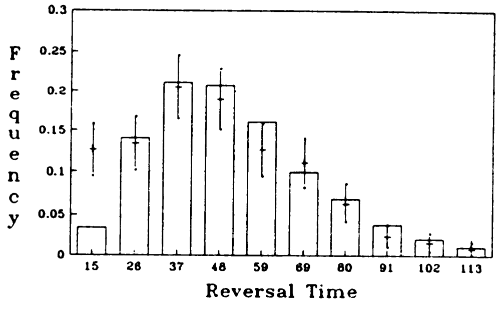


Fig. 2b

Fig. 2a Temporal evolutions of φ_A (continuous line) and φ_B (dashed line) during 240 iterations of one run of the computer simulation, for $l = 6$ and $\sigma_o = 0.6$.

Fig. 2b Comparison between the stochastic distribution of the reversal times of percept B, as obtained from the same run of the computer simulation as in fig. 2a, and the corresponding Gamma distribution histogram.

The values of the parameters b and n of the Gamma distributions, obtained by various computer-simulation runs, range from 0.01 to 0.1 τ^{-1} and from 2 to 5 respectively. In order to compare the simulation results with experimental ones [1,2], we point out that, if we choose the iteration time τ equal to 0.1 seconds, the mean duration times of computer simulations are of the order of a few seconds. This choices allows both the mean duration times and the values of the Gamma parameters to be very close to experimental ones [1,2].

References

1. Borsellino, A., De Marco, A., Allazetta, A., Rinesi, S., and Bartolini, B., "Reversal Time Distribution in the Perception of Visual Ambiguous Stimuli," Kybernetik, 10, 139-144 (1972).
2. De Marco, A., Penengo, P., Trabucco, A., Borsellino, A., Carlini, F., Riani, M., and Tuccio, M.T., "Stochastic Models and Fluctuations in Reversal Time of Ambiguous Figures", Perception, 6, 645-656 (1977).
3. Riani, M. and Masulli, F., "Modelling Perceptual Alternation by using ANN's" , in "Parallel Architectures and Neural Networks II", E.R. Caianiello ed., World Scientific, in press.
4. Anderson, J.A., Silverstein, J.W., Ritz, S.A. and Jones, R.S., "Distinctive Features, Categorical Perception and Probability Learning", Psychol. Rev., 84, 413-451 (1977).
5. Kawamoto, A.H. and Anderson, J.A., "A Neural Network Model of Multistable Perception", Acta Psychol., 59, 35-65 (1985).
6. Edelman, G.M., "Group Selection as the Basis for Higher Brain Function" in "The Organization of the Cerebral Cortex", Schmitt, Worden, Adelman, and Dennis, eds., The MIT Press (1981).

Learning from natural selection in an artificial environment

David H. Ackley
Michael S. Littman
Bell Communications Research
Cognitive Science Research Group

Abstract

The process of natural selection is clearly a source of information about the performance of an individual organism, but — since the signal for failure is death — it is not immediately apparent how it could be exploited to perform learning during an individual's lifetime. This paper defines and demonstrates a strategy called *evolutionary reinforcement learning* (**ERL**) that combines genetic evolution techniques with neural network learning techniques to allow effective learning based only upon natural selection. The strategy is demonstrated in an artificial life environment called **AL**, using computer simulations that span four orders of magnitude in space and six orders of magnitude in time. Successful individuals may achieve lifetimes of 25,000 steps or more, and initial populations that develop long-term viability may descend through 300 generations or more before arriving at the one million step simulation limit.

This paper is a summary of ongoing work that will be described more fully in a future report [1].

Natural selection as teacher

Learning algorithms vary in the amount and nature of the feedback they require to function. For example, supervised paradigms supply correct answers as feedback; the system must learn to produce them on demand. Reinforcement paradigms supply less — only judgments of right or wrong — so the system must first discover and then remember the correct responses. The paradigm of natural selection supplies still less — only birth and death. How can an organism learn in such circumstances, where the only unarguable sign of failure is the organism's own death, and the reproduction process preserves only the "genetic code," which is unaffected by any learning performed during the organism's life?

This paper describes experiments with adaptive "*agents*" controlled by simple neural networks. A population of *agents* struggles for existence in a simulated world in which the only feedback mechanism is natural selection. Sufficiently healthy and well-fed *agents* produce offspring; less effective ones are killed in battle or starve.

A strategy for adaptation called *evolutionary reinforcement learning* (**ERL**) is explored. The primary new contribution of **ERL** is that the "genetic code" subject to evolution specifies not only an *action function* to determine behavior, but also an *evaluation function* [6]. Changes in evaluations from step to step produce reinforcement signals that drive individual *agent* learning. In simulation studies, **ERL** displays better performance compared to control populations of randomly moving agents, and compared to populations that employ just evolution.

It is important to recognize that natural selection, when viewed as a computational paradigm for search and learning, places severe restrictions on possible adaptation strategies. There are only two circumstances in which a strategy has decisions to make. The first situation — concerned with learning — is the choice of behavior for a given agent at a given timestep, and the second situation — concerned with evolution — is the passage of genetic information to the offspring when a birth occurs. Everything else is determined by the "laws of nature" of the world at hand. For example, death requires no action on the part of the strategy. Also, in contrast to conventional genetic algorithms [10, 7], a strategy is not free to specify the existence and maintenance of any particular population size. Similarly, a strategy is not free to determine who lives, who dies, and who reproduces. The strategy influences such decisions only indirectly, via the interactions between the (static and dynamic) properties of the world and the behavior of the *agents* governed by the strategy.

ERL: Evolutionary reinforcement learning

At Birth

Given: A parent agent A and an offspring O to be initialized.

B1. *Clone.* Copy A's genetic code to O. If there is one or more other agents within a prespecified distance of A, pick the closest such agent B and go to B2, otherwise go to B3.

B2. *Crossover.* Modify O's genetic code by crossing with B's using two random crossover points.

B3. *Mutate.* With low probability mutate O's genetic code by flipping random bits.

B4. *Elaborate.* Translate O's genetic code into initial weights for O's evaluation and action networks.

Living at time t:

Given: A living agent A, and a new current input vector I_t.

L1. *Evaluation.* Propagate I_t through the evaluation network producing a scalar evaluation E_t.

L2. *Learning.* If this is A's day of birth, go to L3. Otherwise, produce a reinforcement signal by comparison with the previous evaluation: $R_t = E_t - E_{t-1}$. Use the **CRBP** learning algorithm to update the action net with respect to the previous action X_{t-1} and previous input I_{t-1}.

L3. *Behave.* Use the **CRBP** performance algorithm — a standard forward propagation followed by stochastic output units — to generate a new action X_t based on I_t. Perform the chosen action.

Figure 1. Summary of **ERL**.

Evolutionary reinforcement learning

Evolution (adaptation of a population) and learning (adaptation of an individual) are both important natural processes through which behavior is optimized for survival. The field of *genetic algorithms* [10, 7] focuses on evolutionary issues, while much of the research into neural networks [12, 8] has focused on learning issues. **ERL** is one of a number of approaches that in various ways combine these two areas, e.g. [3, 9, 13].

Figure 1 summarizes the **ERL** algorithm. In principle, any associative reinforcement learning algorithm supporting multiple output bits [5, 4] could be used in steps L2 and L3; we have employed an existing reinforcement learning algorithm called **CRBP** [2] for that purpose. **CRBP** and all reinforcement learning algorithms require the presence of a *reinforcement function* which they attempt to optimize. Evolution's job is to discover useful evaluation functions — functions that produce reinforcement signals that cause agents to learn to be successful under the influence of **CRBP**.

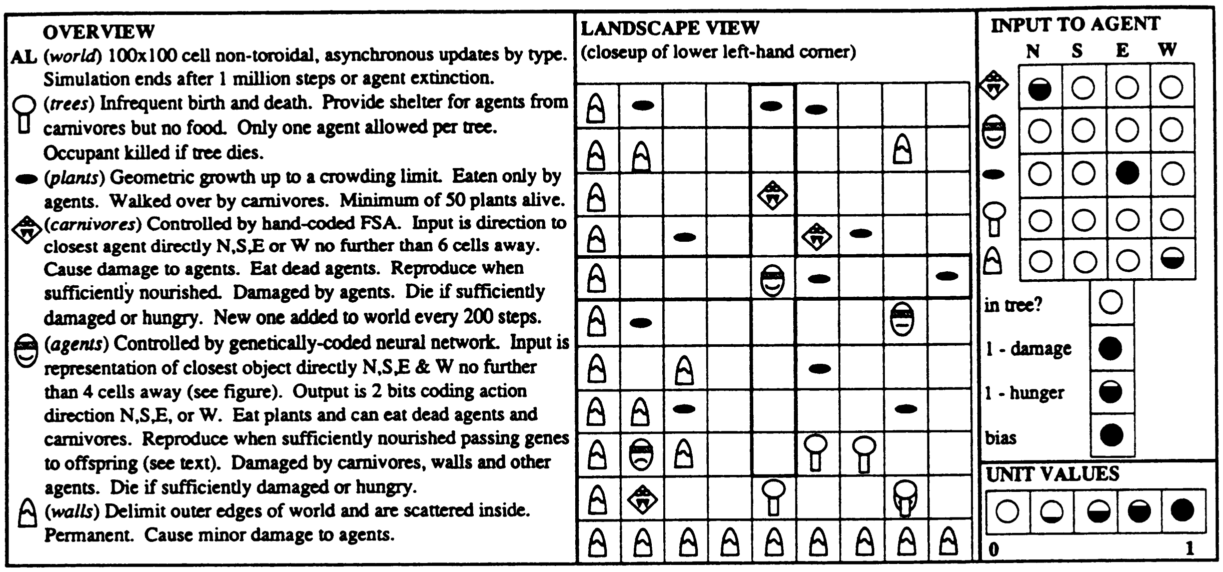
World AL

Figure 2 summarizes the basic characteristics of **AL**. **AL** shares many characteristics with other artificial environments [11, 14, 16] that have been explored. Here, we just highlight a few important aspects:

- The *agents* (and *carnivores*) have fixed orientations in space and possess four "eyes," allowing them to see the nearest object within a given range along each of the compass directions. Actions are also globally oriented: two bits coding for north, south, east, and west. Fixed orientation is one of a number of design decisions aimed at simplifying life for the *agents*, allowing them to make progress even with one-layer, non-recurrent networks, and consequently allowing us to run relatively large simulations for relatively long periods of time.

- No *physical* evolution is allowed. All the adaptive agents employ the same "standard chassis," and both evolution and learning operate solely at the level of behavioral control. Although evolution of physical structures is manifestly of tremendous importance in the natural world, our focus with **AL** is on "intellectual" development.

- The "genetic code" defining an *agent* is fairly large — over 280 bits — compared to research efforts such as [14, 15]. Evolution is posed a non-trivial optimization task.


OVERVIEW

AL (*world*) 100x100 cell non-toroidal, asynchronous updates by type. Simulation ends after 1 million steps or agent extinction.

(*trees*) Infrequent birth and death. Provide shelter for agents from carnivores but no food. Only one agent allowed per tree. Occupant killed if tree dies.

(*plants*) Geometric growth up to a crowding limit. Eaten only by agents. Walked over by carnivores. Minimum of 50 plants alive.

(*carnivores*) Controlled by hand-coded FSA. Input is direction to closest agent directly N,S,E or W no further than 6 cells away. Cause damage to agents. Eat dead agents. Reproduce when sufficiently nourished. Damaged by agents. Die if sufficiently damaged or hungry. New one added to world every 200 steps.

(*agents*) Controlled by genetically-coded neural network. Input is representation of closest object directly N,S,E & W no further than 4 cells away (see figure). Output is 2 bits coding action direction N,S,E, or W. Eat plants and can eat dead agents and carnivores. Reproduce when sufficiently nourished passing genes to offspring (see text). Damaged by carnivores, walls and other agents. Die if sufficiently damaged or hungry.

(*walls*) Delimit outer edges of world and are scattered inside. Permanent. Cause minor damage to agents.

Figure 2. Summary of World **AL**.

An **AL** oversight routine employs "spontaneous generation" to ensure that neither *plants*, *trees*, nor *carnivores* become extinct, but *agents* are given no such safety net. How long can an initial population of *agents* expect to survive in World **AL**?

Results

Simulations of **ERL** in **AL** display phenomena at several time scales. Observing at highest resolution, *agents* are seen moving about or collecting in corners, feeding or starving, encountering *carnivores* and escaping or not, and so on. **AL** is not an overly kind world: Most initial *agent* populations die out quite quickly. Observing summary statistics at the x100 time scale, in those populations that survive the most apparent features are irregular predator-prey oscillations involving *plants*, *agents*, and *carnivores*, interspersed with periods of stable or slowly changing population sizes. The few runs that survived to the one million step simulation limit (e.g., see the x1000 views in Figure 3) generally possessed *agent* population sizes that oscillated in the 30–60 range.

We compared five strategies:

B (Brownian) Non-evolving, non-learning, *agents* with uniform random actions,

F (Fixed) Non-evolving (no crossover or mutation), non-learning, *agents* with random action networks,

E (Evolve) Non-learning *agents* with evolution of action networks,

L (Learn) Non-evolving *agents* with learning based on a fixed random evaluation network, and

ERL *agents* with learning in the action network and evolution of both networks.

We ran each strategy 100 times, varying only the initial random seed. Runs were truncated at one million steps when necessary — twice for strategy *L* and seven times for strategy *ERL*. The mean population survival times were

$$B\,6{,}560, \qquad F\,1{,}562, \qquad E\,1{,}564, \qquad L\,47{,}529, \qquad ERL\,80{,}707.$$

The distributions are highly skewed due to frequent "infant mortality" — i.e., quick population extinctions — combined with very long tails. Figure 4 displays a cumulative plot of all the data from the 500 runs. It appears that the strategies fall into two broad groups: *E* and *F* on the one hand, and *B*, *L*, and *ERL* on the other. Strategy *B* does better than might be expected because uniform random actions will tend to spread *agents* evenly in the non-toroidal world, whereas the other strategies in general display biased behavior patterns, risking increased damage from *walls* and spot famines due to local overgrazing.

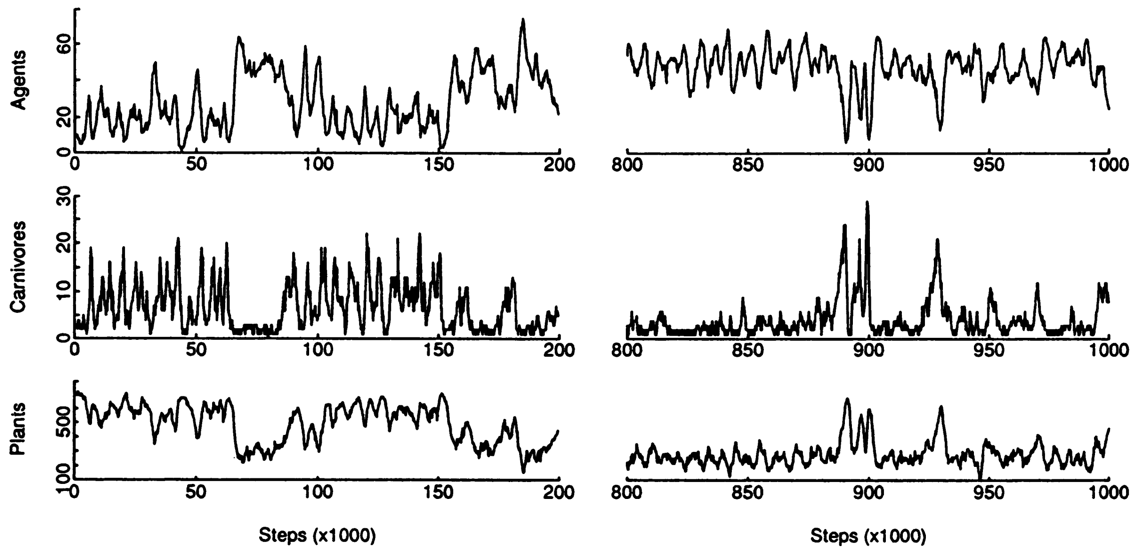


Figure 3. ERL in AL: Species population sizes vs time for a long-term successful initial *agent* population (first and last 20% of run shown).

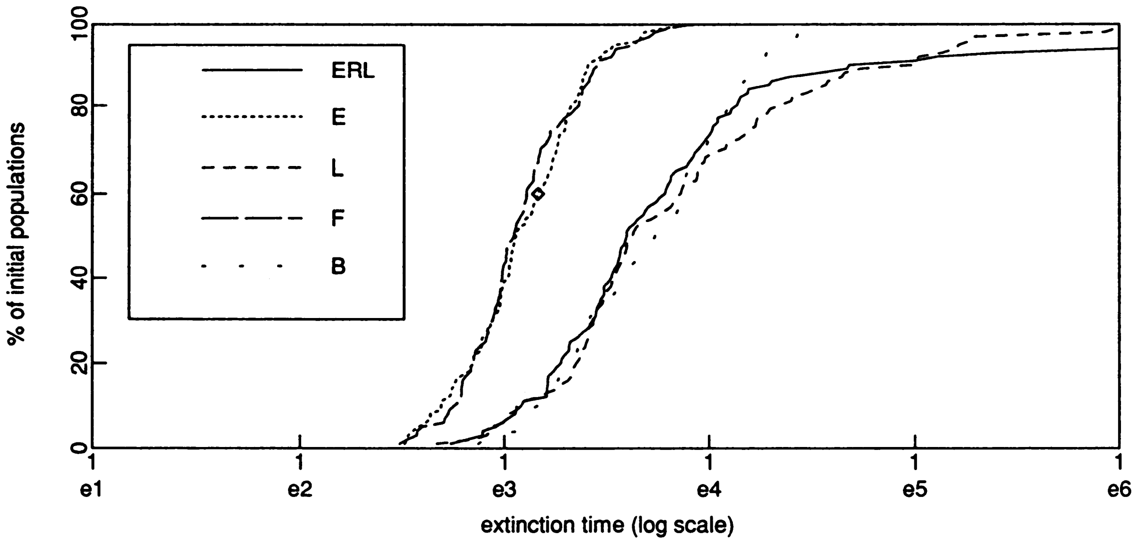


Figure 4. Cumulative plots showing the distributions of population lifetimes generated by the five strategies. The point marked with a diamond, for example, indicates that 60% of the strategy *E* initial populations were extinct by about 1500 timesteps.

A median test verifies that strategy *E* differs from *ERL* ($p < .001$). In addition, above the seventy-fifth percentile *ERL* pulls away from *B*. This is also supported by a median test ($p < .005$). *L* and *ERL* remain indistinguishable up to the ninetieth percentile.

Discussion

There are many interesting issues raised by the simulation results; here we touch briefly on a few. Most striking to us was that evolution without learning did so poorly, and that learning without evolution did so well. The former result was surprising since evolution without learning is the typical approach to artificial life, and the latter result was surprising since, without evolution to improve the evaluation functions, strategy *L* can never move beyond on the randomly-generated evaluation functions found in the initial populations.

We hypothesized that evolution alone has difficulty because survival in **AL** is no trivial matter: Most *agents* with randomly generated action networks die quickly (*viz.* the strategy *F* results). Consequently, most viable populations that do develop descend from a single individual in the initial population, and thus all are close genetic kin. In such circumstances, genetic recombination has little effect. To explore this hypothesis, we investigated variant **AL** worlds in which survival was easier at first and gradually became more challenging. We found that if we slowly decreased the *plant* density while slowly increasing the range of *carnivore* vision (world **AL.1**), strategy *ERL* did display a significant advantage over strategy *L*, indicating a benefit for evolution at least combined with learning.

The success of learning alone was noteworthy. It is easy enough to conclude merely that the space of genetic codes for action networks is more difficult to search than the code space for action-plus-evaluation networks, so that strategy *L* could simply "luck into" good initial populations often enough to make the difference. However, that cannot be the whole story. After all, the code space for *L* is thirty orders of magnitude larger than that for *E*, so one might expect it to be harder to search. Our explanation is that *it is easier to generate a good evaluation function than a good action function.*

Notice, for example, that there are two output units in the action network, but only one in the evaluation network. To specify an action in response to a particular input requires specifying two weights, but to specify that a particular input is "good" requires only one weight. Furthermore, if the evaluation function specifies that the energy level input is positively valued, then there is pressure towards making "eating moves" more probable regardless of the direction of the food source. One such evolutionarily specified weight can have the effect of specifying all eight weights involved in response to *plants*. The insight that strategy *L* highlights is that it can be much easier to specify *goals* than *implementations* — assuming, of course, the existence of a search and learning process adequate to fill in the details.

Even though **AL** is by necessity and by design very much a toy world, within its limited spatial and temporal confines a wide range of phenomena occur. Population survival time is a very natural performance measure, and obviously relevant to the human species. Evolution and learning speak to each other across the gulf of a time scale, and with deep temporal simulations such interactions can be studied. In the field of neural network research, where a predominant urge seems to be for learning algorithms to converge faster, this research suggests another possible goal: to be interesting longer.

Acknowledgments

Tom Landauer, George Furnas, the Cognitive Science Research group, and other members of the technical staff at Bellcore contributed valuable ideas and statistical help. Any errors of substance or style, however, are due to the authors.

References

[1] Ackley, D.H., & Littman, M.S. Evolutionary reinforcement learning. In preparation, 1990.

[2] Ackley, D.H., & Littman, M.S. Generalization and scaling in reinforcement learning. Submitted to Neural Information Processing Systems, 1989.

[3] Ackley, D.H. A connectionist machine for genetic hillclimbing. Kluwer Academic Press, Boston, 1987.

[4] Anderson, C.W. . Learning and problem solving with multilayer connectionist systems. U of Mass. Ph.D. COINS TR 86-50, 1986.

[5] Barto, A.G. Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology*, 4:229–256, 1985.

[6] Berliner, H.J., & Ackley, D.H. The QBKG system: Generating explanations from a non-discrete knowledge representation. *AAAI-82*, Pittsburgh, PA, 213–216, 1982.

[7] Goldberg, D. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.

[8] Grossberg, S. *Studies of mind and brain*. Reidel, 1982.

[9] Hinton, G.E. and Nowlan, S.J. How learning can guide evolution. *Complex Systems*, 1, 495–502, 1987.

[10] Holland, J.H. *Adaptation in Natural and Artificial Systems.* U of Mich. Press, 1975.

[11] Langton, C.G. (Ed.) *Artificial Life*, Addison-Wesley, 1989.

[12] McClelland, J.L., & Rumelhart, D.E. (Eds.). *Parallel Distributed Processing: Explorations in the microstructures of cognition.* Three volumes. The MIT Press (A Bradford Book), 1986.

[13] Montana, D.J. and Davis, L., Training feedforward neural networks using genetic algorithms. (To appear) IJCAI, 1989.

[14] Packard, N.H. Intrinsic adaptation in a simple model for evolution. In C.G. Langton, (ed.) *Artificial Life*, Addison-Wesley, 1989.

[15] Taylor, C.E., Jefferson, D.R., & Goldman, S.R. RAM: Artificial life for the exploration of complex biological systems. In C.G. Langton, (ed.) *Artificial Life*, Addison-Wesley, 1989.

[16] Wilson, S.W., Knowledge growth in an artificial animal. Proc. of an Int. Conf. on Genetic Algorithms and Their Applications. 16–23, Pittsburgh, PA, 1985.

GENETIC PROGRAMMING

Modular Neural Evolution for Darwin Machines

Hugo de Garis

Machine Learning and Inference Laboratory,
Artificial Intelligence Center, George Mason University,
4400 University Drive, Fairfax, Virginia 22030, U.S.A.
email HUGODEG@GMUVAX2.GMU.EDU

Keywords :

Genetic Programming, Neural Networks, Genetic Algorithm, Modular Neural Evolution, Agents, Society of Mind, GenNets(Neural Networks designed with the Genetic Algorithm), Hierarchies of Neural Modules, Sequential Evolution, Genetic Programming Software Shells, Darwin Machines, Wafer Scale Integration, Nanotechnology.

Abstract :

This paper introduces the concept of Genetic Programming, which employs the Genetic Algorithm (GA) [GOLDBERG 1989] to design both Neural Network (NN) [RUMELHART et al 1986] modules (GenNets) and their control circuits. The GA is used to find the weights (and their excitory-inhibitory signs), of fully connected neural networks with feedback. Once a GenNet module performs sufficiently well, its weights are frozen, and the module is then used as a component in more complex circuits. The outputs of NN control circuits are the inputs to these frozen modules. Once the "control circuit plus modules" (considered as a unit), functions as desired, the weights of the control circuit are then frozen. This larger frozen unit can be considered as a component for a yet higher stage of design. This hierarchical module building is similar to Minsky's "Society of Mind" theory, where a GenNet module is equivalent to his concept of an "agent", and Genetic Programming is related to his idea of "mind design" [MINSKY 1986, 1988]. A second concept is also briefly introduced, namely that of the Darwin Machine, which performs GenNet evolution directly in hardware. WSI, (wafer scale integration) (now) and nanotechnology (later), will allow such machines to be built.

Introduction :

The conceptual problems involved in designing and controlling neural computers with m(b,tr)illions of processors can be discussed today. It will be impossible to program each processor individually, and the internal dynamics and connections between processors will be too complex to analyse. Several conclusions and suggestions result from this.

a) Modules of neural nets will need to be treated as black boxes. Only their performance will be of concern. Full analysis of their internal behaviour will have to be abandoned.
b) Neural modules will be designed by the Genetic Algorithm, using coded chromosomes which will compete with each other to reproduce, according to the quality of their performance.
c) These neural modules (agents) will be combined to form functional hierarchies ("agencies", a la Minsky) using neural control circuits which themselves will be evolved by the Genetic Algorithm.

Modular Neural Evolution :

To illustrate the above ideas, a simple example of modular neural evolution is presented. It is the two-eye, two-joint robot arm positioning simulation problem. FIG.1 shows the basic setup. The aim of the task is to move the robot arm from its vertical start position X to the goal position Y. J1 and J2 are the joints, E1 and E2 are the eye positions, and JA1, JA2, EA1 and EA2 are the joint and eye angles of the point Y.

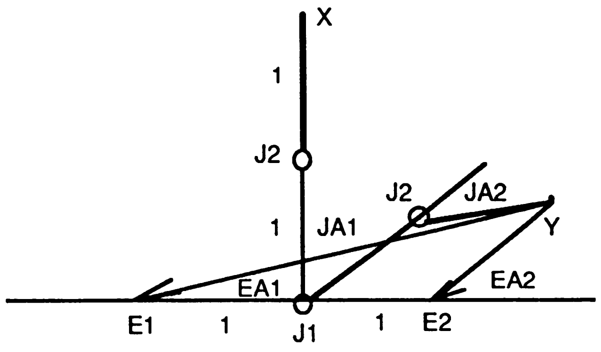


FIG. 1

The modular approach is illustrated by specifying that two different neural net modules will be evolved. The first, called the "joint module", controls the angle JA that a given joint opens to, for an input control signal of a given strength - and the second, called the "control module", receives inputs EA1 and EA2 from the two eyes and sends control signals to the joints J1 and J2 to open to angles of JA1 and JA2.

FIG.2 shows the basic circuit design that the GA uses to find the "joint" and "control" modules. The joint modules (two identical copies) are evolved first, and are later placed under the control of the control module. Each module (containing a user specified number of neurons) is fully connected, including connections from each neuron to itself. Between any two neurons are two connections in opposite direction, each with a corresponding (signed) weight. The input and output neurons also have "in" and "out " links but these have fixed weights of 1 unit. The outputs of the control module are the inputs of the joint modules as shown in FIG. 2.

The aim of the exercise is to use the Genetic Algorithm to choose the values of the signs and weights of the various modules, such that the overall circuit performs as desired. Since this is done in a modular fashion, the weights of the joint module are found first. These weights are then frozen, and the weights of the control circuit found so that the arm moves as close as possible to any specified goal point Y. Each weight is coded (onto a GA chromosome) with a sign, (where 0 means an excitory synapse, 1 means an inhibitory synapse) followed by a user specified number of places after the "binary point". For example, an inhibitory weight of 5 binary places, having value 101101 would take the binary value -0.40625. A chromosome coding for a module of N neurons (hence N*N signs and weights) would have a total length of N*N*6 binary positions. All weights are expressed to the same number of places (e.g. 5).

The activation of each neuron is determined in the usual way, namely the sum of the products of the incoming signal strengths and the corresponding weights of the connections. The neuronal transfer function was chosen to be (2/(1+ exp(-actvn))) - 1, to give an output with a range of -1 to +1. Weight values also ranged between -1 and +1, so as to avoid unbounded output values. With both weights and transfer functions restricted to the -1 to +1 range, output values stabilised, (usually after about 50 cycles or so for 1% accuracy). In each cycle, the outputs are calculated from the inputs (which were calculated in the previous cycle). These outputs become the input values for the neurons that the outputs connect to.

The Genetic Algorithm is then used to choose the values of the weights, such that the actual output is as close as possible to the desired output. To evolve the joint module, 21 input values ranging from -1 to +1 in steps of 0.1, were used. The desired output values were chosen to be half the input values, thus ranging from -0.5 to +0.5, and were interpreted as being the number of turns of a joint, (e.g. +0.5, i.e. half a turn, would mean a joint angle of 180 degrees. A positive angle is clockwise).

No crossover or inversion was used in the Genetic Algorithm, [GOLDBERG 1989], since the problem of neural network design is so nonlinear. Changing one weight influences the outputs of all the neurons. Hence the GA used only mutation (a small probability (e.g.0.001) that each binary value, whether sign or weight, on a chromosome would flip) and selective reproduction, (i.e. those chromosomes obtaining a superior score compared to others in the population, reproduces in proportion to their superiority). The quality measure used in the evolution of the joint module was the inverse of the sum of the squares of the differences between the desired and the actual output values.

FIG. 3 shows the set of points Y used to evolve the control module. These points all lie within a radius of 2 units, because the length of each arm is 1 unit. For each of these 32 points, a pair of "eye angles", EA1 and EA2, is calculated and converted to a fraction of one half turn. These values are used as inputs to the control circuit. The resulting 2 "joint angle" output values JA1 and JA2, are then used to calculate the actual position of the arm Y', using simple trigonometry. The quality of the chromosome which codes for the signs and weights of the control module is the inverse of the sum of the squares of the distances between the 32 pairs of actual positions Y' and the corresponding desired positions Y.

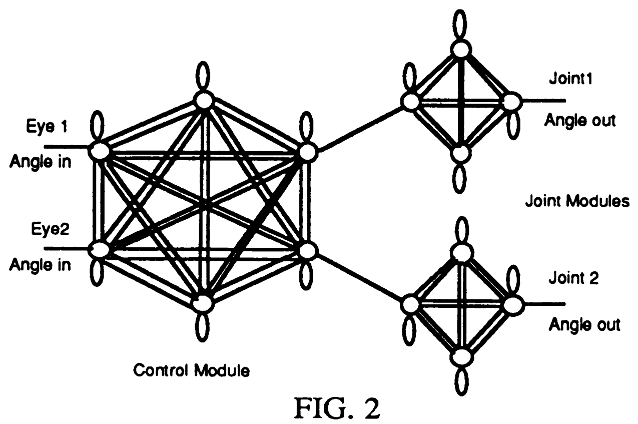


FIG. 2

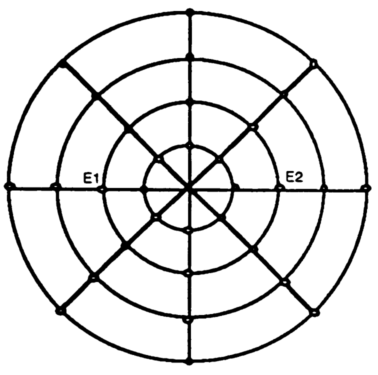


FIG. 3

WEIGHTS		FROM NEURON			
		0	1	2	3
TO NEURON	0	-0.96875	0.84375	-0.875	0.90625
	1	-0.96875	-0.78125	-0.875	-0.78125
	2	-0.9375	-0.71875	-0.75	-0.28125
	3	-0.15625	-0.9375	0.3125	-0.65625

FIG. 4

FIG. 4 shows an example solution for the 16 weight values of the joint module. With these weights (obtained after roughly 100 generations with the GA), the actual output values differed from the desired values by less than 1%. Similar solutions exist for the 36 weights of the control module, which also gave distance errors less than 1% of the length of an arm (1 unit). What was interesting was the fact that every time a set of weights was found for each module, a different answer was obtained, yet each was fully functional. The impression is that there may be a large number of possible adequate solutions, which gives Genetic Programming a certain flexibility and power.

Darwin Machines :

Genetic Programming is a new programming methodology which requires a full research program to develop it. More ambitious projects employing Genetic Programming need to be undertaken, such as trying to design a time dependent system which balances and walks, or a system which can detect certain kinds of objects. Such systems would be hierarchical in nature and appropriate for Genetic Programming.

Within 5 years, it will be possible to put 10 million artificial neurons on a Wafer Scale Integration (WSI) superchip [RUDNICK et al 1989]. Thus machines can be built to implement GP directly in hardware.These machines have been called Darwin Machines in this paper. Later still, nanotechnology (molecular scale engineering) [DREXLER 1986, REED 1988, LANGTON 1989, SCHNEIKER 1989] will be able to build Darwin Machines on a much more impressive scale.

Using Darwin Machines, Genetic Programmers will be able to specify, in a very high level language, such things as functional requirements for GenNets, output/input connections between modules, GA parameter values, number of neurons per module etc. The Darwin Machine will then perform the Genetic Programming directly in hardware and at great speed. It is likely that Darwin Machines will be incorporated as components in real time devices, such as walking robots, and that real time inputs will be fed directly to these Darwin Machines for evaluation. In the immediate future however, industrialists will be able to construct Genetic Programming Software Shells which will perform many of the functions mentioned above, but at a slower software pace.

References :

[DREXLER 1986] "Engines of Creation : The Coming Era of Nanotechnology", K.E. Drexler, Doubleday, 1986.

[GOLDBERG 1989] “Genetic Algorithms in Search, Optimization, and Machine Learning”, D.E. Goldberg, Addison-Wesley, 1989.

[LANGTON 1989] "Artificial Life", C.G. Langton ed., Addison Wesley, 1989.

[MINSKY 1986] "Society of Mind", M. Minsky, Simon and Schuster, 1986.

[MINSKY 1988] See the preface in [WALTZ et al 1988].

[REED 1988] "Quantum Semiconductor Devices", M.A. Reed, in "Molecular Electronic Devices", F.L. Carter, R.E. Siatkowski, H. Wohltjen eds. North Holland, 1988.

[RUDNICK et al 1989] “ An Interconnect Structure for Wafer Scale Neurocomputers”, M. Rudnick and D. Hammerstrom, in Proceedings of the 1988 Connectionist Models Summer School 1988, eds D. Touretzky, G. Hinton, T. Sejnowski, Morgan Kaufmann, 1989.

[RUMELHART et al 1986] "Parallel Distributed Processing", Rumelhart D.E., McClelland J.L., Vols 1 & 2, MIT Press, 1986.

[SCHNEIKER 1989] "Nano technology with Feynman Machines : Scanning Tunneling Engineering and Artificial Life", C. Schneiker, in [LANGTON 1989].

[WALTZ et al 1988] "Connectionist Models and their Implications : Readings from Cognitive Science", D. Waltz, J.A. Feldman, Ablex Publ. Co. New Jersey, 1988.

CART CENTERING AND BROOM BALANCING BY GENETICALLY BREEDING POPULATIONS OF CONTROL STRATEGY PROGRAMS

John R. Koza
Computer Science Department
Stanford University
Stanford, CA 94305
Koza@Polya.Stanford.Edu
415-941-0336
Martin A. Keane
Third Millennium Venture Capital Limited
5733 West Grover
Chicago, Illinois 60630
312-777-1524

Abstract: The paper describes a search for the time-optimal "bang bang" control strategy for the cart centering problem and a version of the broom balancing problem by genetically breeding populations of control strategy programs using a recently developed new genetic algorithm paradigm. The output of the new genetic algorithm paradigm comes in the form of a computer program composed of arithmetic operations, conditional logical operations, and mathematical functions which take the state variables of the problem as input and which produce commands specifying how to apply the "bang bang" force as output .

1.0 INTRODUCTION

The problems of centering a cart and balancing a broom by applying a "bang bang" force from either direction are well-known problems in control theory. The broom balancing problem has been studied extensively in connection with neural networks (Widrow 1963, Barto et. al. 1983, Widrow 1987, Anderson 1989). The cart centering problem has been previously studied in the genetic algorithm field in connection with Holland classifier systems (Goldberg 1983).

2.0 BACKGROUND DISCUSSION OF GENETIC ALGORITHMS

Genetic algorithms are mathematical algorithms that transform populations of individual mathematical objects (typically fixed-length binary character strings) into new populations using operations patterned after natural genetic operations such as sexual recombination (crossover) and fitness proportionate reproduction (Darwinian survival of the fittest). Genetic algorithms begin with an initial population of individuals (typically randomly generated) and then iteratively (1) evaluate the individuals in the population for fitness with respect to the problem environment and (2) perform genetic operations on various individuals in the population to produce a new population. Professor John Holland of the University of Michigan presented the pioneering formulation of genetic algorithms for fixed-length character strings in *Adaptation in Natural and Artificial Systems* (Holland 1975). Holland established, among other things, that genetic algorithms are a mathematically near optimal approach to adaptation when the adaptive process is viewed as a set of simultaneous multi-armed slot machine problems requiring an optimal allocation of future trials given the currently available information. Holland's basic genetic algorithm has been extended to variable length sets (Smith 1980) and further extended (Holland 1986) by embedding it into a cognitive architecture (called a classifier system) containing a population of fixed-length if-then rules. Recent work on genetic algorithms can be surveyed in Goldberg (1989), Davis (1987), and Schaffer (1989).

3.0 HIERARCHICAL GENETIC ALGORITHMS

Populations of computer programs can be genetically bred to solve problems in several different areas of artificial intelligence and symbolic processing (Koza 1989a, 1989b). In this recently developed new genetic algorithm paradigm, the individuals in the genetic population are computer programs rather than fixed-length character strings. In particular, the individuals in the population are LISP S-expressions (i.e., rooted point-labeled trees in the plane) created from compositions of functions and atoms appropriate to the particular problem domain. The set of functions used typically includes conditional logical operations, arithmetic operations, mathematical functions, and particular functions appropriate to the problem domain at hand. The set of atoms used typically includes various constants and particular inputs and sensor values appropriate to the problem domain. The search space is the hyper-space of all possible LISP S-expressions (i.e. computational procedures, computer programs) that can be recursively composed of the available functions and atoms. The crossover operation appropriate for mating two parents from this hyperspace of LISP S-expressions creates new offspring S-expressions by exchanging sub-trees (i.e. subroutines, sub-lists) between the two parents. The results of this paradigm are inherently hierarchical. This new genetic algorithm paradigm has been successfully applied to example problems in several different areas, including (1) sequence induction (e.g. inducing a recursive computational procedure for the Fibonacci sequence and various polynomial sequences), (2) automatic programming (e.g. discovering a computational procedure for solving pairs of linear equations, solving quadratic equations for complex roots, and discovering trigonometric identities), (3) machine learning of functions (e.g. learning the exclusive-or function, the parity function, and a Boolean multiplexer function previously studied as a test function in neural net, decision tree, and classifier system work), (4) planning (e.g. developing a robotic action sequence that can restack an arbitrary initial configuration of blocks into a specified order), (5) pattern recognition (e.g. translation-invariant recognition of a simple one-dimensional shape in a linear retina). In this new genetic algorithm paradigm, the sub-hyper-spaces of similar individual LISP S-expressions grow (and decay) in at the same near optimal rates as in string-based genetic algorithms if, and to the extent that, the similar features involved consist of small compact "building blocks" (i.e. sub-trees, sub-routines, sub-lists).

4.0 THE CART CENTERING PROBLEM

The cart centering problem involves a frictionless push cart with mass mc that can move on a one dimensional track. There are two state variables for this system, namely, position x and velocity v. A force of fixed magnitude F is to be applied to the center of mass of the push cart from either the left or right direction at each time step τ. The problem is to find a control strategy (i.e. a computational procedure, computer program, LISP S-expression) that applies these "bang bang" forces so that, after initial random starting conditions (i.e. an initial random position and an initial random velocity), the cart becomes centered (i.e. approximately arrives at position 0.0 with velocity 0.0) in minimal average time.

A version of the solution for the cart centering problem (Bryson 1975) involves applying the "bang bang" force F from the positive direction if

$$\frac{v^2}{2}\,\frac{\text{Sign } v}{|F/m|} < -x$$

and applying the force from the negative direction otherwise.

The set of functions available for the cart centering problem are multiplication (*), the absolute value function (ABS), and a greater-than function (GT). The greater-than function GT is a function of two arguments that returns +1 if the first argument is greater than the second argument and returns -1 otherwise. The set of atoms available for this problem are the current value of position (POS) and velocity (VEL) and the constant -1. When a particular control strategy (i.e. LISP S-expression) evaluates to any positive value at a particular time step, the force F is applied from the positive direction. Otherwise, the force is applied from the negative direction. The mass of the cart mc was chosen to be 2.0 kilogram and the force F was chosen to be 1.0 Newtons. These choices make the constant denominator 2 |F/m| equal to 1.0.

Thus, a parsimonious version of this optimal control strategy would be the LISP S-expression (GT (* -1 POS) (* VEL (ABS VEL))). The interpretation of this LISP S-expression is as follows: Starting with POS and VEL as inputs, take the absolute value of VEL and multiply it by VEL. Then, multiply POS by -1. Then, compare the first result and second result. If the first result is greater than the second result, apply the "bang bang" force from the positive direction. Otherwise, apply the "bang bang" force from the negative direction.

The environment in which adaptation takes place in this problem consists of a test suite of 20 initial random starting conditions for the two state variables of the system. Position is chosen randomly between -0.75 meters and +0.75 meters and velocity is chosen randomly between -0.75 meters/seconds and +0.75 meters/second.

The fitness of a control strategy (i.e. computational procedure) is determined by evaluating the strategy against all starting conditions in the test suite. If a particular control strategy brings the two state variables of the system (i.e. position x and velocity v) close to the desired zero-zero state (i.e. within a tolerance of 0.001 as measured by a norm equal to the sum of the squares of the two state variables), the fitness value of that control strategy is the time required (in seconds). If the control strategy fails to bring the two state variables close to the desired zero-zero state before the system times out , the control strategy is assigned a fitness value equal to that maximum time. Note that the time for the optimal strategy with worst starting condition in the above region is about 4.5 seconds and the time for the optimal strategy averages about 1.8 seconds over all the various starting conditions in the test suite. Time is discretized into time steps (e.g. 450 time steps of 0.02 seconds). The total time available before the system times out for a given strategy (i.e. 9.0 seconds) is thus about twice the worst time for the optimal strategy.

The process starts with the random generation of 300 random control strategies recursively composed from the available functions and atoms above. The initial population of random control strategies includes many highly unfit control strategies, including strategies that totally ignore the state variables, strategies that blindly apply the force in only one direction, strategies that are correct only for particular specific starting conditions, strategies that are totally counter-productive, and strategies that cause wild oscillations and meaningless gyrations. However, even in this highly unfit initial random population (whose fitness averages 8.91 seconds for the starting conditions in the test suite), some control strategies are somewhat better than others. The best control strategy for the initial random generation averages 2.98 seconds for each starting condition in the test suite. And, in the valley of the blind, the one-eyed man is king. The genetic crossover operation is then applied to parents selected with probabilities proportionate to fitness to breed a new population of offspring control strategies. Although the vast majority of the new offspring control strategies are again highly unfit, some of them tend to be somewhat more fit than others. Moreover, over a period of time and many generations, some of them tend to be slightly more fit than those existing in earlier generations.

At the 5th generation (i.e. after processing a total of 1500 individuals), the best individual in the population of control strategies was the strategy (GT (* (GT (* -1 POS) POS) (ABS POS)) (* (ABS VEL) VEL)).

The first argument of the outermost GT function (* (GT (* -1 POS) POS) (ABS POS)) reduces to -POS for any value of POS. Thus, this control strategy is equivalent to the known optimal solution. Note that this computer program can be viewed as defining the optimal control surface.

5.0 THE BROOM BALANCING PROBLEM

The broom balancing (inverted pendulum) problem involves the elements of the cart centering problem described above and a broom of mass mb attached to the top of the cart with a pivot so that the broom can pivot (in one plane). The broom has an initial angle θ (measured from the vertical) and an initial angular velocity ω. The distance from the center of mass of the broom to the pivot point on the cart is λ. There are four state variables of this system, namely, position x, velocity v, angle θ, and angular velocity ω. The constants of the system are m_c=1.0 kilogram, m_b=0.1 kilogram, gravity g=9.8 meters/sec^2, time-step τ =0.02 seconds, length λ =7.888 meters, and force F=10.780 Newtons. The problem is to find a control strategy that causes the broom to become balanced (i.e. approximately attain an angle 0.0 with angular velocity 0.0). Note that in the version of the problem considered here, we do not require the cart to be centered when these values of θ and ω are attained.

The set of functions available for the broom balancing problem are addition (+), subtraction (-), multiplication (*), the sign function (SIG), the square root of absolute value function (SRT), and the greater-than function (GT). The SIG function returns +1 if its argument is positive and -1 otherwise. The set of atoms available for this problem are the current value of angle (ANG) and angular velocity (AVL) and the constant +1. The optimal control strategy has been computed by Keane as applying the positive force when

$$\theta > \text{Sign}\ \omega\left[1-\sqrt{1+\omega^2}\right]$$

and, otherwise, the negative force.

This can be expressed as a LISP S-expression as (GT ANG (* (SIG AVL) (- 1 (SRT (+ 1 (* AVL AVL)))))).

The environment for the broom balancing problem here consists of a test suite of 12 initial starting conditions. Angle (ANG) is chosen between -0.25 radians (-14.3 degrees) and +0.25 radians and and angular velocity (AVL) is chosen between -0.25 and +0.25 radians/second. The norm used to compute attainment of the desired final state is the sum of the squares of the angle ANG and the angular velocity AVL with the tolerance of 0.001. Note that the time for the optimal control strategy with worst starting conditions in the above region is about 1.5 seconds and the average time for the optimal strategy is about 0.74 seconds. Time is discretized into time steps (e.g. 150 time steps of .02 seconds). The total time available (i.e., 3 seconds) is thus about twice the worst time for the optimal strategy.

Population size was 300. The time consumed by an initial random strategy in the initial random generation averages 2.9 seconds over the starting conditions in the test suite (i.e. about 4 times the average time of 0.74 seconds consumed by the optimal control strategy).

In the process of searching for the optimal control strategy for this problem, the fourth best solution we have found was a simple linear hyper-plane through the origin ($\omega + 3\theta$) that performed in an average of 0.899 seconds (about 21% worse than the optimum). The third best control strategy we have found was the non-linear control strategy w + $3\theta + 2\theta^3 - 3\theta^2 - \theta\omega$ that performed in an average of 0.897 seconds (about 21% worse than the optimum).

The best two control strategies we have found were:

the non-linear control strategy(+ (* (* (* (SRT AVL) AVL) (SRT AVL)) (SRT AVL)) ANG) and

the control strategy (+ (* (+ (* (* (SRT 1) AVL) (SRT AVL)) (* (SRT 1) ANG)) (SRT AVL)) (* (SRT 1) ANG)).

Note that the former control strategy is equivalent to the control strategy $\theta + \omega\ \omega^{3/2}$, while the latter control strategy is equivalent to $\theta + \omega^2(\text{SIG}\ \omega) + \theta\ (\text{SRT}\ \omega)$.

These individuals appeared in the 4th generation of a run (i.e. after processing 1200 individuals). The performance of these two individuals was identical in the actual run and virtually indistinguishable after additional separate testing. These two control strategies each performed in an average of approximately 0.745 seconds (which is within less than 1% of the optimum).

REFERENCES

Anderson, Charles W. Learning to control and inverted pendulum using neural networks. *IEEE Control Systems Magazine*. 9(3). Pages 31-37. April 1989.

Bryson, Arthur E. *Applied Optimal Control*. New York: John Wiley 1975.

Davis, Lawrence (editor) *Genetic Algorithms and Simulated Annealing* London: Pittman 1987.

Goldberg, David E. *Computer-Aided Gas Pipeline Operation Using Genetic Algorithms and Rule Learning*. PhD dissertation. Ann Arbor: University of Michigan. 1983.

Goldberg, David E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley 1989.

Holland, John H. *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI: University of Michigan Press 1975.

Holland, John H. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In Michalski, Ryszard S., Carbonell, Jaime G. and Mitchell, Tom M. *Machine Learning: An Artificial Intelligence Approach*, Volume II. P. 593-623. Los Altos, CA: Morgan Kaufman 1986.

Koza, John R. Hierarchical genetic algorithms operating on populations of computer programs. *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI)*. San Mateo, CA: Morgan Kaufman 1989.

Koza, John R. Genetic computing using hierarchical genetic algorithms. *Machine Learning* journal. In press. 1989.

Schaffer, J. D. (editor) *Proceedings of the Third International Conference on Genetic Algorithms (ICGA)*. San Mateo, CA: Morgan Kaufmann Publishers Inc. 1989.

Smith, Steven F. *A Learning System Based on Genetic Adaptive Algorithms*. PhD dissertation. Pittsburgh: University of Pittsburgh 1980.

Widrow, Bernard. Pattern recognizing control systems. *Computer and Information Sciences (COINS) Symposium Proceedings*. Washington, DC: Spartan Books, 1963.

Widrow, Bernard. The original adaptive neural net broom balancer. *1987 IEEE International Symposium on Circuits and Systems*. Vol. 2.

Preadaptation in neural circuits

David G. Stork, Scott Walker, Mark Burns and Bernie Jackson

Departments of Psychology and
Electrical Engineering
Jordan Hall
Stanford University
Stanford, CA 94305

Abstract

Motivated by a recent analysis of the tailflip circuitry of the crayfish,[1] we simulated the processes of evolution, development and learning in a simple 7-neuron circuit, in order to assess the roles of these processes in preadaptation. Preadaptation occurs when a circuit (or organ or behavior, etc.) which evolved to solve one behavioral/functional problem is later appropriated and used for a *different* problem; such a process can lead to the final circuit being "non-optimal."[2]

Using limited but biologically motivated genetic algorithms, learning mechanisms and neural equations, we programmed a Connection Machine[3] CM-2 to simulate a population of individuals, each with a genome that leads through development to a neural circuit. Darwinian fitness depended on the circuit's performance on a simple behavioral task. Our two tasks were: in the event of external stimulation of a (pressure-sensitive) sensory neuron 1) *large* activity in the motor neuron and 2) *small* activity in the motor neuron. The fitness of networks in population **A** was based first solely on task 1 ("swimming") and at a later epoch solely on task 2 ("tail-flipping"). The fittest individuals in population **A** at the end of the evolutionary scenario differed from those in population **B**, whose survival was based solely based on task 2. Moreover, neural structures were found in population **A** which might not have been simply predicted based on an analysis of the final task alone.

This last result lends support to the notion that "certain features of nervous systems may not have functional significance" and "... there is no reason why the simplest solution to a problem should be the one actually used by the nervous system. As long as both the end result and all the intervening stages work, elegance of design counts for little."[1,4,5] Such a viewpoint can have profound implications for simulation studies of neurobiological networks, and for neurobiology more generally.

Introduction

Whereas the vast majority of neural network research centers on learning or self-organization based on input patterns (and possibly teaching signals), neurobiological circuits (including most in lower species) posses high structure even at birth -- before learning takes place.[6] This innate structure may profoundly affect what an individual can and cannot learn, how fast, etc., and is due to evolutionary and development processes. Neural network simulation research has almost completely ignored the role of phylogeny in such *nature* aspects in the *nature/nurture* debate.

One phylogenetic process has particular import for the analysis of innate structure in biology: *preadaptation* (and its closely related process *exaptation*).[7] Preadaptation in neurobiology describes the phenomenon whereby a circuit which evolved to solve one task is later appropriated and used for a *different* task. In order to understand the evolutionary sources of structure in simple networks, then, we simulated the process of preadaptation.

Methods

All simulations were performed on a Connection Machine CM-2 (using C*, the parallel extension of C), since the fine-grained, massively parallel nature of our problem precluded realistic simulations on all but the most powerful of serial machines. Populations consisted of 60 individuals (each containing 7 neurons) running in parallel; the component neurons ran in parallel as well.

- **Genome representation and genetic algorithm**

The model genome consisted of sequences of promoter (or controller) genes, as well as structural genes. The structural genes represent 1) markers indicating whether a neuron is a sensory, motor, or interneuron, 2) cell surface molecules (used for development and initial network connectivity),[8] 3) neurotransmitter type, 4) synapse receptor types, 5) cell channel densities (used for gain and saturation points in network behavior, constants a-e, in eq. 1 below), and other functional properties of the network. Promoter genes activate or de-activate distributed *patterns* of structural genes, which might include correlations between such aspects as cell type and neurotransmitter. At the end of any generation, there is fitness proportional reproduction, and the genomes of surviving individuals (see "Selection," below) undergo single gene mutation (p = 0.02/bit/generation), crossover (75%), or direct duplication[9] to yield the genomes of the offspring.

- **Development**

Initial neural connectivity and synapse strength was a function of the similarity of neural cell surface molecular markers,[10] as encoded in the genes. We did not include activity-dependent development.

- **Behavior**

The equations for each neuron are based on the work of Hodgkin and Huxley:[11]

$$\frac{dx_i}{dt} = -ax_i + (b - cx_i)\left\{\sum_{j\in G_{ex}} z_{ij}f(x_j) + I_i\right\} + (d - ex_i)\left\{\sum_{j\in G_{in}} z_{ij}f(x_j)\right\} \qquad \text{[eq. 1]}$$

where:

x_i:	activity in neuron i (depolarization -- mV)
$f(x_j)$:	output spike rate (spikes/second) -- compressively nonlinear
a, b, c, d, e:	constants: describe ion concentrations, channel densities, etc.
z_{ij}:	synaptic strength (mV/spikes/sec)
α:	decay constant for synapse (mV/spikes/sec^2)
$G_{ex(in)}$:	set of neurons connected via excitatory (inhibitory) synapses to neuron i

- **Learning**

In those simulations in which learning was permitted to occur, synapses were modified according to a pseudo-Hebbian rule:

$$\frac{dz_{ij}}{dt} = -\alpha z_{ij} + (x_i - \Gamma_{ij})f(x_j)$$

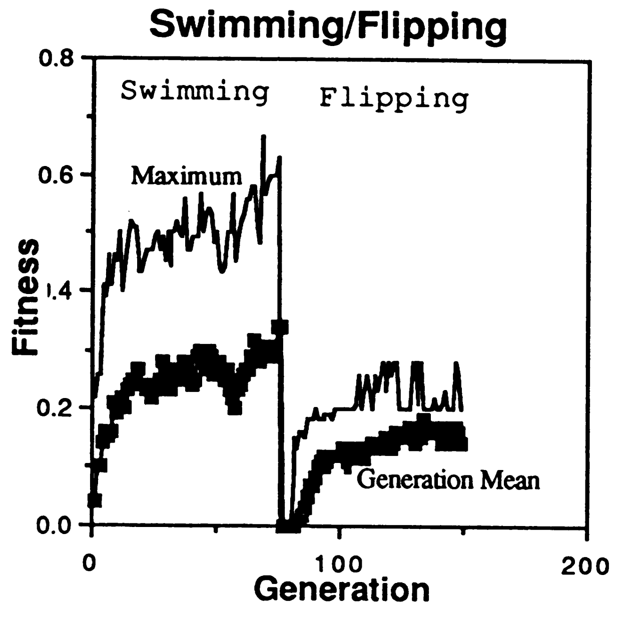
where Γ_{ij} is a threshold value.

- **Selection**

For each network, the sensory neuron was stimulated briefly and the resulting activity in the motor neuron was monitored over a sufficient time that the network relaxed to a (nearly) quiescent state. The maximum of the instantaneous output of the motor neuron was an index to the fitness: for task 1 (swimming), a *large* such maximum conveyed fitness, whereas for task 2, a *small* such maximum conveyed fitness. Fitness-proportional stochastic selection was used to determine which individuals would participate in the production of the subsequent generation.

Results

The fitness function for the two populations of 60 individuals over 150 generations are shown in Figure 1. Population A was "preadapted" -- first selected on task 1 (swimming), and later on task 2 (flipping), whereas population B was selected solely on the basis of flipping. The sharp drop at generation 76 in population A reflects the fact that swimmers were initially ill-fit for flipping. Most importantly, the structure of the networks differed between populations. Figure 2 shows the most fit circuits after evolution in the preadapted population **A** (left) and the normal population **B** (right). We also found circuits in population A that contained structures (in particular "useless synapses") that did not appear in the fittest individuals of population **B**.



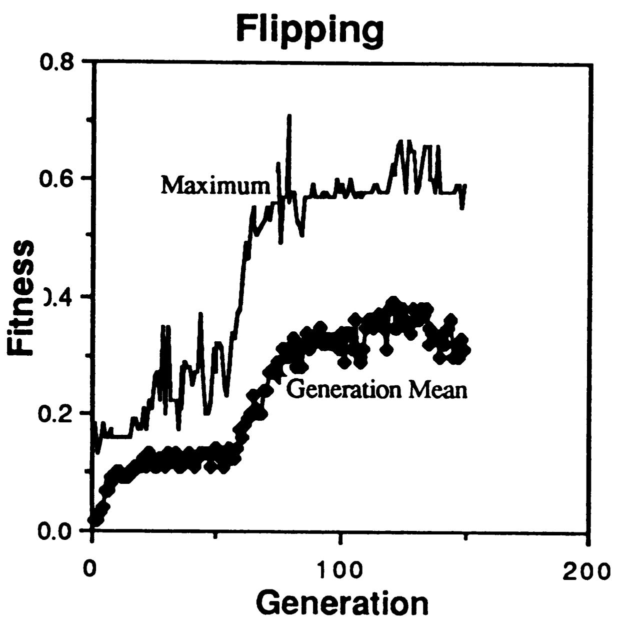


Figure 1: (*left*) The maximum individual fitness and the generation mean fitness for population **A**, selected first for swimming and then (after generation 75) for flipping. (***right***) Maximum and mean fitness for population **B**, selected solely on flipping. Each generation had 60 individuals throughout.

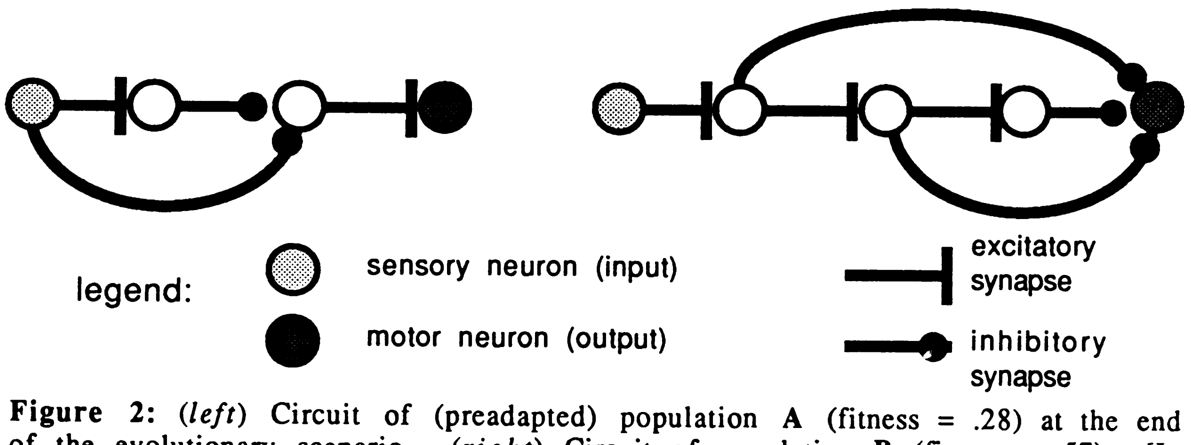


Figure 2: (*left*) Circuit of (preadapted) population A (fitness = .28) at the end of the evolutionary scenario. (***right***) Circuit of population B (fitness = .57). [In both circuits, connections that are negligibly small or neurons with no effect on the output are not shown.]

Conclusions

Our preliminary results give us confidence that simulation techniques can shed further light on evolutionary effects in neural networks. Although our simulations lend support to the crayfish scenario postulated by Dumont and Robertson,[1] only with further refinements -- more realistic transmitter-receptor data, genetic representations, activity-related development, etc. -- will we be able to rule out a competing hypothesis based on developmental constraints.

The existence of preadaptation undermines the biological relevance of the neural network approach of analyzing just inputs and outputs and network "design constraints" since historical factors can preclude "elegant" biological implementations of computational problems.[1] These historical effects may be especially relevant in particularly complex neural structures responding to astoundlingly complex environmental pressures, as well as evolutionarily late processes such as language, which may have appropriated structures "preadapted" to functions *other* than speech or communication.[12]

Acknowledgements

We gratefully acknowledge Connection Machine resources provided through both DARPA grant DACA 76-88-C-0012 (administered by Thinking Machines Corp., Cambridge, MA) and through RIACS, Moffett Field, CA. Thanks to Jeffrey Wine for explanations on crayfish tail circuitry and to David Rumelhart for support and methodological insights.

References

1 James P. C. Dumont and R. M. Robertson, "Neuronal Circuits: An Evolutionary Perspective," *Science* **233**, 849-853 (1986).

2 Stephen Jay Gould, "Darwinism and the expansion of evolutionary theory," *Science* **216**, 380-387 (1982).

3 W. Daniel Hillis, ***The Connection Machine***, MIT Press (1985).

4 David G. Stork, "Preadaptation and evolutionary considerations in neurobiology," ***Learning and Recognition -- A Modern Approach*** (Y.C. Lee, ed.) World Scientific Publishing (1989).

5 David G. Stork, "Exaptation, preadaptation, evolution and biological neural networks," *Synapse Connection* **1**(3), 2-5 (1987).

6 David G. Stork, review of ***Parallel Distributed Processing*** by D. E. Rumelhart and J. L. McClelland *Bulletin of Mathematical Biology* **50**, 202-207 (1988).

7 Stephen Jay Gould & E. S. Vrba, "Exaptation -- a missing term in the science of form," *Paleobiology* **8**, 4-15 (1982).

8 Gerald M. Edelman, ***Topobiology***, Chapter 7, Basic Books (1989).

9 John H. Holland, ***Adaptation in Natural and Artificial Systems*** Michigan (1975).

10 Gerald M. Edelman, ***Topobiology***, Chapter 7, Basic Books (1989).

11 Stephen Grossberg, ***Studies in Mind and Brain***, Chapter 4, Boston: Reidel (1982).

12 David G. Stork, "Sources of structure in neural networks for speech and language," in ***Progress in Connectionism*** (J. Elman and D. Rumelhart, eds.) (1989, in preparation)

Optimizing Small Neural Networks Using a Distributed Genetic Algorithm

Darrell Whitley and Timothy Starkweather
Computer Science Department
Colorado State University
Fort Collins, Colorado 80523
whitley@CS.Colostate.edu

1. Introduction

Our experiments dealing with neural network optimization using standard genetic algorithms indicate that these systems, while adequate to easily solve Xor, fail to yield sufficiently accurate optimization on other relatively small neural network problems. However, we have successfully optimized several of these same neural nets using *GENITOR*, a genetic algorithm that differs in fundamental ways from standard genetic algorithms. This algorithm employs one-at-a-time reproduction and allocates reproductive opportunities according to rank to achieve an appropriate selective pressure.

Recently, we have found that a distributed genetic algorithm can be used to dramatically speed up genetic search while at the same time increasing the accuracy of the solutions and the consistency of the genetic algorithm at finding high quality solutions. *GENITOR II* is a parallel version of *GENITOR* which uses many smaller distributed populations in place of a single large population. One of the key benefits of a parallel genetic algorithm is that it allows one to aggressively exploit information within a subpopulation while not totally exhausting the genetic diversity of the entire population. Each subpopulation will, due to inherent sampling biases, exploit its genetic material in a slightly different way and explore the search space differently. Thus, by occasionally swapping individuals between the subpopulations two complementary effects are achieved. First, the effects of genetic drift in the various subpopulations are countered. Second, the variability that results globally between the subpopulations is actually used at a local level as a source for new, yet high quality genetic material that allows diversity to be sustained in a way that constructively contributes to the search process.

2. Foundations of Genetic Algorithms

Genetic algorithms, in their simplest form, are a global optimization method capable of robust combinatorial optimization. The theoretical foundations of genetic algorithms have been built on the idea that an optimization problem can be encoded as a list of concatenated parameters that has a binary (or sometimes symbolic) encoding.

To understand the theory behind genetic algorithms one must first understand how recombination takes place. Consider the following binary string: 1101001100101101. In general, the string might represent a possible solution to any problem that can be parameterized. This string could represent, for example, a simple neural net with 4 links, where each connection weight is represented by 4 bits. The goal of genetic recombination is to find a set of parameters that yields an optimal or near optimal solution to the problem. Recombination requires two parents. Consider the string 1101001100101101 and another binary string which we represent using x and y to represent 0 and 1. Let the second string be yxyyxyxxyyyxyxxy. Using two "break-points," recombination occurs as follows:

— 11010 ╳ 01100101 ╳ 101 —
— yxyyx ╳ yxxyyyxy ╳ xxy —

Swapping the fragments between the two parents produces the offspring 11010yxxyyyxy101 and yxyyx01100101xxy. Booker's (1987) two point reduced surrogate operator was used in the experiments reported here.

A population of random binary strings representing the encoded parameters is first generated. The strings, or "genotypes," are then evaluated to obtain a quantitative measure of how well they perform as possible problem solutions. Next, reproductive opportunities are allocated such that the best strings receive more opportunities to reproduce than those which have poor performance; this bias need not be great to produce the required selective pressure to allow "artificial selection" to occur. To understand how recombination on binary strings can be related to hyperspace, consider a "genotype" that is encoded with just 3 bits. With 3 bits we have a three dimensional problem and can illustrate the distribution of possible points with a simple cube. The front plane or face of the cube could, for example, be defined as "all points that begin with 0." If * is used as a "don't care" or wild card match symbol, then this plane can also be represented by the schemata, or similarity template, 0**.

All bit strings that match a particular schemata lie in its hyperplane. In general, every binary encoding corresponds to a corner in the hypercube and is a member of 2^L-1 different hyperplanes, where L is the length of the binary encoding. For example, the string 011 not only samples its own corner in the hypercube (011) but also the hyperplanes represented by the schemata 0**, *1*, **1, 01*, 0*1, *11, and ***. This last schema, ***, corresponds to the entire search space and thus provides baseline information. In terms of the sample population, however, it represents the current mean performance, which is monotonically increasing in the *GENITOR* implementation.

This idea, that one point samples numerous hyperplanes, is referred to as "intrinsic parallelism" because many hyperplanes are sampled when one bit string is evaluated. The search power of a genetic algorithm lies in the fact that selective pressure causes hyperplanes with above average performance to gradually increase their representation in the population. Recombination reshuffles this hyperplane information so that new corners in the hypercube can be sampled; unfortunately recombination also introduces a slight bias. Hyperplanes represented by bits that are physically distance on the encoding are more likely to be disrupted during recombination than hyperplanes represented by bits that are physically close on the encoding. Thus, some hyperplanes do not increase their representation in the population unless they are somewhat above average. Nevertheless, recombination is largely "information preserving" as opposed to a process such as mutation which destroys information (Holland 1975).

A standard genetic algorithm and *GENITOR* have previously been compared on a standard set of test functions (Whitley and Kauth 1988) with *GENITOR* performing slightly better on small problems and significantly better on larger problems such as optimizing the neural nets described here (Whitley and Hanson 1989). In a standard genetic algorithm the parents are replaced by their offspring after recombination. The entire population undergoes recombination in a single generation with offspring displacing parents. The idea is that the genetic material of the parents will largely survive and remain in the population, although the parents themselves do not. In the *GENITOR* algorithm two parents are first selected from the population. Copies of the parents are then recombined, producing two offspring. One of the offspring is randomly discarded, the other is then allowed to replace the lowest ranking string in the population--the offspring does not replace a parent string. In this way the algorithm is much more likely to accumulate schemata associated with high performance hyperplanes, especially "long" schemata represented by bits that are physically separated on the encoding. This new genotype is then ranked according to its performance relative to the remainder of the population, so that it may now compete for reproductive opportunities. Note that this means that fragments, or schemata, which are commonly found in strings with above average performance will tend to increase in number in the population. A theorem has been developed which suggests that this approach does as well or better than a standard genetic algorithm at finding and preserving good genetic "building blocks" in the form of schemata (Whitley and Kauth 1988).

GENITOR is also designed to allocate reproductive trials to individuals (ie: an individual problem encoding or "genotype") based on their rank in the population rather than directly using the value returned by the evaluation function to calculate fitness. When the value from the evaluation function is used directly, "selective pressure" can fluctuate so that the algorithm converges too quickly, or the search may stagnate because the selective pressure is inadequate. *GENITOR* abandons fitness values which are directly proportional to performance and instead uses the rank of the "genotype" in the current population to assign a fitness value. Allocating reproductive trials in this way ensures that a constant and effective selective pressure can be maintained no matter how performance is calculated.

3. Distributed Genetic Search

Genetic diversity is a key ingredient of genetic search. More diversity in the population translates into more hyperplane information to drive the search. However, by increasing the representation of high performance hyperplanes in the population, diversity is lost. Thus, exploitation reduces the potential for further exploration. One way to obtain and sustain genetic diversity is to use larger population sizes. Empirical tests show that this has the desired effects, except that it is usually necessary to double population size for an incremental increase in performance. Unfortunately search time (as measured in number of recombinations) tends to double as well (Whitley and Hanson 1989). The theoretical importance of genetic diversity has led us to explore other methods of sustaining genetic diversity without resorting to excessively large populations.

A distributed genetic algorithm can actually enhance search speed and accuracy without the negative impact on search time produced by simply increasing population size. A distributed genetic algorithm allows information to be aggressively exploited within a subpopulation without totally exhausting the genetic diversity of the entire population. Each subpopulation will, due to inherent sampling biases, exploit its genetic material in a slightly different way and converge to a slightly different, but similiarly competitive solution. We have shown this to be true on a sample "ugly deceptive" problem developed by Goldberg et al. (1989) which is known to be difficult for a genetic algorithm because of deceptive hyperplanes in the search space. The problem has a single correct solution and Goldberg reports that a standard genetic algorithm fails to solve this problem. When running *GENITOR* in a non-distributed mode, the algorithm typically gets approximately 30% of the bit-pattern correct; increasing the population size has almost no effect on this problem (perhaps because of the deceptive hyperplanes.) However, on multiple runs, *GENITOR* finds a different portion of the correct pattern. Again, this seems to be the effect of different sampling biases in the initial population and subsequent allocation of reproductive opportunities that induce different genetic drift patterns. However, when run in parallel, *GENITOR* is able to take the various "distributed solutions" and, through recombination, construct a complete solution to this difficult and deceptive problem.

By occassional swapping individuals new yet high quality genetic material can be introduced into the evolving subpopulations. While these are only rough guidelines, we have found that with a subpopulation of size X, swapping after 5X recombinations (in each subpopulation) works well. When a swap occurs, each subpopulation sends its best to one of its neighbors; in our implementation, a round-robin swap is used. Swapping should not occur too often or the effect will be similar to simply having a single population. On the other hand, swapping must occur often enough to prevent the subpopulations from converging toward incompatible solutions. This is particularly true in a problem such as neural network optimization; there are typically multiple ways in which to find an appropriate set of weighted connections and recombining disparate solutions may result in a disfunctional net.

4. Optimizing Neural Networks: Results

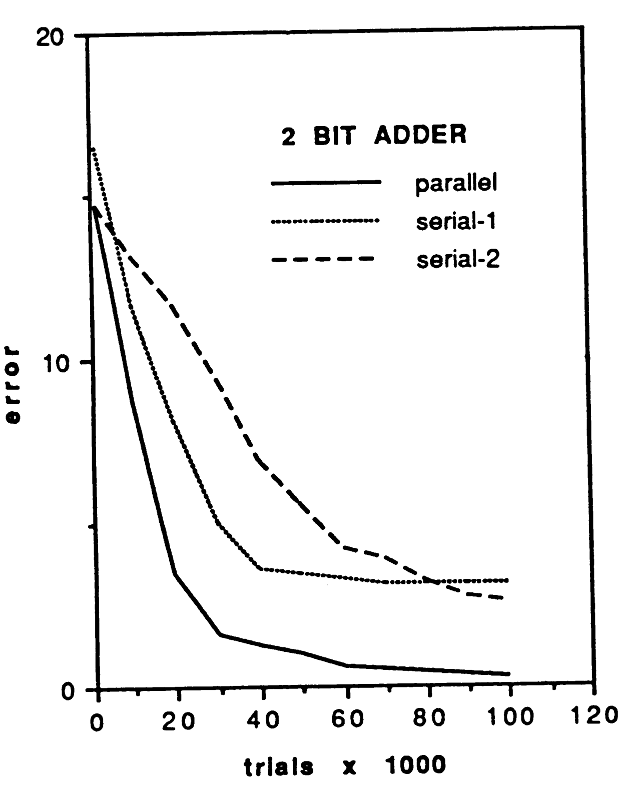
The optimization problems posed by neural networks provide a new and challenging test for genetic algorithms. Previously, *GENITOR* has been used to optimize the connection weights in four neural net problems; the reader may refer to (Whitley and Hanson 1989) for more specific details on how to encode the neural net optimization problem for a genetic algorithm. The test problems are 1) the exclusive-or (Xor) problem, 2) a 424-encoder, and 3) two versions of an adder problem. The adder problem involves the addition of two two-bit numbers. One version of the adder problem uses minimal connections and "reliably leads the system [using back propagation] into a local minimum" (Rumelhart et al. 1986:343). *GENITOR* easily and reliably solved the minimally connected adder, but had difficulty with the fully connected adder (Whitley and Hanson 1989). In general, it seems that the genetic algorithm has difficulty with problems whose encoding grows too large (over 250 bits). This is consistent with the biases known to exist in the genetic algorithm. As noted already, hyperplanes represented by bits that are highly distributed over a long encoding are inadequately sampled by the genetic algorithm because of a higher rate of disruption during recombination.

The graphs in Figure 1 show new parallel results for the 424 encoder and the fully connected adder using 10 subpopulations; the behavior displayed here is typical of that obtained on other problems. On numerous experiments the distributed algorithm has found solutions which we have not been able to duplicate with the serial version of *GENITOR*, despite the fact that we have spend far more time, effort and computer time developing and testing the serial version of *GENITOR*. In serial runs, we have also used population sizes on both problems that are up to 5 times larger than the population sizes used in the distributed genetic algorithms reported here with only very modest improvements over the smaller population sizes after much longer search times.

5. Other Problems, Future Work

We have also shown that this approach works on a number of other difficult optimization problems. On a 30 city, 50 city, 75 city and 105 city Traveling Salesman Problem, the distributed genetic algorithm consistently found new best known solutions (See Whitley, Starkweather and Fuquay 1989, for non-parallel results on these problems). As noted, the parallel algorithm also easily finds the optimal solution to a problem that is is difficult for genetic algorithms to solve because of deceptive hyperplane information.

Recently we have used genetic algorithms to define the connectivity of neural nets with positive results. The *GENITOR* algorithm has not only found smaller nets, but also nets that learn much faster and much more consistently that fully connected nets. (Whitley and Bogart 1989; cf. Miller 1989.)



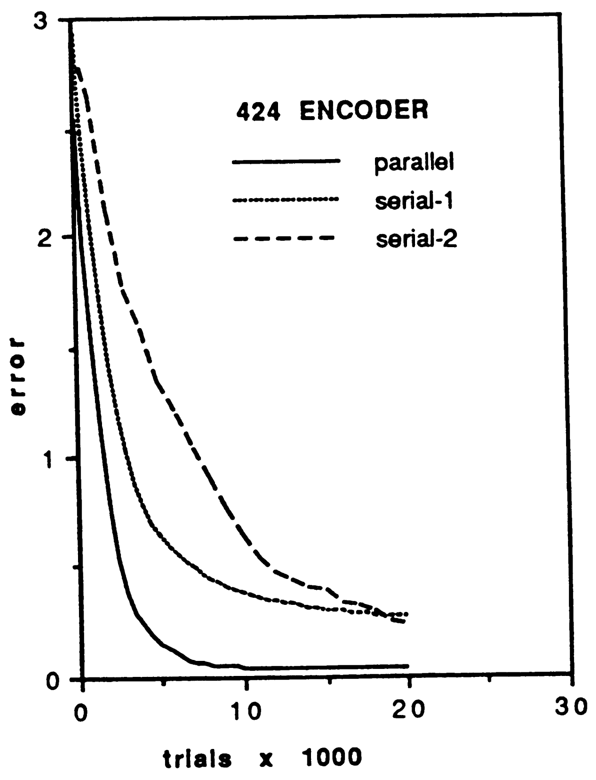


FIGURE 1. Graphs for parallel and serial runs of the genetic algorithm. On the adder problem, the parallel and serial-1 runs used a population size of 1000; serial-2 shows the effects of increasing the population size to 2000. On the 424 encoder problem, the parallel and serial-1 runs used a population size of 200, while serial-2 used a population of 500. The sample size for the 424 encoder was 20 runs each. The sample size for the adder was 10 runs for the parallel, 5 runs each for the serials.

ACKNOWLEDGMENTS

This research was supported in part by a grant from the Colorado Institute of Artificial Intelligence (CIAI). CIAI is sponsored in part by the Colorado Advanced Technology Institute (CATI), an agency of the State of Colorado. CATI promotes advanced technology education and research at universities in Colorado for the purpose of economic development.

REFERENCES

Booker, L. (1987) Improving search in genetic algorithms, in: Lawrence Davis (Ed.), *Genetic Algorithms and Simulated Annealing*. Morgan Kauffmann.

Goldberg, D., Korb, B. and Keb, K. (1989) Messy Genetic Algorithms: Motivation, Analysis, and First Results. TCGA Report 89003, University of Alabama.

Holland, J. (1975) *Adaptation in Natural and Artificial Systems*. Univ. of Michigan Press, Ann Arbor.

Rumelhart, D., Hinton, G. and Williams, R. (1986) Learning Internal Representations by Error Propagation. *Parallel Distributed Processing, Vol I* Cambridge, MA: MIT Press.

Miller, G., Todd, P. and Hegde, S. (1989) Designing Neural Networks using Genetic Algorithms. *Proceeding of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann.

Whitley D., and Kauth J. (1988) GENITOR: a different genetic algorithm. *Proceeding of the Rocky Mountain Conference on Artificial Intelligence*, Denver, CO.

Whitley, D. and Bogart, C. (1989) The Evolution of Connectivity: Pruning Neural Networks Using Genetic Algorithms. Tech Report, Colorado State University.

Whitley, D. and Hanson, T. (1989) Optimizing Neural Networks using Faster, More Accurate Genetic Search. *Proceeding of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann.

Whitley, D., Starkweather, T. and Fuquay, D. (1989) Scheduling Problems and Traveling Salesmen: The Genetic Edge Recombination Operator. *Proceeding of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann.

Using Verbs and Remembering the Order of Events

Robert B. Allen
Bellcore, Morristown, NJ
rba@bellcore.com

The *connectionist language users* (CLUES) paradigm is applied to several relatively complex tasks involving actions and events in a microworld. In the first study, a network is trained to observe an object moving, and to describe the action in the past tense if it has been completed. of a verb. In the second study, a network is trained to recognize actions and descriptions of those actions which involve the transfer of possession of an object. In the final study, a network maintains order information about events and answers questions about their relative order. Overall, these studies show that the CLUES model may be readily adapted to even fairly complex tasks involving the perception and labeling of events.

1. The CLUES Model

The addition of a memory to a back-propagation network [1] allows processing of temporal patterns. Allen [2, 3, 4] has proposed that temporal networks be used as intelligent agents. A particularly effective application of this approach is the use of the intelligent agents as *connectionist language users* (CLUES). The CLUES model has been employed in tasks such as generation of descriptions of objects and answering questions about objects.

In many cases, the CLUES model includes a microworld to which the language can refer. Allen and Kaufman [5] applied CLUES to processing events which occur in the microworld. In particular, they report: (a) the ability to answer questions which include verbs of possession; (b) labeling simple motions of objects with action verbs; and (c), discriminating the order of two events which occur in the microworld. The research in this paper extends the earlier work with three additional studies on labeling actions and events.

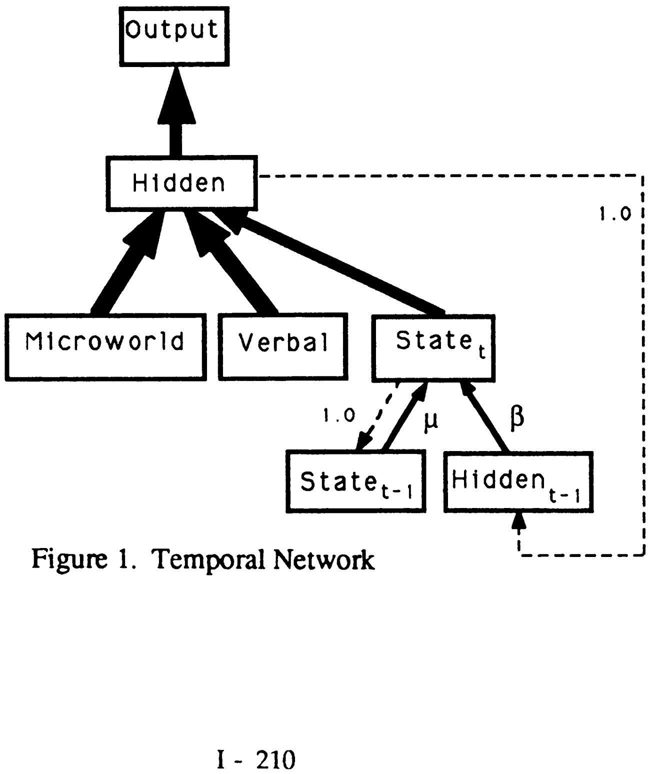


Figure 1. Temporal Network

2. Overview of Procedures

As shown in Figure 1, the networks include the usual 3-layer back-propagation architecture, along with a layer of units which maintain state information. During error correction all of the weights shown in bold, including those between the hidden layer and the state layer, are updated. The other weights are fixed with $\beta=1.0$, and $\mu=0.5$. Three-level (-1, 0, 1) encoding was used for the inputs such that when an object was absent (Sections 4 and 5), nulls were presented. The output codes were 0/1. Network parameters were set at $\eta=0.01$, $\alpha=0.9$.

3. Learning when to Make a Verb Past Tense

One of the greatest controversies in the connectionist literature concerns the construction of the past tense of verbs given an encoding of their present tense [6, 7]. However another seemingly more natural approach to the construction of the past tense follows from the CLUES model. Thus, the effort here was to develop a network which learned to generate a past tense marker when the observed action has been completed. Essentially, this past tense model requires the network to begin learning a rule for the construction of regular endings.

A corpus was developed in which objects could move left or right or stay where they were. In addition, objects which move during the first time step may stop moving after the second cycle, in which case the movement is described as happening in the past. In the example shown in Table 1, obj_A moved to the right and then stopped while obj_D stayed still. Because the probe asked about the object on the left (obj_A), the correct response to the probe is that obj_A moved right in the past. [5] found that actions were more accurately labeled by a network when the 'background' was moving than when the object moved across slots in the microworld. That approach was adopted here; however, to generate enough examples, two objects were present in different parts of the microworld and the network was required to answer about one or the other of them. A key aspect of the recognition task is the use of *don't care cycles* [2] during which no error corrections were made. In Table 1 these are shown as *s in the response column. There were 8 objects and 6 slots. There were 1506 training patterns of which 8 were saved for the transfer tests. A (12/1)-30*-7 network was trained for 800K patterns. On the 8 transfer questions, the network yielded 4 bit errors, with only one error on the tense bit.

Table 1. *Example of Past Tense Generation Task and Coding*

cycle	microworld		verbal input	response
1	obj_A $slot_3$	obj_D $slot_5$	*	*
2	obj_A $slot_4$	obj_D $slot_5$	*	*
3	obj_A $slot_4$	obj_D $slot_5$	*	*
4	*	*	left	obj_A right past
1	1 1 -1 -1 1 1	1 1 1 1 1 -1	*	*
2	1 1 -1 -1 -1 -1	1 1 1 1 1 -1	*	*
3	1 1 -1 -1 -1 -1	1 1 1 1 1 -1	*	*
4	*	*	-1	0 0 1 1 1 0 1

4. Transfer of Possession

Another complex class of verbs are those which require both a direct and indirect object. In [5], verbs which describe possession were considered. It is also possible to model verbs which describe the transfer of possession between two objects in the perceptual field. There were 8 major objects (which possessed the other objects) and 4 minor objects. The major objects were coded in 3 bits each and the minor objects in 2 bits. When the minor object was absent it was coded with nulls. There were three possible actions (*gave*, *received*, and *kept*). Overall there were 15 verbal input terms each requiring 4 bit coding. Because the verbal input was up to 4 words long, the entire input field was 16 bits. The output, however, required only 1 bit. Three types of negative questions were employed: (a) when the other major object engaged in the action; (b) when some other minor object was being asked about; and (c), when the wrong action is described. Thus, to keep equal numbers of positive and negative questions only 1/3 of the negative questions were used. Altogether there were 2509 patterns and 15 of these were reserved for the transfer test. A (10/16)-50*-1 network was trained for 900K patterns. There were errors on 2 of the transfer questions.

Table 2. *Example of Transfer of Possession Task*

cycle	microworld		verbal input	output
1	o_1m_1–	o_2–	*	*
2	o_1–	o_2m_1	*	*
3	*	*	o_1 gave m_1 o_2	yes

5. Remembering Temporal Events

Remembering and reasoning about the order of events is important for both language processing and planning [8]. In [5] networks were shown to be able to learn which of two events came first. However, processing of temporal events often requires determining the relative order of several events. In this section events are presented in four different time periods. On a fifth cycle the network is probed with pairs of events and required to report which came first. In this study, one pair of time periods was not included in the training set. Thus, in addition to the more complex pattern of events, this study also examines whether the network can generalize to comparing pairs of intervals that were not trained.

Table 3. *Example of Event Order Discrimination Task*

cycle	microworld	probe	output
1	obj_B	*	*
2	obj_C	*	*
3	obj_A	*	*
4	obj_D	*	*
5	*	DC	after

There were 6 events or objects (A-F) which were each coded in 3 bits. The probe employed a second set of 3-bit codes to identify the events. A typical training sequence is shown in Table 3. Because obj_D appeared in the microworld after obj_C, the correct response is "after". A total of 4320 patterns were prepared; however, all 720 probes involving the relative order of events at t_2 and t_3 were withheld for transfer tests. A (3/6)-60*-1 network was trained for 150K trials. On the 720 transfer questions, the net made 7% errors. It seems likely that the network has developed a temporal representation for the events, and is able to apply that to answering questions about the relationship of a pair of intervals that it has not been exposed to before.

6. Conclusion

Because the identification and discrimination of temporal intervals is a common problem across many areas of artificial intelligence, these results have broad implications. For instance, it is possible to apply temporal processing to connectionist models of plan recognition [9]. While the results also have clear application to language, these results are certainly not a complete model of human language acquisition or use. The full complexity of past tense formation has not been captured in Section 3. However, some extensions such as the generation of tenses involving the relative order of events should be straightforward. In addition to the cognitive issues, other networks should be explored. It would be of interest to employ models which move away from the discrete-trials procedure employed here and use more continuous intervals.

REFERENCES

1. Jordan, M.I., 1986, "Attractor dynamics and parallelism in a connectionist sequential machine." *Proceedings of the Cognitive Science Society* (Amherst, MA, Aug. 1986), 531-546.
2. Allen, R.B., 1988, "Sequential connectionist networks for answering simple questions about a microworld." *Proceedings of the Cognitive Science Society* (Montreal, Aug. 1988), 489-495.
3. Allen, R.B. and Riecken, M.E., 1988, "Reference in connectionist language users." *Connectionism in Perspective* (Zurich, Oct. 1988).
4. Allen, R.B., "Connectionist language users." submitted.
5. Allen, R.B. and Kaufman, S.M., 1989, "Identifying and discriminating temporal events with connectionist language users." *IEE Conference on Artificial Neural Networks* (London, Oct. 1989).
6. Rumelhart, D.E. and McClelland, J., 1986, "On learning the past tense of English verbs." In J.L. McClelland and D.E. Rumelhart (Eds.) *Parallel distributed processing*. Vol. 2. Cambridge MA: MIT Press. 216-271.
7. Pinker, S. and Prince, A., 1988, "On language and connectionism: Analysis of a Parallel Distributed Processing model of language acquisition." In S. Pinker and J. Mehler (Eds.), *Connections and Symbols*. Cambridge, MA: MIT Press. 73-193.
8. Allen, J.F., 1983, "Maintaining knowledge about temporal intervals." *Communications of the ACM, 26,* 832-843.
9. Allen, R.B. "A connectionist model for plan recognition." *IJCAI Plan Recognition Workshop* (Detroit, Aug. 1989).

Visual Navigation with a Neural Network

Nicholas G. Hatsopoulos & William H. Warren, Jr.
Departments of Psychology and Cognitive and Linguistic Sciences, Brown University, Providence, RI 02912

Understanding the control of locomotion is a major problem in psychology, neuroscience, and robotics. Among the perceptual systems involved in navigating through the environment, vision plays a major role in controlling locomotion in animals from insects to humans. In this paper, we focus on the information afforded by the changing structure of light at an eye due to the movement of an organism relative to a rigid environment, known as optical flow. It is of great practical and theoretical interest whether an organism can rely on optical flow to determine its direction of self-motion, or heading. We present here a simple linear network, based on the known architecture of areas MT and MST in primate visual cortex which models the low level brain process by which an organism could determine its heading as it translates in a planar environment.

J.J. Gibson (1947)[1] was the first perceptual psychologist to point out the invariant radial pattern of optical flow emerging from translatory motion of an observer in a rigid environment. If the temporally changing optic array is represented as a vector field of optical velocities, the vectors form a radial pattern emanating from a center point, called the focus of outflow or focus of expansion. This global radial pattern is a natural consequence of optical perspective and is invariant with respect to the depth structure of the environment. Gibson claimed that this invariant is detected by the observer and used to visually control its locomotion.

Our derivation of the optical flow produced by movement relative to an environmental element is based on that of Rieger (1983)[2] and relates the optical velocity of a point **P** expressed in terms of spherical coordinate basis vectors $(\hat{\theta}, \hat{\phi})$ to the observer's movement:

$$v_p = \alpha M\hat{V} + N\Omega \tag{1}$$

where α is a scalar consisting of a product of coefficients representing the observer speed and distance to the point **P**, $\hat{V}$ is a unit vector in Cartesian coordinates pointing in the direction of translation, M and N two non-square matrices of trigonometric quantities, and $\Omega \equiv (W_x, W_y, W_z)$, the rotation of the observer. Since we are assuming pure observer translation, the rotation term goes to zero and equation (1) simplifies:

$$v_p = \alpha M\hat{V} \tag{2}$$

The above mathematical analysis takes as givens the observer's movement parameters and derives the flow produced by a point in the environment. The problem to be solved by the neural network is the inverse problem: given the flow at certain points in the ambient optic array, find the movement parameters of the observer. A two-layer linear network is adequate to solve this problem. Algebraically, the problem can be solved by finding the inverse of M:

$$\hat{V} = M^{-1}v_p/\alpha \tag{3}$$

[1]Gibson, J. J. (1947) Motion picture testing and research (AAF Aviation Psychology Research Report No. 7). Washington, DC: Government Printing Office

[2]Rieger, J. H. (1983). Information in optical flows induced by curved paths of observation, 3, 339-344.

However, M is not square and, therefore, does not have a unique inverse. A second image point must be sampled to solve for these three unknowns. The basic two-layer feedforward architecture of the network was constant throughout the research described in this paper (Anderson, 1983)[3] and incorporated features of recent work by Sereno (1987)[4]. A set of neuron-like input nodes or cells representing the velocity field of a sample of the ambient optic array fed signals through a fully connected matrix of weighted connections to a layer of output cells representing the direction of heading. The input layer of nodes are modelled after the cells in the middle temporal area of the superior temporal sulcus. For simplicity, we used a triangular tuning curve for both speed and direction coding in the input layer instead of the gaussian-like curves found by Rodman & Albright(1987)[5]. In order to determine cell activity, we implemented a product rule (cell firing rate = speed activation x direction activation) in the network in order to match the neurophysiological data (Rodman & Albright, 1987) which indicates no interaction between speed and direction tuning. For computational reasons, we used cells with only four different primary direction selectivities and two primary speed selectivities for most of the simulations, making a total of eight input cells per visual field location. Also, we limited the visual field in our simulations to 20 deg x 20 deg consisting of 4 deg x 4 deg non-overlapping input cell receptive fields. Therefore, the input layer consisted of 200 cells, 8 cells per receptive field region times 25 receptive field regions.

The output layer of nodes corresponds to MST cells and represents direction of heading. We decided on a topographic coding of heading in which azimuth and declination are coded conjunctively and distributively. To be consistent with the input layer, We used 25 cells to form a 5 x 5 grid representing a 20 deg x 20 deg portion of the visual field. Each cell responds maximally to a particular heading direction. Response drops off linearly as a function of Euclidian distance between the heading of maximal response and the heading direction to be coded: The inverse mapping from distributed coding of heading to azimuth and declination necessary during the testing phase required an interpretive rule. The azimuth and declination are calculated by taking the average of the azimuths and declinations of maximal response for all cells weighted by each cell's level of activity.

Twenty five simulations were performed in all, each defined with respect to the stimulus set used to train the network. The Widrow-Hoff error-corrrecting learning algorithm (1960)[6]. was used to train the network. We present the results of two simulations which are particularly informative.

Simulation #1

The training set consisted of 400 flow fields corresponding to 400 randomly selected heading directions. The network was first tested on the training set after learning had stopped. Average heading error (averaged over elevation and azimuth) was about 0.83 deg. It is crucial that the network generalize to flow fields produced by new random dot planes and new observer speeds. The network's accuracy on 100 new flow fields corresponding to 100 randomly chosen headings was nearly identical, about 0.89 deg.[7]

[3] Anderson, J. A. (1983). Cognitive and psychological computation with neural models. IEEE Transactions on Systems, Man, and Cybernetics, 13, 799-815

[4] Sereno, M. (1987). Implementing stages of motion analysis in neural networks. Ninth Annual Conference of Cognitive Science Society, 405-416.

[5] Rodman, H. R., & Albright, T. D. (1987). Coding of visual stimulus velocity in area MT of the macaque. Visual Research, 27, 2035-2048.

[6] Widrow, B., & Hoff, M. E. (1960). Adaptive switching circuits, WESCON Convention, Record Part 4, 96-104

[7] These new flow fields may correspond to novel heading directions although the chances are slim that many of them were not contained in the training set.

Simulation #2

In simulation #2, the 400 training optical flow fields corresponding to 25 heading directions which were evenly spaced through out the 20 deg x 20 deg visual field. Remarkably, the network performed better on 100 novel flow fields corresponding to 100 randomly selected heading directions (0.77 deg) than it did on the training set (1.16 deg), and more than twice as accurately on optical flow fields for heading directions that were 2.5 deg out of phase with respect to those in the training set (0.49 deg).

Since simulation #2 generalized so well to novel headings, tests of noise sensitivity were conducted with it. Two kinds of additive, independent noise were introduced: speed and directional. Additive noise was introduced into the optical flow field by taking each velocity vector independently and disturbing either its magnitude (speed) or its direction according to a probability distribution with a mean of zero and a specified standard deviation. Under various noise conditions, the performance of the network was hardly affected. These findings are very similar to those of Warren et al., 1989) [8], who found that human observers perform as well with flow fields whose magnitudes are randomly scrambled (heading accuracy of 0.9 deg) as they do with proper vector fields (0.7 deg). Thus, both the network and the visual system appear to have abstracted the fact that it is the radial pattern of vector directions that provides information for heading, generalizing over variations in vector magnitudes.

To examine the network's noise tolerance in this radial pattern, additive directional noise was introduced by randomly perturbing the direction of each vector from its proper orientation within a specified envelope. A drop off in heading error was observed as a function of standard deviation which is qualitatively similar to the results of a human experiment conducted to test this prediction (Figure 1) (Warren et al., 1989). Nevertheless, heading accuracy was quite good even with a 180 deg range of uniform noise, yielding a heading error of only 3 deg. The network's tolerance of noise again suggests that the redundancy in the global flow field is being used to solve the problem.

In order to visualize the connectivity structure of the network, the weights connecting the input layer with the output layer in simulation #1 are displayed in Figure 2. The diagram shows the weights connecting all the low speed input cells to a single output cell, where the circle indicates the direction of heading to which the output cell is most selective. The orientations of the lines represent the directions of primary selectivity of the input cells, and their lengths represent the magnitudes of the weights; solid lines indicate positive weights, dotted lines indicate negative weights. It is clear that the output cells have become selective to radial patterns of optical flow similar to a class of cells that Saito et al. (1986)[9] identified in MST. Each output cell is maximally responsive to a radial pattern with a particular focus.

By observing the resulting weighted connections between the input and output layers, it becomes clear now why the network generalized so well to stimuli corresponding to new random dot configurations and to new observer speeds in simulation #1. The network's output nodes are acting like a set of radial flow filters or templates. The output node whose template best matches the novel flow field will be most activated. A stimulus correponding to a novel random dot configuration differs from a training stimulus in that the random dots fall in different locations in the receptive fields of the input layer so that the flow vectors registered by each receptive field vary somewhat for a given heading direction. Novel random dot configurations produce a sort of directional noise in the flow field. However, it is a special kind of noise that decreases with visual angle from the

[8] Warren, W. H., Jr., Griesar, W., Blackwell, A. W., Hatsopoulos, N. G., &, Kalish, M. (1989). On the sufficiency of the velocity field for perception of heading. Unpublished manuscript.

[9] Saito, H., Yukie, M., Tanaka, K., Hikosaka, K., Fukada, Y., & Iwai, E. (1986). Integration of direction signals of image motion in the superior temporal sulcus of the macaque monkey. Journal of Neuroscience, 6, 145-157.

focus of expansion. Since flow vectors near the focus of expansion are not weighted very much and directional noise away from the focus is small, a stimulus corresponding to a novel random dot configurations will highly activate the output node corresponding to the correct heading. Moreover, the network generalizes to new observer speeds because they do not affect the radial pattern of the flow field. Only the magnitudes of flow vectors are affected. Since the network's output nodes are particularly tuned to the radial structure of the flow field, variability in vector magnitude will not greatly affect the network's performance.

In conclusion, many of the results found in perceptual experiments with humans have been replicated qualitatively with the model. First, heading accuracies on the order of 1 deg were found using high density, noiseless optical flow fields which is similar to the results of human experiments done in our lab (Warren et al., 1988) [10] Second, the network's behavior in response to additive noise in the flow fields was strikingly similar to that of humans. Heading accuracy was hardly affected by speed noise, which is not only consistent with the human data but agrees nicely with Gibson's radial outflow hypothesis. Likewise, the slight degradation of heading accuracy with increasing directional noise variance matches the human data described above (Warren, et al., 1989).

There are three appealing characteristics of the neural model besides the close match with the psychological data. First, unlike many of the other computational models, it makes no assumptions about the structure of the environment, in contrast to many models that assume a smoothness constraint. Since the network is picking up the radial invariant structure, it succeeds in more complex non-planar environments. Specifically, the model has been shown to perform well under various speed noise conditions that approximate complex 3D environments. Second, the model is, on a crude level, biologically plausible. The brain contains simple velocity- sensitive elements working in parallel. In particular, there is neurophysiological evidence that a mapping from velocity-sensitive units to radial pattern-sensitive units exists in primate visual cortex. Third, the model is surprisingly tolerant of noise, unlike many previous approaches. This property makes it potentially useful for applied problems in computer vision and robotics.

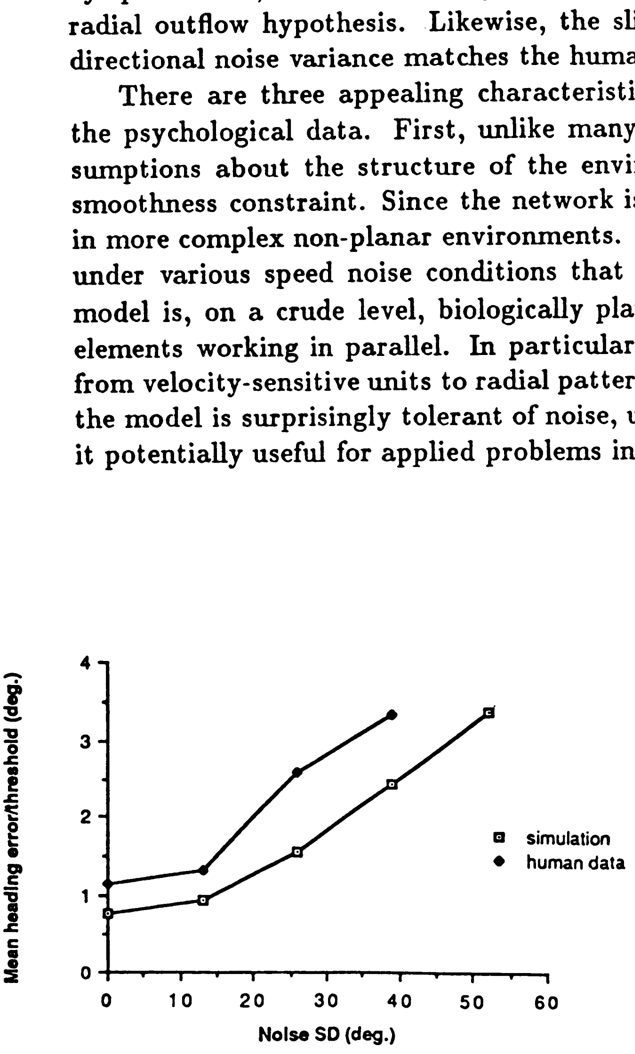


Figure 1

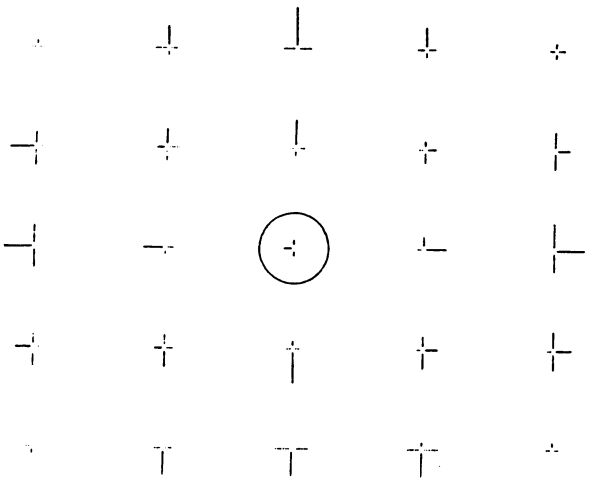


Figure 2

[10]Warren, W. H., Jr., Morris, M. W., & Kalish, M. (1988). Perception of translational heading from optical flow. Journal of Experimental Psychology: Human perception and performance, 14, 646-660.

An Unsupervised Learning Procedure that Discovers Surfaces in Random-dot Stereograms

Geoffrey E. Hinton and Suzanna Becker
Department of Computer Science, University of Toronto
10 King's College Road, Toronto M5S 1A4, Canada

Abstract

A major goal of research on unsupervised learning procedures is to discover an objective function that defines the quality of an internal representation without any externally supplied information about the desired outputs of the system. If such a function could be found, it should allow a hierarchy of representations to be organized bottom-up in a time roughly linear in the depth of the network. This would allow much faster learning than supervised procedures which are generally very slow in networks with many layers of hidden units. Following (Gibson, 1950), we propose that a good objective for perceptual learning is to extract higher-order features that are coherent across time or space. This can be done by maximizing the explicit mutual information between parameters extracted from spatially or temporally adjacent parts of the input.

Introduction

The intensity values in one patch of an image contain information about the intensity values in nearby patches, but this information is in a complicated form because the imaging process combines several different physical parameters to produce the intensity of each pixel. If we could first extract important, underlying, intrinsic parameters such as depth, reflectance, or surface orientation, we could then express the mutual information between neighboring patches in a simpler form. This suggests that we could insist on the mutual information being in a simple form and search for the parameters that must be extracted to allow this. One obvious simple form of spatial coherence is for the underlying parameters for one patch to be equal to underlying parameters for the neighboring patch plus some gaussian noise. Another more interesting form of coherence, which would be appropriate for the depth of slanted planes, is for a parameter value extracted from one patch to be the average of the values extracted from neighboring patches. We describe a family of learning procedures that start by making an assumption about the form the coherence will take and then try to discover parameters that are coherent in this way.

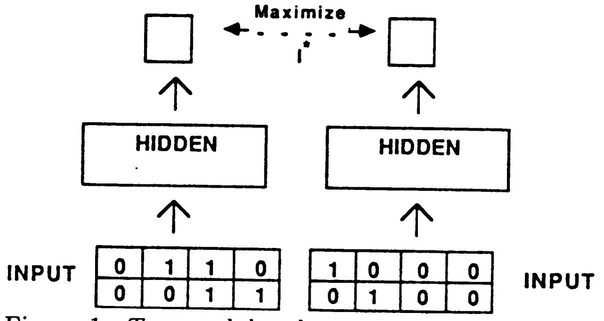


Figure 1: *Two modules that receive input from adjacent, non-overlapping parts of the image. Each module has one layer of hidden units. The learning algorithm adjusts the weights in each module to maximize the mutual information, over the ensemble of training cases, between the states of the two output units.*

Some unpublished results of Peter Brown suggest that a good way to implement this general idea is to try to maximize the explicit mutual information between pairs of parameters extracted from adjacent but non-overlapping parts of the input. The mutual information between two binary variables, a and b, is given by

$$I(a;b) = H(a) + H(b) - H(a,b)$$

where $H(a)$ is the entropy of a, and $H(a,b)$ is the entropy of the joint distribution of a and b. The equation shows that the mutual information between two variables can only be high if each variable has high individual entropy. This is one advantage of mutual information over measures like the correlation between two variables. Mutual information forces each variable to convey a lot of information about the image. Figure 1 shows how this objective function can be used in a multilayer network. The derivative of the mutual information between the outputs of two local modules provides error signals that are back-propagated in order to train the modules.

A very simple example

Our method works well for the task of discovering depth in an ensemble of very simple, binary random-dot stereograms. Each input vector consists of a one dimensional strip from the right image and the corresponding strip from the left image. The right image is purely random and the left image is generated from it by choosing, at random, a single global shift. So the input can be interpreted as an approximation to a one-dimensional stereogram of a fronto-parallel surface at an integer depth. The only local property that is invariant across space is the depth (i.e. the shift). Hence, if one module looks at one area of the two images, and another module looks at another area, the only way they can provide mutual information about each other's outputs is by representing the depth.

We used two global shifts (one pixel rightwards or one pixel leftwards) operating on binary image strips. Each pair of strips was divided into 4 by 2 patches with a gap of one pixel between patches. Each 4 by 2 patch was used as input to a separate module that contained a layer of hidden units and one stochastic binary "output" unit, that used the logistic non-linearity to determine the probability of outputting a 1. Each of the output units tried to maximize the sum of its mutual information (over the ensemble of training cases) with each of the other output units. The derivatives of this objective function are simple to compute using two passes through the training set (Becker and Hinton, 1989). In the first pass we accumulate the probability that each output unit is active, and also all the pairwise joint probabilities. In the second pass we use these accumulated probabilities to compute, for each training case, the derivatives of the objective function w.r.t. the output of a module, and these derivatives are then backpropagated and accumulated to determine the direction of the vector of weight changes within the module at the end of the second pass. The magnitude of the weight change vector is determined by a crude line search along the direction of steepest descent.

With random patterns and a small training set size there is a high probability that units will learn some of the random structure in the data in addition to the shift; as the number of training cases increases, and as we increase the number of modules (and hence the size of the input), sampling error decreases and units become more tuned to shift. The most shift-tuned network we tested had 5 modules and took about 500 passes for the learning to converge on a training set of 1000 random patterns. When we present the complete set of unambiguous binary patterns to a smaller network consisting of just two modules, we usually get output units that are pure shift detectors within about 300 passes through the training set.

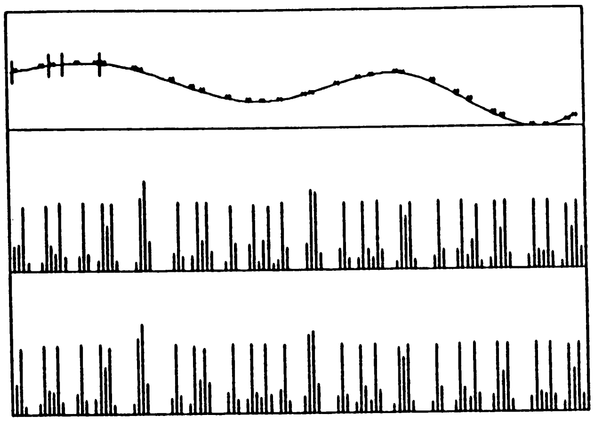


Figure 2: *Part of a cubic spline fitted through seven randomly chosen control points, with randomly located features scattered on it, and the "intensity" values in the two images of the surface strip. The images are made by taking two slightly different parallel projections of the feature points, filtering the projections through a gaussian, and sampling the filtered projections at evenly spaced sample points. The sample values in corresponding patches of the two images are used as the inputs to a module. The boundaries of two neighboring patches are shown on the spline.*

The learning is rather slow for two reasons. First, we are not specifying desired values for the "output" units—we are only specifying that their pairwise mutual information should be high. The derivative of this objective function w.r.t a weight depends on *three* other layers of adaptive weights—one other layer in the same module and two layers in the adjacent module. So in this respect the difficulty of learning resembles back-propagation through four layers of weights. Second, with random starting weights, the initial gradient of the objective function is very small. The convergence speed is greatly increased by using a "bootstrapping" method that starts by applying the objective function between pairs of units in the first hidden layers of pairs of modules until these units are somewhat tuned to the shift. Then the gradients of the mutual information between the output units are much bigger and the objective function can be applied at that layer and the derivatives back-propagated. More globally coherent information can now be provided to the hidden units that failed to find any useful features in the bootstrapping phase.

We compared the performance of the algorithm on a network with 2 modules, and 8 hidden units per module, on the "complete pattern set" with and without 50 bootstrapping iterations. After bootstrapping, units were highly shift-tuned within 50 learning iterations, and by 250 iterations the algorithm nearly always found a globally optimal solution where the mutual information reached 1 bit (in 43 out of 50 repetitions). Without bootstrapping, while the top-level units always became highly shift-tuned, only in 7 out of 50 repetitions did they converge to the global maximum.

Modules with real-valued outputs

The binary output units we used in our initial experiments are suitable for extracting binary features, but they make it difficult to represent depth in more realistic images that contain smoothly curved surfaces which have real-valued depths. In the following simulations, we use images like those shown in figure 2 and modules with deterministic, real-valued outputs that learn to represent real-valued depths (disparities) with sub-pixel accuracy.

We start by making the following very simple coherence assumption (which will be relaxed later): There is some locally detectable parameter which is approximately constant for nearby patches. So, given two modules A and B that receive input from neighboring patches, we want the output of A, a, to be approximately equal to the output of B, b. We can think of b as a signal that we are trying to predict and a as a noisy version of that signal that is corrupted by additive, independent, gaussian noise. If we assume that both a and b have gaussian distributions, the information (ignoring a factor of 2) that a provides about b is determined by the ratio of two variances:

$$I_{a;b} = \log \frac{V(signal + noise)}{V(noise)} = \log \frac{V(a)}{V(a-b)}$$

So, for a to provide a lot of information about b we need a to have high variance but $a - b$ to have low variance. For symmetry, we actually optimize the following function:

$$I^*_{a;b} = I_{a;b} + I_{b;a} = \log \frac{V(a)}{V(a-b)} + \log \frac{V(b)}{V(a-b)}$$

Some possible variations of this objective function are discussed in (Becker and Hinton, 1989).

Speeding the learning using radial basis functions

Instead of using the bootstrapping method described above to speed the learning, we used an alternative method in which the adaptive hidden units of each module are replaced by a large number of non-adaptive radial basis functions (Moody and Darken, 1989). Each radial basis unit has a "center" that is equal to a typical input vector selected at random, and gives an output which is a

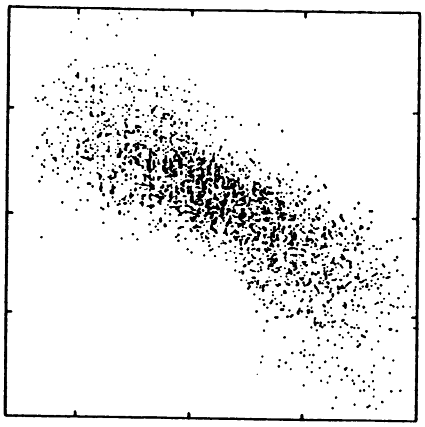


Figure 3: *The activity of the output of a module (vertical axis) as a function of the disparity (horizontal axis) for all 3000 training cases using planar surface strips.*

gaussian function of the distance between the current input vector and the unit's "center". All the gaussians have the same variance which is chosen by hand. So the only adaptive weights in a module are those from the radial basis units to the output unit. In the experiments with curved surface strips each module had 8 by 2 input units connected to a layer of 100 radial basis units. Every module used exactly the same set of radial basis functions so that we could constrain all the modules to compute the same function.

Discovering real-valued depth for planar surfaces

Figure 2 shows how we generate stereo images of curved surface strips. The same technique can be applied to generate images of planar surfaces with randomly chosen slants. Using 3000 training cases of this simpler type of input, we trained a network that contained 10 modules each of which tried to maximize I^* with the immediately adjacent modules. Each update of the weights involves two complete passes through the training set. In the first pass, we compute the mean and variance of each output value and the mean and variance of each pairwise difference between output values given the current weights. In the second pass we compute the derivatives of I^* for each pair of modules, and use these derivatives to accumulate dI^*/dw for all weights, w, from the radial basis units to the output units. Then we update all the weights in parallel using steepest descent with a simple line search. After each weight update, we average corresponding weights in all the modules in order to enforce the constraint that every module computes exactly the same function of its input vector. After 30 weight updates, the output of a typical module gave a good representation of the disparity as shown in figure 3.

More complex types of coherence

So far, we have used a very simple model of coherence in which an underlying parameter at one location is assumed to be approximately equal to the parameter at a neighboring location. This model is fine for fronto-parallel surfaces but it is far from the best model of slanted or curved surfaces. Fortunately, we can use a far more general model of coherence in which the parameter at one location is assumed to be an unknown linear function of the parameters at nearby locations. The particular linear function that is appropriate can be learned by the network.

We used a network of the type shown in figure 4 (but with 10 modules and with a contextual predictor unit for all modules except the two at the ends). We tried to maximize I^* between the output of each module and the contextual prediction of this output that was produced by computing a linear function of the outputs of one adjacent module on each side. We used weight averaging to constrain this interpolating function to be identical for all modules. We also back-propagated the error derivatives through the interpolating weights. Before applying this new objective function, we first used a bootstrapping stage in which we maximized I^* between adjacent pairs of modules as before, for 30 learning iterations.

After having been trained for 100 iterations on 3000 patterns, the two weights learned for the interpolating function were .55, .54. The output of each of these units is similar to the response profile shown in figure 3, but even more finely depth-tuned. Thus, the interpolating units have learned that the depth at one patch on a planar surface can be approximated by the average of the depths of surrounding patches.

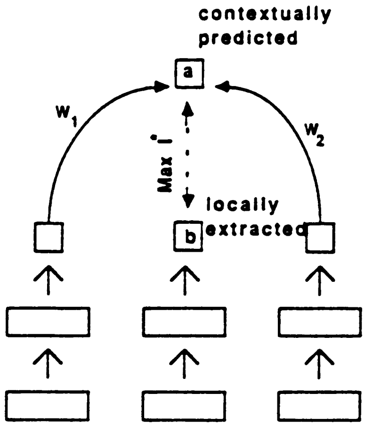


Figure 4: *A network in which the goal of the learning is to maximize the information between the output of a local module and the contextually predicted output that is computed by using a linear function of the outputs of nearby modules.*

Discovering coherence in curved surfaces

As we introduce curvature in the surfaces, the prediction of depth from neighboring patches becomes more difficult; at regions of high curvature, a simple average of the depths of 2 adjacent patches will under- or over-estimate the true depth. In this case, a better interpolator would base its predictions on more than two local measurements of depth, thereby taking curvature into account.

We trained a network of 10 modules on 1000 of the stereograms of curved surface strips, using the same architecture and objective function as for the planar surface task, for 30 iterations. We then added an interpolating layer; this time, however, the contextual prediction of a given module was a linear function of the outputs of *two* adjacent modules on either side. After 100 iterations, the four weights learned for the interpolating function were $-.04, .64, .65, -.04$. Positive weights are given to inputs coming from the immediately adjacent modules, and smaller negative weights are given to inputs coming from the more distant neighbors. The activity of these units is well tuned to disparity, as shown in figure 5. Given noise-free depth values, the optimal linear interpolator for the surface strips we used is approximately -.2, .7, .7, -.2. But with noisy depth estimates it is better to use an interpolator more like the one the network learned because the noise amplification is determined by the sum of the squares of the weights.

Discussion

Discovering how to predict one value from a linear combination of nearby values is equivalent to finding a linear combination of all the values that always equals zero (Richard Durbin, *personal communication*). This amounts to discovering invariant higher-order properties by learning invariance detectors that have low variance even though their input lines have high variances (when weighted by the squares of the weights). One attractive aspect of this view is that the actual output of an invariance detector would represent the extent to which the current input violates the network's model of the regularities in the world. This is an efficient way of transmitting information about the current input.

An invariance detector that minimizes the ratio of its output variance divided by the variance that would be expected if the input lines were independent gaussian variables is a real-valued, deterministic version of the G-Maximization learning procedure (Pearlmutter and Hinton, 1986) which finds regularities by maximizing the extent to which the independence assumption is incorrect in predicting the output of a unit. It also has an interesting rela-

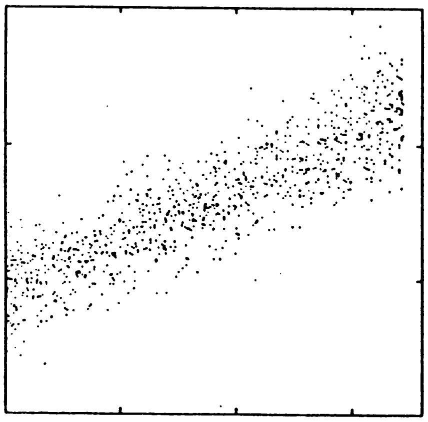


Figure 5: *The output of a unit (vertical axis) as a function of the local disparity (horizontal axis) when trained on 1000 curved surface strips. The unit learned to predict the depth locally extracted from one module as a linear function of the outputs of the 2 adjacent modules on either side.*

tion to Linsker's learning procedure (Linsker, 1988). Linsker assumes the variances and covariances of the activities on the input lines to a unit are fixed (because he does not backpropagate derivatives) and he shows that, with the appropriate gaussian assumptions, the information conveyed by the unit about its input vector is maximized by using weights which *maximize* the ratio of the output variance divided by the sum of the squares of the weights.

We have described the learning procedure for modules which each have a single real-valued output. For modules with several real-valued outputs, the natural way to generalize the objective function is to replace the variance by the determinant of the covariance matrix. It remains to be seen whether this causes unacceptable problems with the learning speed and whether it can be modified to avoid the difficulties that arise as the covariance matrix becomes singular.

We have also ignored the ubiquitous problem of discontinuities. Images of real scenes have strong local coherence punctuated by discontinuities. We do not want our learning procedure to smear out the strong local coherence by trying to convey information across the discontinuities. We would prefer a module to make accurate predictions in continuity cases and no predictions in other cases rather than making rather inaccurate predictions in all cases. We can achieve this by letting each module use a mixture of two gaussian models to predict the output of a neighboring module. One part of the mixture model is for continuity cases, and the other part is for discontinuity cases. During learning, the module computes the probability that the current case is an example of continuity by comparing the probability densities, under both its continuity and its discontinuity models, of the observed output of its neighbor. The contribution to the accumulated gradient is then made proportional to the probability that the current case is a continuity case. This means that clear cases of discontinuity do not affect the weights learned by the continuity model.

In this paper, we have used coherence across space, but the same techniques could be applied to coherence across time. The procedure we have described has several appealing properties. First, it builds into the objective function (and the architecture) a type of prior knowledge that is strongly constraining but widely applicable to perceptual tasks. Second, using the bootstrapping approach it may be possible to train deep networks fairly rapidly, provided the domain is such that the very high-order features that exhibit very long-range coherence can be built out of lower-order features that exhibit shorter range coherence.

Acknowledgements

We thank Peter Brown, Francis Crick, Allan Jepson, and Barak Pearlmutter for helpful discussions. This research was supported by grants from DuPont, the Ontario Information Technology Research Center, and the Canadian National Science and Engineering Research Council. Geoffrey Hinton is a fellow of the Canadian Institute for Advanced Research.

References

Becker, S. and Hinton, G. E. (1989). Using spatial coherence as an internal teacher for a neural network. Technical Report in preparation, University of Toronto.

Gibson, J. J. (1950). *The perception of the visual world.* Houghton Mifflin, Boston, Mass.

Linsker, R. (1988). Self-organization in a perceptual network. *IEEE Computer,* March, pages 105–117.

Moody, J. and Darken, C. J. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294.

Pearlmutter, B. A. and Hinton, G. E. (1986). G-maximization: An unsupervised learning procedure for discovering regularities. In Denker, J. S., editor, *Neural Networks for Computing: American Institute of Physics Conference Proceedings 151*, pages 333–338.

EXPERIMENTS ON CONSTRUCTING A COGNITIVE MAP: A NEURAL NETWORK MODEL OF A ROBOT THAT DAYDREAMS

Larrie Hutton and Vincent Sigillito
The Johns Hopkins University Applied Physics Laboratory; Laurel, MD 20707
Phone (301) 953-6242

Howard Egeth
The Johns Hopkins University Department of Psychology; Baltimore, MD 12345

Abstract

We describe a recurrent neural network, with psychologically motivated assumptions about the underlying architecture, that exhibits behavior consistent with that seen in living cognitive systems. In particular, we find that such a system models abstract cognitive maps of at least 2-dimensional cognitive space by showing evidence (1) of habit formation (increasing rigidity of behavior with repeated exposure); (2) of the effects of forced movements away from obstacles (external objects have nonsymbolic representations); and (3) of learning to "navigate" appropriately when contextual information was changed dramatically. Furthermore, the system is capable of showing "surprise" when there is a large mismatch between an anticipated event and an actual event (and is thus consistent with the neuronal model of Sokolov, 1960) and of recognizing a goal state (i.e., the system spontaneously switches to a new task upon reaching a goal, or it "waits" for a new input.) Although the system does not employ any novel architectures or assumptions, the interpretation of the system's behavior is original. Because the system can be described as a highly nonlinear system of coupled difference equations, for example, it shows properties characteristic of such systems (such as the appearance of spurious attractors and of extreme sensitivity to initial conditions). These same characteristics, however, give the system the ability to engage in "daydreaming", to show creativity, and to "rehearse" solutions in short-term memory.

Introduction

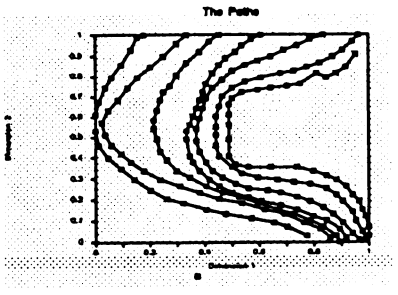
In this paper, we describe a method that a cognitive system might use to construct a cognitive map (Tolman, 1948). The approach that we use is a bit unorthodox. We did not start out with a problem to be solved and invent (or discover) an appropriate transformation of the data, upon which an appropriate architecture was invented (or discovered) to map an I/O relationship. Rather, we started out with an architecture that we felt was defensible and observed the network's behavior as it trained, and was tested, under conditions that we assumed were a plausible representation of a cognitive space. Thus our role was as much that of a trained observer as it was that of an active designer. We feel that this is an important role--that of observer--that is too often neglected in neural net research. (A parallel in animal behavior might be drawn between psychologists, who specialize in experimental manipulations, and naturalists--ethologists, for example--who are relatively noninvasive observers of animals in their natural habitat.)

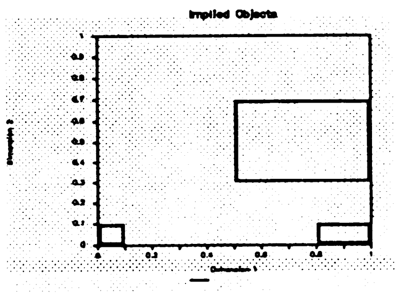
We chose a model architecture that we felt satisfied the minimal conditions required of a system that could be expected to engage in "thoughtful" spontaneous behavior. This implied an input representation that was isomorphic with the output representation; that is, a transformed representation of the input would have to serve as the sole input to the system while it was daydreaming. (Daydreaming is defined as a free-running state in which inputs from the external world are blocked. Defined in this way, a Hopfield net is daydreaming when it is engaged in associative recall. However, "associative recall" captures neither the architecture nor the cognitive spirit of our model.) We also wanted the model to be capable in principle of possessing (or even better, constructing) an internal representation that permitted the solutions to problems that involve interactions in the input data. Finally, the model needed to be dynamic in nature, changing in "meaningful" ways over time, to possess emergent properties that admitted to a cognitive interpretation, and to be both resistant and sensitive to stochastic noise. The latter requirement is quite frankly internally inconsistent. Nevertheless, living cognitive systems are relatively resistant to noise (they persist in the face of distraction), and yet they can be exquisitely sensitive to initial conditions (they are "creative" and "inscrutable").

Methodology

The network architecture during the learning process was that of an ordinary supervised feedforward network. Thus the net had full connectivity between adjacent layers and no lateral connections. Although we did use the generalized delta rule, the particular learning algorithm was not especially important for our purposes, since we were more interested in what the net did after learning the I/O map than we were in how it learned the map. On the other hand, the I/O representation itself was very important. The net had 3 input nodes, 3 output nodes, and 10 hidden nodes. The input and output vectors were isomorphic: one node represented a normalized horizontal component (in a range of 0 to 1); a second node represented a normalized vertical component; the third node (not always used) represented the "context" and--although the context nominally represented a categorical variable--was also permitted to vary over the range from 0 to 1. During recall (but not learning), however, the net was recurrent: each output node fed back to its corresponding input node, and only that node. Although we varied the amount of feedback, we show here only the results in which the net's current output (after an initial input) was strictly determined by its last output: i.e., the net "daydreamed" without feedback from its environment. Because we did not want our net to become "lost in thought" as it constructed a cognitive map of its environment, it was important that the intermediate states in the internal representation (as exemplified by the hidden node activities) be expressible in terms of the original environmental variables. It might be said that we are accepting Wittgenstein's (1953) premise of the impossibility of a purely private language by assuring the recoding of an internal representation (the hidden layer) back into a representation that is in principle and in fact a transformation of the original input variables. When the input nodes are completely determined by the previous output, and the input and output representations are isomorphic, then the system has the following useful properties: (1) all internal representations are directly expressible as environmental states (thus making it possible for one cognitive system to communicate about its internal states to another cognitive system); (2) the system can be viewed architecturally as a modified BAM during recall (the hidden layer constitutes one layer of the BAM and passes messages to the I/O layer through the hidden-to-output weights, and the I/O layer passes messages to the hidden layer through the input-to-hidden weights); (3) the weights can be formed through any defensible mechanism, including but not restricted to back-propagation; (4) the system can easily be "tuned" to trade off sensitivity to its own internal state against sensitivity to environmental assaults--clearly a desired result in any cognitive system. Thus, the system might be thought of as a 13-neuron brain that thinks about its past experiences after it has formed a cognitive map of its environment.

All problems described in this paper involved the same basic procedure: use back-propagation to "train" an homuncular system to follow a series of seven fixed paths, and then "test" the system by giving it several random starting points and observing the system's behavior as it "imagined" its environment. (Additional training was provided in some experiments to be described later.) The paths that all of our experimental systems learned about are shown in the first figure. There were some "implied objects" in the environment as well. They are "implied" in the sense that the homuncular system (HMS) "knows" about them only with respect to its own behavior in the region of the objects; thus there is a behavioral rather than symbolic representation of objects in the environment. The second figure shows the implied objects explicitly. The large rectangular object might be thought of as a forbidden zone in which the HMS was either never placed, or punished (by immediate removal), for being inside the area. The smaller rectangle in the lower right-hand corner was in initial goal area; all initial paths terminated in that region, as can be seen in the first figure. The secondary goal area, seen in the lower left-hand area, was an appropriate goal state in later experiments; it was visited only as a secondary site after the initial goal had been visited.



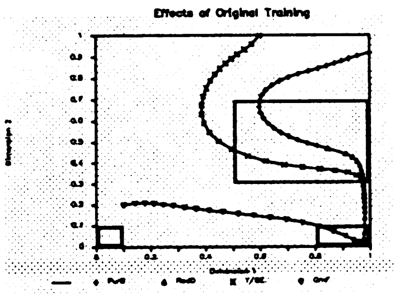


All training sets involved from 1200 to 1600 iterations through the approximately 300 points shown in the first figure. The targets were always the next point in the path (sometimes this included a context dimension, for a total of three output nodes) or, as noted below, three points ahead. We investigated the effects on the system's cognitive map of learning larger steps, of being punished, of being trained on a secondary goal with and without a contextual dimension, and of being given information that was inconsistent with its history (i. e., a "mismatch" between an anticipated and an actual environment). The effects of the manipulations were manifested by changes in path efficiency (did the system learn to take short cuts spontaneously?), by gross behavioral changes in the vicinity of objects or goals, by evidence of chaotic behavior, and by the ability to generalize to novel situations. Our emphasis was on plausible psychological interpretation of a cognitive system that daydreamed after spontaneously forming a cognitive map of two-dimensional space.

Procedure and Results

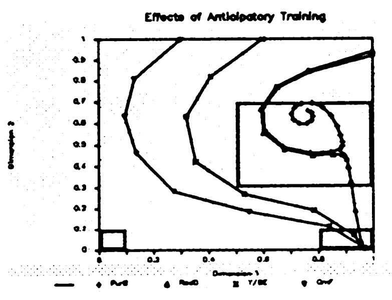
In this section we describe the manipulation and provide a brief interpretation of the results. The reader should keep in mind that the paths shown are completely determined by the initial training set (which determined the weight matrix) and an initial point.

Effects of Original Training



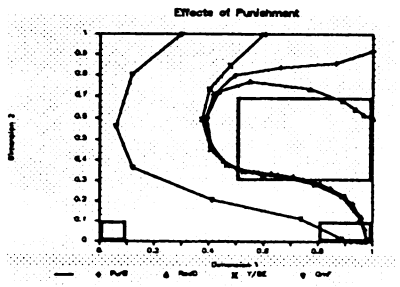
Only three initial points are shown here, but several trends are clear. First of all, the HMS learned to take "short cuts" through forbidden zones, although it still tended to respect the zones. Second, the HMS learned to recognize a goal state; it did not wander aimlessly once it reached the initial goal. Psychologically, the fixed points of the system can be regarded as goal states. Simply recognizing a goal is an important and nontrivial characteristic of a cognitive system: it is equivalent to a system that "knows that it knows" an answer, without an initial awareness of that answer. So our back-propagation has some of the same properties that we see in Hopfield networks (e.g., the presence of fixed points). Note that the HMS has generalized its goal-oriented behavior to include regions (such as the initial point in the lower-left corner) that it had never "seen" before.

Effects of Anticipatory Training



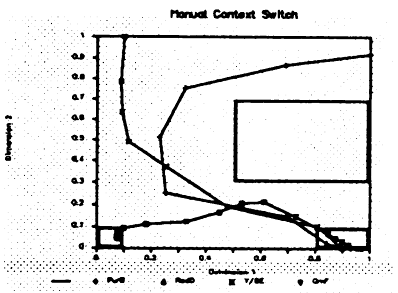
Anticipatory training is defined as training in which the target vector is a point three steps ahead of the input vector (except near the goal). Psychologically, the system can be said to "look farther ahead" in the same way that we ourselves look farther ahead when we drive our automobiles at a higher rate of speed. There are three matters worth noting here: (1) fewer moves are required to reach the goal (cognitive effort is reduced); (2) even though fewer moves are required to go around the barrier, the barrier is nevertheless better accommodated; (3) we see the emergence of spurious solutions. The last item is particularly interesting from a psychological point of view. Although spurious solutions developed without anticipatory training, there appeared to be fewer of them. An important point here is that we have one of the ingredients for a chaotic system: extreme sensitivity to initial conditions. The bifurcation that develops in the upper right-hand corner is a good example. The HMS is sensitive to a an initial condition in which an infinitesimally small change in the starting point results ultimately in a completely different solution or--as in this case--to either a solution or to capture by a basin of attraction that was not explicitly introduced. As an abstract cognitive system, this should be construed as a crucial feature, since it introduces the possibility of truly creative behavior. Wrong perhaps, but creative nevertheless. And if noise (stochastic or otherwise) in the system is greater than our power to resolve the initial conditions, then the divergent solutions would appear to a naive observer as a manifestation of "free will".

Effects of Punishment



We also tested the effects of "punishing" the HMS for being in the forbidden region. (In this and in the following sections, anticipatory training was also in effect). Punishment was effected by placing the HMS slightly within the barrier at the top and the left and then immediately moving the HMS to a point slightly outside the barrier. The most noticeable effect was to cause the HMS to go around the barrier when it would otherwise have been drawn to the strange attractor found in the previous figure. The effect was to cause the HMS to in some sense "think about the world" in a different way. On the plus side, the effect could be construed as increasing the probability of "good solutions"; on the negative side, it could be construed as a loss of creativity.

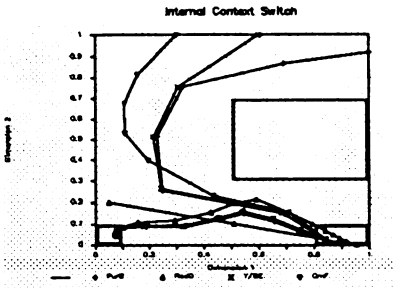
Manual Context Switch



In this condition a manual switch was added, corresponding to a context switch. (In this and in the following sections, anticipatory and punishment training were also in effect). For the first time, a third input and output node were used. When the third node was "off", conditions were exactly as in the previous condition. However, an additional condition was included as well; in this condition only, which was signalled by the "on" state for the third node, the HMS was trained to go from the primary goal to the second goal. In the simulation (after training) shown here, the HMS was started in the two positions shown in the top half of the diagram with the context switch set to "off". As might be expected, the HMS went directly to the first goal and stayed there. When the context node was changed "manually" (by forcing the third input node to "on" after the HMS had persisted in the initial goal), the system immediately moved to the second goal state, where it again persisted. Thus the "meaning" of an event could be changed by changing the context in which it was assessed; ultimate goals are a matter of circumstance.

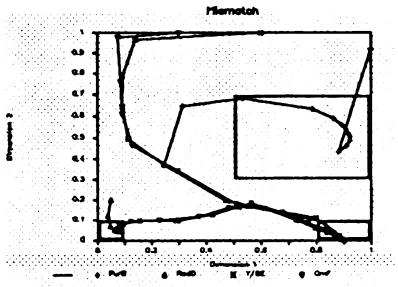
Internal Context Switch

The results of the previous condition suggest an interesting question: can the system learn to change the manual switch internally? If so, the HMS might be able to "decide" what an appropriate goal state happens to be, use its cognitive map to approach the goal, change the goal state, and then to "imagine" approaching a new goal state. (It is the kind of thing that humans might be imagined to do when they, for example, think about going to the kitchen, which may "remind" them to go to the bathroom to wash their hands. We are not going to worry about returning to the kitchen--we have only a 13-neuron brain here.) The



training was similar to the previous condition except that whenever an output vector resulted in entry into the first goal state, an additional I/O vector pair was provided. The input vector of this new pair was the previous output vector (this is nothing new), but the output vector was precisely the input vector with only the context node changed. Thus, the context node effectively reversed itself if and only if the HMS was in the first goal state. It can be seen from the figure that the HMS did indeed learn to go to the first goal, "spontaneously" change the context, and then proceed to reverse the path over which it had just travelled to go to the next goal. Note also that a starting point near the second goal nevertheless resulted in an initial response pattern that, as appropriate, is directed toward the first goal.

Mismatch



In the final condition reported here, we introduced a deliberately confusing circumstance: the HMS was placed in a region on which it had been trained to go to the first goal state, but was told (by setting the context switch to "on") that the appropriate goal was the second. How would the system handle the conflicting information? The most interesting response came from the initial placement in the upper right-hand part of the figure. It might be said that the HMS, which was now free to change any I/O node on the basis of its own cognitive meanderings, effectively decided that we had lied to it. The first response to that initial condition was to move into the forbidden zone and to completely reverse the sense of the context node. The HMS then began to move out of the forbidden zone, around the barrier to the first goal, once again change the context node, and finally to move to the second goal state, where it remained. Although this somehow seems a "sensible" thing to have done, it must be remembered that it had certainly never encountered this sequence before. Yet all of this happened without any external input save the first point! It is worth noting also that the initial point nearest the second goal state resulted in entry into the second goal state directly, bypassing the first goal state entirely. Yet this too seems sensible: the context was initially set to approach a goal state that it happened to be near.

Conclusions

The interpretation of complex systems can be as illuminating as the development of systems that are designed to solve particular problems. That such a simple dynamical system as that described here shares, in an abstract way, the properties that we might expect to find in the development of cognitive maps in biological systems deserves further reflection. There is nothing new here mathematically nor psychologically, but we believe that careful consideration of such simple systems is warranted, even imperative, to develop realistic psychological systems.

References

Sokolov, E. N. (1960). Neuronal models and the orienting reflex. In M.A.B. Brazier (Ed.), *The Central Nervous System and Behavior.* New York: Josiah Macy, Jr. Foundation.

Tolman, E. C. (1948). Cognitive maps in rats and men. *Psychological Review*, 55, 189-208.

Wittgenstein, L. (1953). *Philosophical Investigations.* New York: MacMillan.

Directing Focus of Attention Through Conflict in Depth Perception

Clayton McMillan
Dept. of Computer Science
University of Colorado
Boulder, Colorado 80309-0430

Gerhard Dirlich
Max Planck Institute for Psychiatry
Kraepelinstr. 2
8000 Muenchen 40
West Germany

In this paper we describe a neural network model that develops a piecewise internal representation of the topology of an image over time. The internal representation is constructed in part from a local focus of attention *window* that moves from one *focal point* in the image to another to build a global representation, and in part from existing knowledge about the object's topology. The model is part of a two track investigation of how visual depth perception is disturbed in schizophrenic patients. The two tracks in this investigation are as follows: 1) an empirical exploration of the various cues used in depth perception and how they interact with each other, and in parallel to this track, 2) modeling of the observed phenomena. Performing visual experiments is costly and often contrary to the interests of a schizophrenic patient. Consequently, we view the modeling as an important element of this project. We hope to show that it is a useful and efficient way of choosing what direction the empirical track should take.

Conflicting cues in perception of depth

The phenomenon we are currently concerned with is the perception of surface topology in a post categorical image. In particular, we are interested in how *top down* information and *bottom up* information interact and compete with each other in the development of an internal representation. One experiment illustrates nicely this interaction (Yellot, 1981). When presented with the image of a *hollow* mask, that is, a mask of a human face seen from the inside, people perceive the mask as being convex. In reality the mask is concave and bottom up cues provide strong information to that effect. However, people possess the knowledge that human faces are convex and this top down information appears to override the bottom up perception of the surface topology. Preliminary results from the empirical investigation (Emrich, et al., 1988) show that when such conflicts exist, a subject may observe the image for a period of several minutes before developing an impression that is consistent with his expectation of the surface topology. In addition, eye movements appear to play a critical role in the resolution of such conflicts (Gale & Findlay, 1983). It has been suggested (Marr, 1982) that a small, high resolution focus of attention *window* moves about in a scene collecting local perceptions that are added to a composite perception, somewhat analogous to a blind man feeling a face with his fingers. This *window* is frequently but not always related to eye movements. Interestingly, in schizophrenic patients a conflict between top down and bottom up processing is often never resolved.

Neural network for moving the focus of attention

The neural network we describe here is a model of how this conflict directs the movement of the focus of attention *window* from one fixed focal point to another until the conflict is resolved. Although there are many visual cues used to decide the depth of a local area on a surface, we limit our model to two: 1) light and shading, and 2) stereo disparity. The model is a five layer network that actually consists of two, three layer back propagation networks. The output layer of the first network serves as the input layer to the second (Figure 1). All units in the network assume continuous levels of activation in the range [-1.0, 1.0] and all connections feed forward, except for a set of self inhibitory connections on the units labeled *Internal* and a set of sigma-pi connections from the layer labeled *Focus* to the connections between input and Internal. Input to the network consists of two parts: 1) a pattern representing

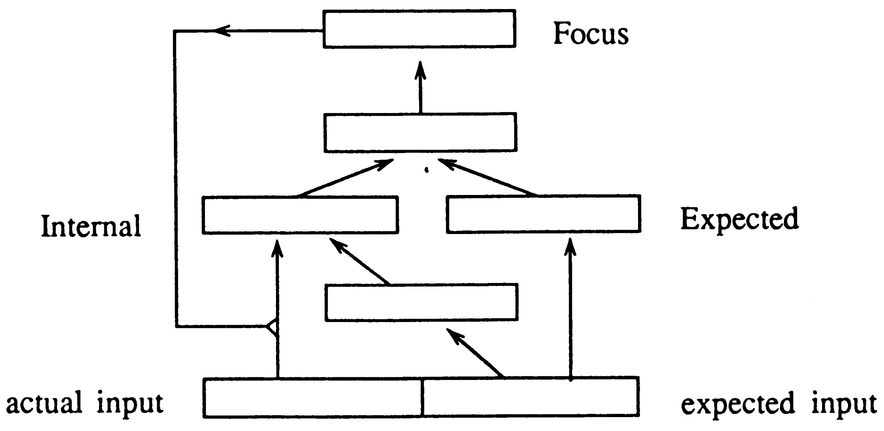


Figure 1

the bottom up, *actual* information collected at fixed focal points, and 2) a pattern representing the top down, *expected* topology of the post categorical image.

The representation of *actual* input is composed of a set of units that contain information about light intensity across a small strip of the surface at the current focal point. If we divide this strip into four equal squares, one unit in the input layer per square, then the activation of the first unit measures the light intensity on the first square, the second unit the light intensity on the second square, etc. A higher activation level indicates greater light intensity. Therefore, a pattern such as (.6, .2, .9, .6) would correspond to a prism shaped object that protrudes from a flat surface, where .6 on both sides is the flat surface, .2 the side of the prism farthest away from the light source, and .9 the side closest to the light source. This pattern of activity indicates a convex surface and holds if we assume the light source is fixed in the upper right. Lehky and Sejnowski (1988) describe how a network can learn to decide depth using light and shading.

In addition to the pattern for light and shading, there is an additional set of four units for determining depth from stereo disparity. This representation assumes that two points A and B have been identified and the correspondence between two retinitopic images of those points has already been determined (Marr (1982) describes how this might be done computationally). In our model, if the point A is represented by the activation value 1.0 and the point B by -1.0, then the four units can be used to identify the relative location of A and B on the left and right retinitopic images. The pattern (1, -1, -1, 1) indicates that the point A lies outside the point B in both images. If we assume that A is always to the left of B on the surface of the object being perceived, then this pattern indicates a convex surface. Similarly the pattern (-1, 1, 1, -1) indicates a concave surface. Activation levels between -1.0 and 1.0 indicate various levels of surface depth.

The representation of *expected* input has the same form as the two layers labeled Internal and Expected. If we assume that focal points on the object's surface have already been identified and are fixed, then each unit in each of these representations corresponds to one focal point. Gale and Findlay (1983) show that in resolving ambiguous figures a limited number of focal points, corresponding to obvious features such as the tip of the nose on a face, seems probable. In the model we describe here, we have used patterns with four focal points, although the number can be arbitrarily large within efficiency constraints. Each of the units in these three layers then, represents the degree to which the corresponding focal point in the observed image is considered to be convex or concave, where a value of 1.0 indicates a maximally convex local surface and -1.0 a maximally concave local surface.

The layer labeled Focus has a similar correspondence between units and focal points. *Focus* serves as the output to the network and its pattern of activity indicates which focal point the focus of attention *window* should take input from in the next time step. The activation of these units is dictated in part by conflicts between the internal representation and the expected representation, and in part by Gaussian noise. Since the focus of attention can reside only at one point at a time, the activation of units in this layer is winner take all. Through the sigma-pi connections to the Internal layer the Focus layer gates input from the *actual* portion of the input pattern to the current focal point in the internal representation. Therefore, the model should tend to focus attention at a point where the greatest conflict exists between expected and perceived input, but at the same time move around randomly from one focal point to another regardless of conflicts. As the internal representation develops a more stable form and conflicts are diminished, random movement of the focus of attention should increase.

Simulations of conflict resolution in surface topology

The behavior of the model can be summarized as follows, then: The Expected layer receives activation directly from the *expected* portion of the input pattern through connections with weights equal to 1.0. One unit in the Internal layer at a time receives activation from the *actual* portion of the input pattern, but all units receive activation in parallel from Expected. Over time an internal representation of the surface topology is developed and conflicts with the expected pattern may arise. The unit in the Focus layer that corresponds to a conflict point will tend to activate and gate input in the next time step from the *actual* portion to the correct local surface in the internal representation. Over time activation from expected input increases in strength and eventually suppresses conflicting input from the *actual* input. In this model this was accomplished through a linearly increasing function of the net activation from *expected* input.

The number of time steps required to develop a stable internal representation then, is a function of the number of conflicts between the *expected* and *actual* input patterns. Figure 2 shows this behavior in three different runs. In each case the curve shows the error as a function of time steps. Error is simply the sum of the differences squared between the Internal layer and the Expected layer. As the difference between the perceived and expected pattern diminishes, so does the error. Case 1 in Figure 2 shows a simulation in which the starting difference is 0.0. A small conflict exists at the beginning in Case 1, however, because the starting Internal pattern is random. As soon as all four focal points have been visited, the conflict is resolved. In Case 2 a conflict exists at focal point 4 and in Case 3 at focal points 3 and 4. A substantially larger number of time steps are spent at those points, 56, 59, and 44 steps respectively, because the movement of the focus of attention is driven by conflict. Cases 2 and 3 show how additional conflict increases the number of steps required to develop a stable interpretation of the surface. The irregular error curve in these simulations is due to the movement of the focus of attention. If a conflict exists at point *n*, but the focus of attention is at point *m*, $m \neq n$, then no input is received from *actual*

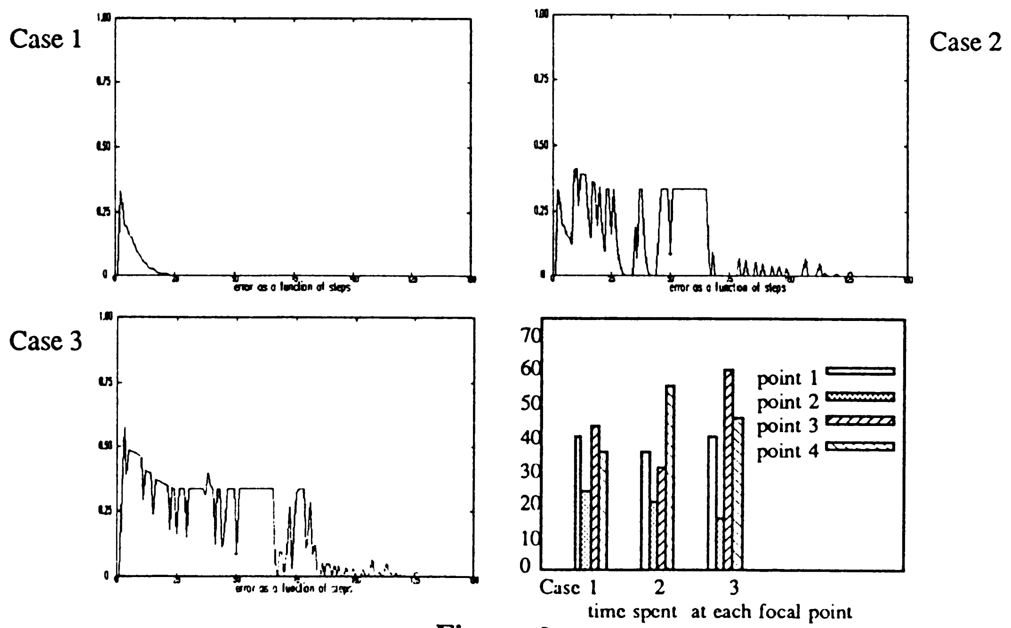


Figure 2

input at *n* during those time steps. In other words, there can be no conflict when the focus of attention is not at point *n*. As the conflict is resolved, the selection of focus becomes more random. Even after a stable interpretation is arrived at, however, decay of the internal representation still drives the movement of the *window* to maintain the composite interpretation.

Conclusion

This neural network model represents the computational implementation of a hypothesis developed to account for the directed movement of the focus of attention during visual perception of depth. By weakening the connections from the *expected*, top down input, it can also be made to exhibit the inability to resolve conflicting information that was observed in schizophrenic patients in the preliminary empirical section of this project. Based upon this model it has been decided that future experiments should center around two major areas: 1) the interaction of low level cues such as light and shading and stereo disparity with each other and with higher level cues such as top down knowledge, and 2) the role eye movements play in the development of a composite perception of depth gathered from local perceptions over time. The modeling in the project has helped test earlier hypotheses about the perception of surface depth, and will be expanded upon or used as a point of departure for more complete models as empirical results become more conclusive.

Bibliography

Emrich, H.M., Weber, M.M., Wendl, A., Zihl, J. (1988): Impaired Binocular Depth Inversion as an Indicator of Psychosis. Submitted.

Gale, A.G., Findlay, J.M. (1983): Eye Movement Patterns in Viewing Ambiguous Figures. In: Groner, R., Menz, C., Fisher, D.F., Monty, R.A. (Eds.) *Eye Movement and Psychological Functions: International Views.* Larence Erlbaum, Hillsdale, New Jersey.

Lehky, S., Sejnowski, T. (1988): Neural Network Model for the Cortical Representation of Surface Curvature from Images of Shaded Surfaces. In: Lund, J.S. (Ed.) *Sensory Processing*, Oxford: Oxford University Press.

Marr, D. (1982): From Images to Surfaces, Stereopsis. *Vision.* Freeman, San Francisco. pp 111-159.

Yellot, J.I. (1981): Binocular Depth Inversion. *Scientific American.* Vol. 245. pp. 118-125.

Acknowledgments

This research was supported by BMFT (Bundesministerium für Forschung und Technologie) and DAAD (Deutscher Akademischer Austausch Dienst). Thanks to Holger Knopf for help on earlier simulation software and to Yoshiro Myata for the use of SunNet.

The effects of threshold modulation on recall & recognition in a sparse auto-associative memory: implications for hippocampal physiology*

Valeriy I. Nenov[1,2], Walter Read[1,3], Eric Halgren[2] & Michael G. Dyer[1]
[1] Artificial Intelligence Lab, CSD, UCLA, Los Angeles, CA 90024
[2] Brain Research Institute, UCLA & VAMC Wadsworth, Los Angeles, CA 90073
[3] CSD, California State University, Fresno, CA 93740

Abstract

In this note we discuss a modification and implementation of a memory model originally proposed by Gardner-Medwin. This is an auto-associative partial matrix model with threshold used as a content-addressable memory. It is of the general type of the models of Willshaw, Kohonen and Anderson. Gardner-Medwin controls the recovery of correct and spurious memories by carefully changing the neural firing threshold. Simulation studies indicated that the original strategy for changing the threshold allowed too many spurious neurons in multiple cycles of recall. We modified the threshold strategy so that the computation is both more effective and more plausible neurally. The memory network can now be used for *recognition* as well as *recall*. We report here on simulation studies of both the original and the modified model. The biological plausibility of our model as well as some implications for the significance of the cognitive evoked potentials N4 and P3 in the storage and retrieval of memories in the human hippocampus are discussed.

Introduction: the Task

Evidence suggests that in humans memory traces for recent memories reside largely within the hippocampal formation (HCF). Important questions in hippocampal neurophysiology are the relationships among memory formation, recognition and recall. Electrophysiological measurements at the scalp show evoked potential components that are correlated with the processing of complex meaningful stimuli. It has been suggested that these components represent a cycle of excitatory / inhibitory modulation of hippocampal interneurons (Halgren et al. 1983). This suggests an associative memory controlled in part by inhibition and for which the recognition of familiarity is part of the recall process.

The Gardner-Medwin Model

The work described in this paper was inspired by an almost forgotten but very enlightening paper by A. R. Gardner-Medwin (1976). This paper presented a mathematical analysis of an auto-associative partial matrix content-addressable memory model. In the Gardner-Medwin model, memory consists of N elements, which may be taken to be simple McCulloch-Pitts neurons. Each neuron receives excitatory input and when it fires it sends out an action potential along an axon (the output). The axon of each neuron gives collaterals which synapse to R of the other neurons. In general, $R << N$. There is also an overall inhibition which sets the firing threshold. The memory can go through several cycles of excitation and firing. Each of the synapses is initially *ineffective* (zero weight). An *event* (a particular pattern of activation) is *learned* by making any synapse between two neurons in the event *effective* (weight one). In a recall task a pattern of firing on the input, which can be considered as a *cue* into the memory, is given to the network and in one or more cycles of excitation and firing, a set of the neurons will be sending output. A general problem with a partially connected network is setting the firing threshold for recall. Gardner-Medwin proposes a method for starting the threshold low and increasing it in later cycles based on computing Poisson tail probabilities. It turns out that

* This research was supported in part by grants from the Keck Foundation, the ITA Foundation, and by a contract with the JTF program of the DoD, monitored by JPL, and by the USPHS (NS18741) and the Veterans Administration.

some of the simplifications in the mathematical analysis cause the amount of spurious recall to be seriously underestimated.

Comparison with other work

Associative models of memory have been studied in psychology, neuroscience and computer science in many forms (cf. Anderson & Bower, 1973; Hinton & Anderson, 1989). Linear matrix memory models similar to Gardner-Medwin are described by Kohonen (1972, 1977) and Anderson (1970, 1972). Kohonen analyzes the case of a partially connected linear model and Anderson studies the problem of recognition. However, neither of these models includes a firing threshold and so they do not deal with the resulting non-linearity. The present model is closest to that proposed by Willshaw et al. (1969). In both the Willshaw and Gardner-Medwin models, a memory trace is a pattern of on-off neurons. The trace is learned by setting synaptic connections and the recall mechanism is implemented as a cycle of activation followed by firing if the activation is above some threshold. Willshaw considers completely connected networks ($R = N$ or $N - 1$) so that the threshold is always set equal to the size of a learned pattern. In an incompletely connected model ($R < N$) the problem of choosing the threshold is more important.

Simulation studies

For a memory model intended to include recognition as well as recall, the problem of spurious recall is critical. In order to study the Gardner-Medwin model, we implemented it on a Thinking Machines CM-2 (Connection Machine). The memory had 4096 processors representing neurons and each neuron had 256 potential synapses with other (randomly chosen) neurons. Each event was a randomly chosen subset of size 10% of the total memory (about 400 neurons) and the memory first learned 10 such patterns. To test recall, subsets of size 10%, 20%,..., 90% of a particular pattern were given to the memory as cues and several cycles of excitation-firing were run. The percentages of correct and spurious neurons recovered were computed after each cycle. To test recognition, we gave the memory cues from no learned pattern. The threshold calculation proposed by Gardner-Medwin gave too many spurious neurons after a few cycles of recall with the number of spurious neurons larger with larger cues (Figure 1.) so we modified the threshold function to be a sum of two components. First, the threshold should grow proportional to the cue so that for small cues correct neurons can be rapidly recruited but as the cue grows spurious neurons are suppressed. This corresponds to a recurrent component of inhibition (Ir). Second, there is no optimal balance between rapid correct recall and low spurious recall. This parameter should be settable exogenously according to something like level of arousal of the organism. This corresponds to a general inhibition (Ig) such as is often seen coming from the brainstem. With this threshold function, we ran the simulations with a base value for Ig and also for values 50% higher and lower.

Recall

For the base value Ig and given a cue from a known pattern, the network recalled almost all of the pattern after at most three cycles (more for a 10% cue). Note that with only partial connectedness, there is no guarantee that all of an event will be recovered even with the whole event as cue if the threshold is high. But in all these cases 99% of the event is recovered. The number of spurious neurons grew very slowly through all the cycles computed, generally reaching a steady state of about 1-2% of the number of number of correct neurons (Figure 2.).

Recognition

We tested recognition by giving a cue from an unlearned pattern. For the base value of Ig, we found in every case, with cues from 10% to 90% of the size of a pattern that the cue died out completely in two cycles and in only a few cases were there any neurons cued after one cycle. Thus, the memory has a simple way of knowing after one cycle that this cue is not familiar.

Exogenous control of memory

If the general (arousal dependent) inhibition, Ig, is cut in half in recall of a known pattern, correct neurons are recovered much faster with substantially all the pattern recovered after two cycles. However, the number of spurious cells rises much faster than in the previous case, rising to as much as 8% of the correct cells (Figure 3.). If, instead, Ig is increased 50% from its base value, the recruitment of spurious cells is almost totally suppressed. However, the recovery of correct neurons from the pattern is much slower, especially for small cues. For a cue of size 10% of the pattern, the cue drops to zero just as in the case of an unknown event and the memory is unable to recover a pattern (Figure 4.). For an unknown cue and the low Ig the successive cues drop very low after one cycle but then begin recruiting cells rapidly, so that after a few cycles and even with a small cue, more than 10% of all neurons in the memory were involved. Thus the exogenous control can set the memory to a more relaxed or more focused state.

Neural plausibility and implications for hippocampal functioning

A computational study (on a network level) of the basic mechanisms for learning/recall in the hippocampal formation (HCF) led us to the implementation of this model. Numerous studies in physiology and neuropsychology have shown the importance of this brain structure in the formation and retrieval of memories for recent events. The original Gardner-Medwin model and our modifications of it bear a number of structurally and functionally significant similarities to the HCF and also a number of differences (most of them due to oversimplification). One simplification is the hebbian type of learning used in this model which has not been identified in the brain. In the hippocampus synaptic plasticity follows the rules of long term potentiation (McNaughton, 1983) which are similar to the ones postulated by Hebb but with a constant component (Marr, 1971).

An important similarity between our model and the HCF is the extensive and random connectivity between neural elements. Similarity between the HCF and our model however, extends beyond the level of connectivity to some aspects of functionality including the general inhibition and the recurrent feedback inhibition. Another important similarity is the modifiability of the neuronal firing threshold. The existence of modulatory processes which facilitate recall has been recently hypothesized (Halgren & Smith, 1987). Electrophysiological manifestation of these processes can be recorded as event related potentials (ERPs) on the scalp. In general ERPs are generated by quasi-synchronous activation of spatially organized populations of synapses. Two specific ERP components have been correlated with cognitive processing -- the N4 and the P3. They are generated in the HCF during memory formation and retrieval as well as in the association neocortex. Across a large number of psychological tasks, the N4 is correlated with associative activation to complex meaningful (or potentially meaningful) stimuli (words, faces) and the P3 with cognitive closure (Kutas & Hillyard, 1984). The N4/P3 sequence has been suggested to represent a cycle of excitatory/inhibitory modulation of hippocampal interneurons (which in turn inhibit the hippocampal pyramidal cells) (Halgren et al. 1983). A functional analog of these hippocampal processes in our model is the sequence of falling/raising threshold. The network behavior which we observed while modulating the threshold during recall supports the hypotheses that the N4/P3 complex has a physiological significance rather than being simply an epiphenomenon measured on the scalp. In the brain, the most likely sites generating the N4/P3 are cholinergic or monoaminergic synapses which receive input from the brainstem. Thus, the emergent recognition behavior of the modified Gardner-Medwin model discussed here is yet another evidence for its neural plausibility.

Summary

In a partially connected pattern associator, Gardner-Medwin suggested a method for choosing the threshold dynamically to keep down spurious recall. Simulation studies showed that the problem of spurious recall was more serious than originally estimated. We modified the threshold function to make it more effective and neurally plausible and extended the model to include recognition as well as recall. We believe that this model provides a basis for studying the hippocampus.

Figure 1.

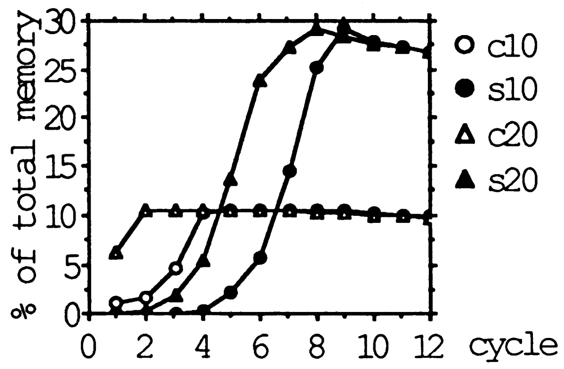


Figure 2.

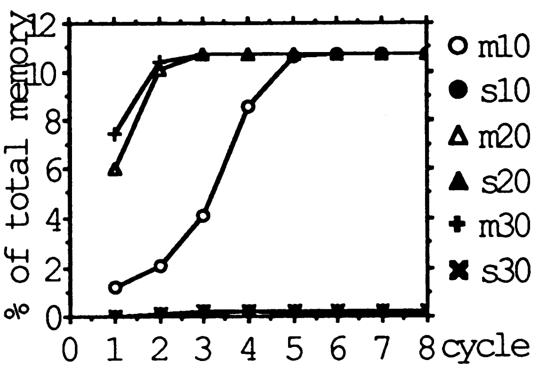


Figure 3.

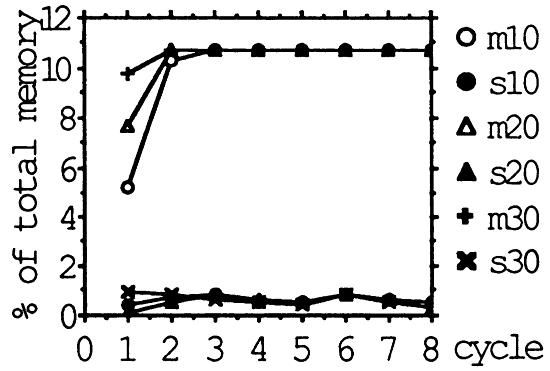
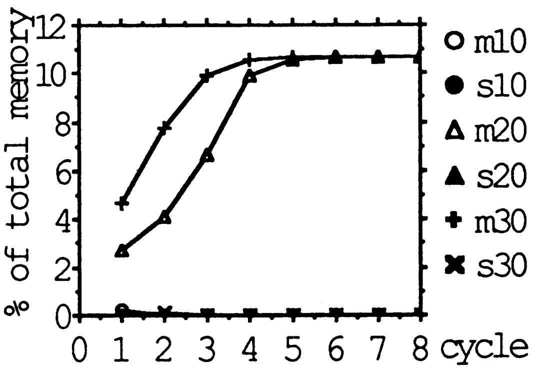


Figure 4.



References

Anderson, J.A. (1970). Two models for memory organization using interacting traces. Math Biosci, 8:137-160.
Anderson, J.A. (1972). A simple neural network generating an interactive memory. Math Biosci, 14:197-220.
Anderson, J.R. & Bower, G.H. (1973). Human associative memory. V.H. Winston, Washington, D.C.
Gardner-Medwin, A.R. (1976). The recall of events through the learning of associations between their parts. Proc. R. Soc. Lond. B., 194:375-402.
Halgren, E. & Smith, M.E. (1987). Cognitive evoked potentials as modulatory processes in human memory formation and retrieval. Human Neurobiology, 6:129-139.
Halgren, E., Wilson, C.L., Squires, N.K., Engel, J.Jr., Walter, R.D. & Crandall, P.H. (1983). Dynamics of the human hippocampal contribution to memory. In: Neurobiology of the hippocampus. (Seifert, W., ed.), Academic, London. pp. 529-572.
Hinton, E. & Anderson, J.A. Eds. (1989). Parallel Models of Associative Memory. Lawrence Erlbaum, Hillsdale, NJ.
Kohonen, T. (1972). Correlation matrix memories. IEEE Transactions on Computers, C-21(4):353-359.
Kohonen, T. (1977). Associative memory: A system-theoretical approach. Springer Verlag, Berlin.
Kutas, M. & Hillyard, S.A. (1984). Brain potentials during reading reflect word expectancy and semantic association. Nature, 307:161-163.
Marr, D. (1971). Simple memory: A theory for archicortex. Phil. Trans. R. Soc. Lond. B, 262:23-81.
McNaughton, B.L. (1983). Activity dependent modulation of hippocampal synaptic efficacy: Some implications for memory processes. In: Neurobiology of the Hippocampus. (Seifert, W., ed.), Academic, New York. pp. 233-249.
Willshaw, D.J., Buneman, O.P. & Longuet-Higgins, H.C. (1969). Non-holographic associative memory. Nature, 222:960-962.

Expertise acquisition through concepts refinement in a self-organizing architecture

Philippe G. Schyns
Dept. of Cognitive and Linguistic Sciences, Brown University, Providence, RI 02912, USA

1.Introduction.

People often differ dramatically in the knowledge they have about the objects of a given domain in the world. Although we know that expertise develops through the repeated interaction with a specific part of the environment, how the differences between an expert and a novice reflect in their conceptual system is still an important issue in Cognitive Science. The purpose of this work is to show how a simple self-organizing architecture can give an account of expertise acquisition through the refinement of a conceptual *representation*.

How the conceptual system of experts and novices differ has been given a careful theoretical investigation in Murphy and Wright (1985). Among the possible accounts for the conceptual organization of experts they consider, two are of primary importance for our work, they will be briefly described hereafter.

The first theoretical hypotheses states that the difference between the conceptual system of an expert and a novice might just be a quantitative one. As experts are able to make finer segmentations between the objects of their domain of expertise, they must have a more specific representation of the categories composing the domain.These more finer representations would be made out of a larger number of low-level concepts.

The second hypotheses stresses a qualitative difference in the categorical representation of experts and novices: Experts might have more differentiated concepts than novices. Experts might perceive more similarity (vs. contrast) for different exemplars of the same (vs. different) category than novices would do. So the categorical judgments made by experts might be more accurate, with less inter-category overlap, than those made by novices.

Of course, as Murphy and Wright (1985) point out, "... the first two possibilities need not be mutually exclusive. Experts may have both more differentiated ... and more specific categories than novices."

2. Kohonen's self-organizing architecture.

In this work, to model concept acquisition, we have used a variant of the self-organizing procedure described in Kohonen (1982, 1984). The prototypical Kohonen architecture is composed of a n-dimensional input vector fully connected to a two dimensional output map. This architecture is linked to a discriminant function, an adaptation rule, and a neighborhood topology that shrinks with time. The overall result of the system is to achieve a vector quantization of the input space according to a Voronoï tessellation (Kohonen, 1988). We have slightly modified the standard architecture to make it less unbiological, and the results obtained are, in general, similar to Kohonen's although less robust. Our architecture and its appendices will be summarized in the following paragraphs.

2.a. discriminant function.

$$o_o(t) = \max_i (m_i^T(t)\, i(t))$$

At time t, select the output unit *o* for which the inner-product of its weights and the input vector is the maximum.

2.b. neighborhood topology and updating rule.

Kohonen has suggested to update the weight vector afferent to the winning units as well as the members of a certain neighborhood topology. In order to achieve a convergence of the ordering of the map, the neighborhood size has to shrink with time. Here, we propose an implementation of this constraint that might have some biological flavor: Each output unit has local, fixed, excitatory connections with its neighbors. The local connection strength between two output units, lc(x,y), is given by a Gaussian function of the Euclidian distance between a particular output unit and its neighbors. If the distance exceeds a given value, the connection is non-existent.

With these little modifications, the learning rule becomes

$$w_i(t+1) = w_i(t) + i(t) * [\,(1 - |o_i(t)|)) * lc(o_i, o_w)\,] \text{ for } i \text{ in } N_o$$
$$w_i(t+1) = w_i(t) \text{ for } i \text{ not in } N_o$$

The weights afferent to output unit *i* will be added a fraction of the input i at time t. This fraction will be computed by multiplying the inverse of the activation of the absolute value of the output unit o_i and the value of the local connection between the considered output unit and the winning output unit, lc (o_i, o_w).If there is no local connection between the winning unit and the considered output unit, the weight vector remains unchanged.

With this scheme, when the activation of the winning output unit will tend to its maximum, the modification of the weights will tend to 0. For this reason, and the gaussian local connections, we neither need a gain parameter nor a topology structure that reduce with time. The learning rule take them implicitly into account. By so doing, we can achieve a topological ordering of the map from global to local as in Kohonen (1982, 1984, 1988).

3. The experiment.

3.1. Parameters of the architecture: In this experiment, the input vector had dimensionality 100 and was fully connected to a two dimensional map of 10 x 10 output units. The maximal distance for the local connection was 2.83, and the sigma of the gaussian function had been set to 1.5.

3.2. Stimuli: We constructed a taxonomy of four different categories each composed of three subcategories. This make a total of twelve prototypical stimuli. Each element of the input vector encodes the presence or absence of a particular component of meaning by being respectively "on" or "off" -- 1 or -1 --. In order order to be able to interpret the results later on, we have represented the prototypes as simple drawings on a 10 x 10 array (cfr. figure 1).

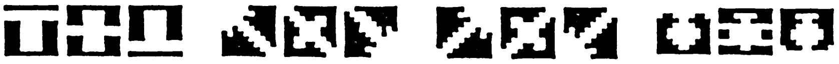


Figure 1: Prototypes from category 0, category 1, category 2, category 3.

At each iteration of learning, the network was fed with a *normalized* distortion from one prototype. These distortions were made out of noisy features added to the initial prototype. The noisy features could be located anywhere on the contour of the drawing, and their size was given by a random value ranging from 1 to 10. In order to prevent the prototype to be composed of necessary and sufficient features, each of the units with a white, "on", value was effectively turned on with a probability of 0.75. This last constraint should prevent the exemplars of being defined by singly necessary, and jointly sufficient, features.

We assigned a different probability to each category -- respectively 0.65, 0.2, 0.1, 0.05 for category 0, 1, 2, 3 --. Underlying this distribution of probability is the reasonable hypothesis that the most (vs. least) frequent categories should the ones for which the network should built the most (vs. least) refined categorical representations. Once a category was chosen according to this distribution, a subcategory was evenly chosen and the exemplar was computed from the prototype of this subcategory.

4. Results.

Previous simulations related to the organization of conceptual maps have given the following qualitative results. From now on, we will consider the activation states of the map as the codes built by the system to represent the category input space on a two dimensional representational space.

4.a. Global organization into conceptual regions: With learning, the output map gets organized into contrasted regions that respond mostly to exemplars drawn from particular categories. When one region is activated by the presentation of an exemplar from a particular category, the activation of the other conceptual regions is a function of the similarity (measured by the vector_cosine) between the exemplar and the other categories. The topographical proximity of categories on the map encodes their similarity.

4.b. Local organization inside a conceptual region: In a conceptual region, the organization follows the same principle than the previous ones but one level down: in each conceptual region different patterns of activation can be observed depending on the subcategory the exemplar is coming from. When a particular subregion is mostly activated, the activation of the other subregions is a function of the vector cosine between the presented exemplar and the prototype of the other subcategories.

4.c. Prototypical organization inside the conceptual subregions: Each of the subregion is organized according to the same principle. One unit, usually around the center of the subregion, will respond mostly to the prototype of the concerned subcategory. The other units composing the subregion will respond mostly to specific distortions of the prototype according to the following general rule: *the less similar the exemplar is from the prototype, the further away from the unit that respond maximally to the exemplar will be the unit that responds maximally to the prototype.*

4.d. The organization of the map proceeds from global to local;The categories are represented before the subcategories. This is due to the global to local organization of the learning rule.

To summarize, going from global to local, the map is organized first into different, contrasted, conceptual regions. Then, inside each region, a differentiated pattern of activation can be observed for each subcategory of the category considered. Each of these differentiated patterns of activation will be centered on a unit. The distance of this unit from the center of the subregion will be an inverse function of the similarity of the exemplar presented with the prototype of the subcategory considered. These results hold for all categories of an equiprobable distribution.

The activation of each output unit is a function of its afferent weights and the sample exemplar. So, the knowledge of category organization must be included in the weights underlying the output units. They have to pick out the general features that are characteristics of the categories and the specific features that characterize the subcategories. To see this, after the network had been presented different numbers of patterns, we saved the values of the weights of each output units. The learning rule proceeds so that the weight vector of the winning unit and its neighbors rotate in direction of the presented input. We should be able to have a clear account of the filtering done by the weights on the inputs. Since each subcategory is characterized by having some particular features, we should be able to see if some weight vectors have been able to pick them out, and if so, with what strength.
The next figure shows the weights afferent to each output unit after 0, 10 , 50, 1000 iterations of learning. The connection strength is coded with levels of grey: black and white mean respectively maximally inhibitory and maximally excitatory connection.

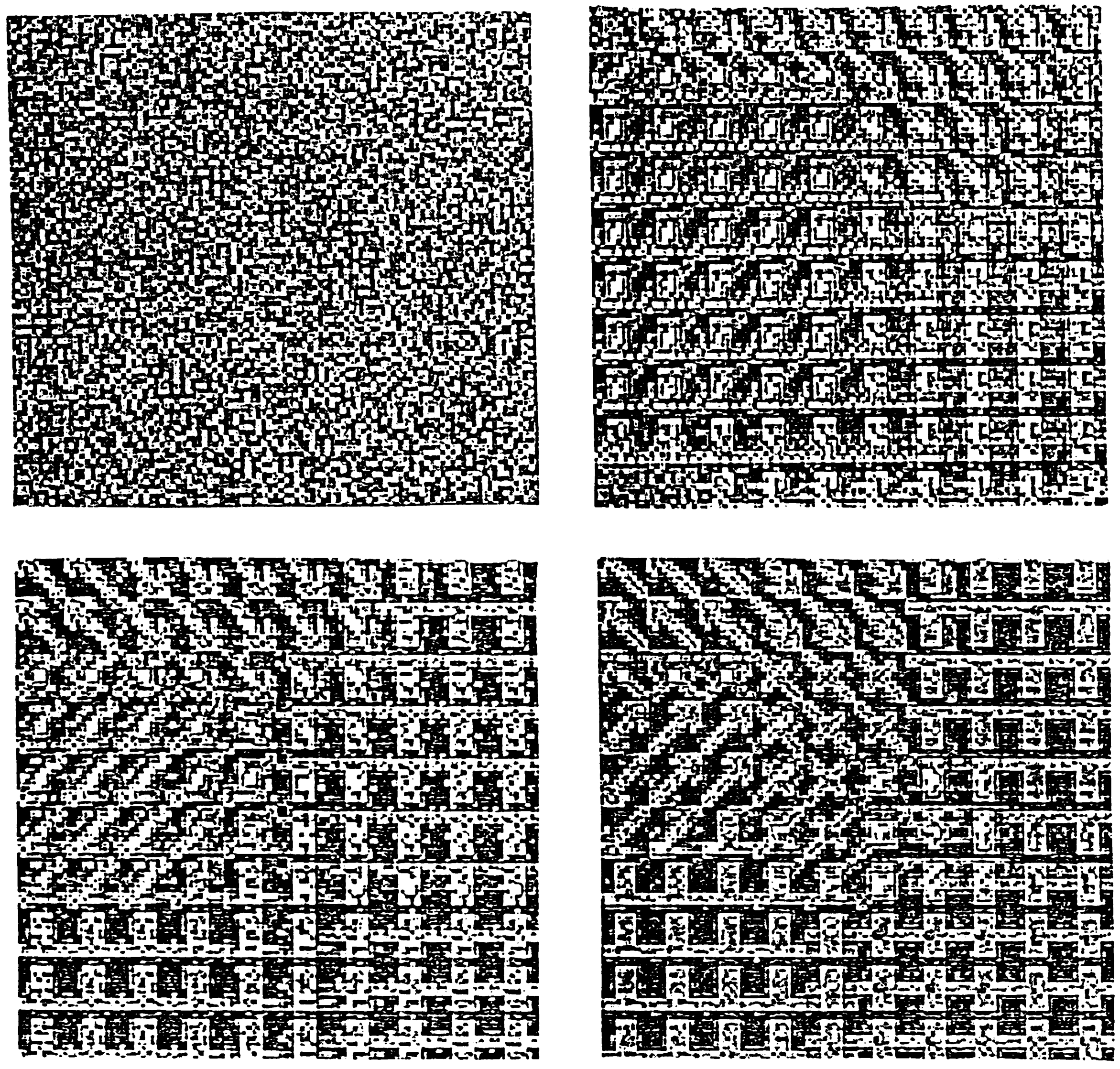


Figure 2: The weights afferent to each output unit after 0, 10, 50, 1000 iterations

We can observe from the figure why the organization of the output map proceeds from global to local: The features common to all members of a given category are first picked out by the weights. Then, as time goes on, more specific features, characteristics of the subcategories, are picked out and the subcategories get organized. At that time, the system should be able to make much finer segmentations among the exemplars of the domain it is confronted to. It

should also be observed that with time the relevant features get more and more distinct, white or black. The irrelevant features, with respect to the underlying structure of categories, are simply not picked out.
To synthesize these first observations, we can say that the weight vectors notice more and more of the features that characterize a category. As a result, the output developed on the map should get more and more contrasted, allowing another layer, added on top of the current architecture, to name these representations more accurately and more easily. These observations are true only for the most frequent categories. What about the others?
The learning rule has the interesting side effect that the representation of each category on the map is attributed a number of output units that is a function of the distribution of probability of the categories. This can be observed on the previous figure, where category 3 has been allocated a smaller number of output units than category 0 for example. Now, the bigger (vs. smaller) the number of units dedicated to the representation of a category, the more (vs. the less) the underlying weights will be able to pick out the relevant features that discriminate among the members of the category. On the previous figure, we can see that, for category 3, the weights have not been able to pick out any of the features characteristics of the subcategories, while for category 2, the weights have picked out two out the three subcategories. So, the map won't be able to represent the different subcategories of category 3 by differentiated output patterns. Each subcategory will have a similar output pattern. This is of course contrasted with the output map obtained for category 0 where each subcategory is clearly differentiated (cfr. figure 3)

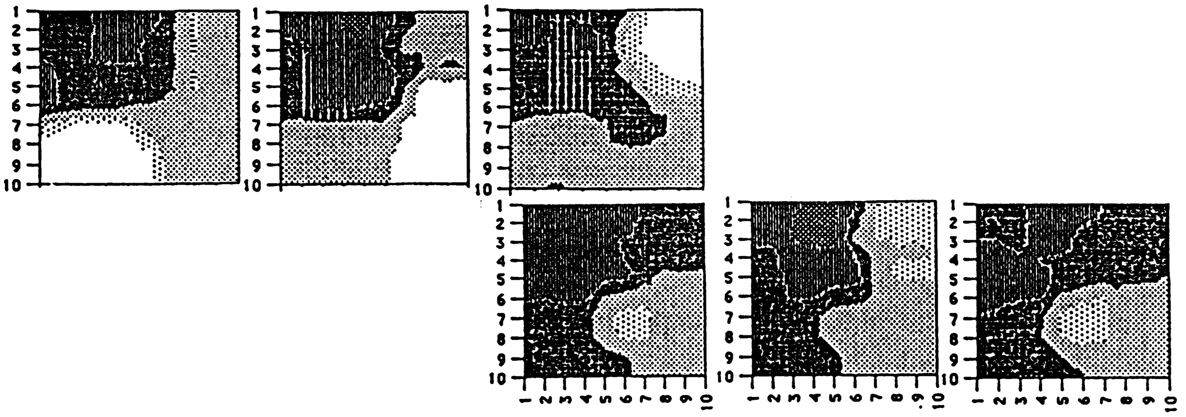


Figure 3: output on the map for exemplars from the three subcategories of category 0 (left) and 3 (right).

5. Conclusion

In this short paper, we have shown that a self-organized network could display various degrees of expertise in different conceptual domains. We have shown that the characteristics of this expertise acquisition reflect in the hierarchical conceptual organization of a system both as a quantitative and a qualitative difference. Here, the quantitative difference is due to the extraction of relevant features by the network that enable the system to have more concepts, thereby allowing finer discriminations in the domain of expertise. To the contrary, such discriminations are not possible for the categories the system is a novice of. The qualitative difference in the conceptual representation is related to the contrast between the features in the weights: the more the features are contrasted, the better will be the contrast of the conceptual representation on the output map.
So far, our work has concentrated on a suitable self-organized *representation* for categories. Others have worked in concept learning using different architectures to *name the representations of categories* (Knapp & Anderson, 1984; McClelland & Rumelhart, 1985; Anderson & Murphy, 1986). In the future, we will add another layer to the architecture and model some psychological effects related to the acquisition of a lexicon.

6. References

Anderson, J., A., Murphy, G. L. (1986). Psychological concepts in a parallel system. *Physica*, 22D, 318-336.
Knapp, A., G., Anderson, J. A. (1984). Theory of categorization based on distributed memory storage. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 10, 616-637.
Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43, 59-69.
Kohonen, T. (1984). Self-organization and associative memory. Berlin: Springer.
Kohonen, T. (1988). The neural phonetic typewriter. *IEEE Computer*, March, 11-22.
McClelland, J., L., Rumelhart, D. E. (1985). Distributed memory and the representation of general and specific information. *Journal of Experimental Psychology: General*, 114, 159-188.
Murphy, G., L., Wright, J. C. (1984). Changes in conceptual structure with expertise: differences between real-world experts and novices. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 10, 144-145.

POSSIBLE MECHANISMS OF EXPERIENCE-DEPENDENT SYNAPSE MODIFICATION IN THE VISUAL CORTEX

Mark F. Bear
Center for Neural Science
Brown University
Providence, Rhode Island USA

The cat visual cortex has proved to be a valuable experimental model for the purpose of determining how neural networks modify according to experience. Neurons in the primary visual cortex, area 17, of normal adult cats are sharply tuned for the orientation of an elongated slit of light and most are activated by stimulation of either eye. Both of these properties -- orientation selectivity and binocularity -- depend on the type of visual environment experienced during a critical period of early postnatal development (reviewed by Frégnac and Imbert, Physiol. Rev. 64: 325; 1984). For example, monocular deprivation (MD) during the critical period [extending from approximately 3 weeks to 3 months of age in the cat] has profound and reproducible effects on the functional connectivity of striate cortex. Brief periods of MD will result in a dramatic shift in the ocular dominance (OD) of cortical neurons such that most will be responsive exclusively to the open eye.

The consequences of binocular deprivation (BD) on visual cortex stand in striking contrast to those observed after MD. While 7 days of MD during the second postnatal month leave few neurons in striate cortex responsive to stimulation of the deprived eye, most cells remain responsive to visual stimulation through either eye after a comparable period of BD. Thus, it is not merely the absence of patterned activity in the deprived geniculocortical projection that causes the decrease in synaptic efficacy after MD.

Gunther Stent, in an influential 1973 paper (Proc. Natl. Acad Sci. USA 70: 997), suggested that the crucial difference between MD and BD is that only in the former case are cortical neurons active. According to this "learning rule", postsynaptic activation is necessary for all synaptic modifications and the sign of the change (+ or -) is dependent on the concurrent level of presynaptic activity. However, subsequent work suggested that the generation of action potentials in a cortical neuron does not ensure that ocular dominance modifications will occur after MD. To reconcile these data with the Stent model, Singer (In FO Schmitt and FG Worden (eds) "The Neurosciences Fourth Study Program," Cambridge, MA: MIT Press, pp 1093-1109; 1979) introduced the idea that there is a critical level of postsynaptic activition that must be reached before experience-dependent modifications will occur, and that this threshold is higher than the depolarization required for somatic sodium-spikes.

This hypothesis was recently tested in a study by Reiter and Stryker (Proc. Natl. Acad Sci. 1988). They infused continuously the GABA$_A$ receptor agonist muscimol into striate cortex as kittens were monocularly deprived for 7 days. With the muscimol still present in cortex, they mapped the cortex to determine the extent of activity blockade. They found that all cortical cell responses were eliminated within several millimiters of the infusion cannula, even though LGN fiber activity was readily demonstrated. When the muscimol wore off, they performed an ocular dominance assay in the zone of cortex whose activity had been blocked. They observed an unexpected ocular dominance shift toward the *deprived eye*; that is, most neurons were no longer responsive to stimulation of the retina that had been more active during the period of MD. These data suggest that patterned presynaptic activity can lead to either an increase or a decrease in synaptic strength, depending on whether or not the target neurons are allowed to respond.

The results of this study are inconsistent with the Stent-Singer model because synaptic modifications were observed in the absence of demonstrable postsynaptic activity. However, they are compatible with an alternative theoretical solution to the problem of visual cortical plasticity, developed by Leon Cooper and his associates at Brown University (reviewed by Bear, *et al.*, Science 237: 42; 1987). According to this theory, the efficacy of active synapses increases when the postsynaptic target is concurrently depolarized beyond a "modification threshold", θ. However, when the level of postsynaptic activity falls below θ, then the strength of active synapses decreases. Thus, "effective" synapses are strengthened and "ineffective" synapses are weakened, where synaptic effectiveness is determined by whether or not the presynaptic pattern of activity is accompanied by the simultaneous depolarization of the target dendrite beyond the modification threshold, θ.

An important feature of this model is that the value of the modification threshold is not fixed, but instead varies as a non-linear function of the average output of the postsynaptic neuron [the time over which postsynaptic activity is averaged can be inferred to be in the range of several hours]. This feature provides the stability properties of the model, and is necessary to explain why the low level of postsynaptic activity caused by binocular deprivation does not drive the strengths of all cortical synapses to zero.

Analysis and computer simulation have shown that virtually all of the available data on the experience-dependent modification of striate cortex can be explained by this simple learning algorithm (Bienenstock, *et al.*, J. Neurosci 2: 32; 1982; Clothiaux, Bear and Cooper, unpublished). While this is satisfying in its own right, the critical question remains as to whether this form of synaptic modification has a neurobiological basis. Recall the following three distinctive features of the theory.

(1) The sign of the synaptic modification depends on whether the postsynaptic depolarization is greater or less than θ.

(2) The rate of the synaptic modification depends on the concurrent level of presynaptic input activity.

(3) The value of θ varies with the average activity of the neuron.

This leads to the following 3 questions that we have begun to address experimentally.

(1) When input activity is high, what distinquishes postsynaptic depolarizations greater than θ from those less than θ?

(2) When postsynaptic depolarization is less than θ, what distinquishes high from low input activity?

(3) What is the molecular basis of the sliding modification threshold?

My lecture constitutes a status report on our efforts to answer these questions.

CHAOS IN THE BIODYNAMICS OF PATTERN RECOGNITION BY NEURAL NETWORKS

Walter J. Freeman and Yong Yao
Department of Cell & Molecular Biology, University of California at Berkeley, CA 94720

Abstract: Through an appropriate choice of the values of its parameters, a distributed olfactory model maintains a low dimensional global chaotic attractor with multiple "wings". The central part of the attractor is its basal chaotic activity, which simulates the EEG activity of the olfactory system under exhalation. Each of the wings may be either a near-limit cycle or a broad band chaos, which simulates the EEG activity under inhalation. The reproducible spatial pattern of each near-limit cycle is determined by a template made in the system. A novel input with no template activates the system to activity without a reproducible spatial pattern of either near-limit cycle wing or a broad band chaotic wing. Pattern recognition in the system may be considered as the transition from the basal state to a wing or from one wing to another, as demonstrated by computer simulation. The computer simulation also shows that the adaptive behavior of the system is scaling invariant, and it is independent of the initial conditions at the transition from one wing to another. These properties enable the system to classify an uninterrupted sequence of contiguous stimuli at an interval of 200 milliseconds per each stimulus. Using this chaotic system, we have obtained decent results for classifying sets of industrial data, which are not solved as well by other methods tried.

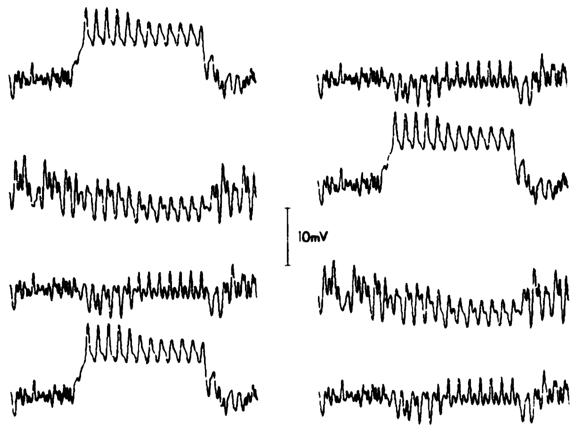
1. INTRODUCTION

Physiological studies have been made of the dynamics of the olfactory system during learning by animals to discriminate odors under reinforcement. The neural activity has been recorded in the form of electroencephalograms (EEG). For each odor that is discriminated a characteristic spatial pattern has been found when that odor is presented, which can serve to classify that EEG segment. The time series of the EEG shows a brief episode of high amplitude oscillatory activity called a "burst", during the inhalation by which the odor is taken in. It has a degree of regularity that might suggest convergence to a near-limit cycle attractor, if the odor is discriminated by virtue of prior training. Otherwise the burst is aperiodic. So also is the time series of the EEG between the inhalations and at rest, which is called the basal state. This basal state appears to be chaotic, because the digitized EEG has a Gaussian amplitude histogram, an autocorrelation that rapidly approaches zero, and a broad power spectrum. The widespread spatial coherence shows that this activity manifests a chaotic attractor in the nonlinear cooperative dynamics of the neural system [1][2].

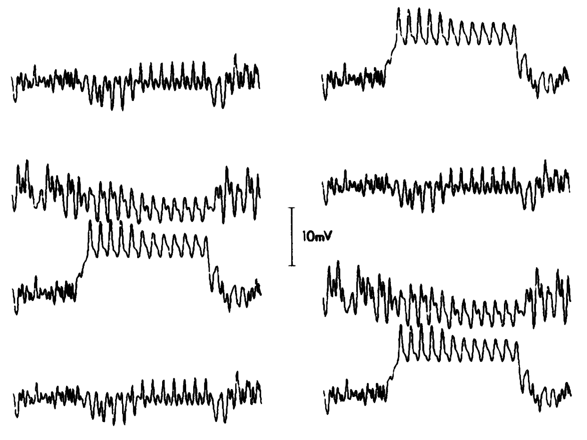
This report describes computer modeling of the olfactory system and aims to determine the role of chaos in the process of pattern recognition. The model is distributed, and it is described by sets of coupled nonlinear differential equations [3]-[4]. It is a natural extension of the lumped olfactory model [5] and a distributed bulb model [6]. The lumped olfactory model showed how to generate a EEG-like chaos; the distributed bulb model showed how to classify patterns with Hopf bifurcation. The layers of the system are simulated with distributed arrays of elements, the correlation rule is applied, and the long connections with delay are structured so as to yield chaotic solutions in the steady state and in bursts caused by inputs on simulated inhalations. The connections and boundary conditions are modeled on the functional anatomy. The values of parameters in the model are identified by computer experiments under the insights from neurobiology.

2. AN ASSOCIATIVE MEMORY NETWORK

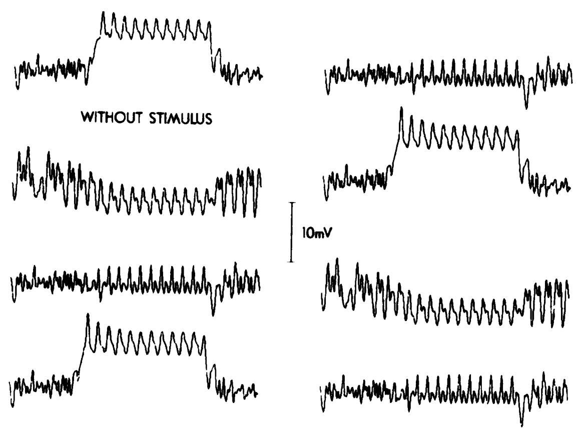
Our computer simulation shows that the distributed olfactory model acts like an associative memory (see Fig.1). The difference between the associative memory studied here and others in the literature is that the former is of chaotic dynamics, which is much closer to the functions we found in the biological systems.



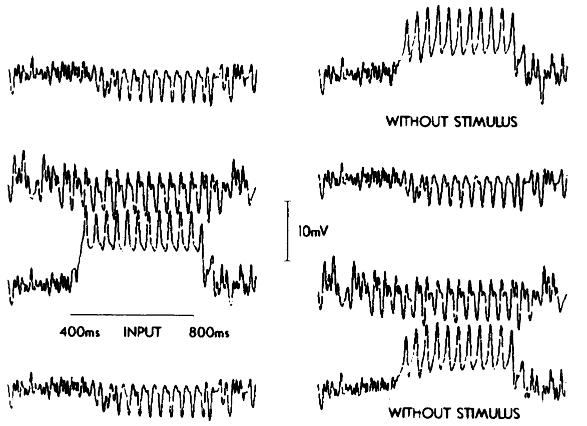
a) The spatial amplitude pattern corresponding to the input Patt A =(1 0 0 1 0 0 1 0), i.e. the input is given to all the template Patt A channels;



b) The spatial amplitude pattern corresponding to the input Patt B =(0 1 0 0 1 0 0 1). If the trace with a high DC offset is considered "1", otherwise "0", the array of the traces in Fig.1a gives rise to the vector (1 0 0 1 0 0 1 0), i.e. Patt A. The array of the traces in Fig.1b is Patt B.



c) Patt A is retrieved by the input (0 0 0 1 0 0 1 0),



d) Patt B is retrieved by the input (0 0 0 0 1 0 0 0).

Fig.1. The output waveforms from the 8 channel array of G-cells. The stimulus inputs are added in the period 400ms to 800ms. Different inputs give rise to different spatial patterns. The "on" channels have high DC offsets, while the "off" channels have low DC offsets. By retrieval it means that the Fig.1a and Fig.1c, and Fig.1b and Fig.1d have the same high DC offset channels and the same low DC offset channels, respectively.

3. INDEPENDENCE OF INITIAL STATES

An artificial system can be returned repeatedly to a particular initial condition. However, this is nonsense for a living animal. It is found that the convergence of the olfactory model is independent of its initial conditions. The system comes back to its basal state whenever the input is terminated. The inputs drive the system back and forward among the wings and the basal state.

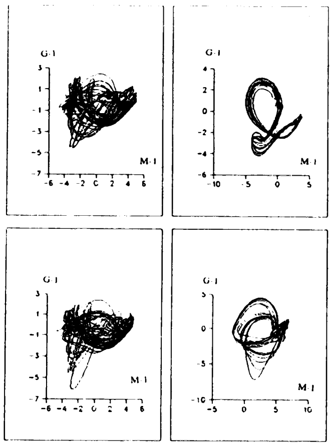


Fig.2. Each of the pictures is the phase portrait of a "mitral cell" against a "granule cell". It is composed of 2000 points and its time duration is 1000ms. The top left plot (from 500ms to 1500ms) and the bottom left plot (from 2500ms to 3500ms) appear like each other, which corresponds to the phase portrait of the basal state. The top right (from 1500ms to 2500ms) is the phase portrait of Patt A, while the bottom right (from 3500ms to 4500ms) is the phase portrait of Patt B. Patt A and Patt B are defined in Fig.1. Patt A is presented during 1500ms to 2500ms, and Patt B is presented during 3500ms to 4500ms.

4. SCALING INVARIANCE

Fig.3 shows the scaling invariant property of the distributed olfactory model. This is crucially important for practical applications and for the plausibility of the model. As we can see, the system jumps to a high-level chaotic activity from its low chaotic basal activity when a stimulus is given, and it returns to the basal activity when the stimulus is terminated whatever the number of its channels.

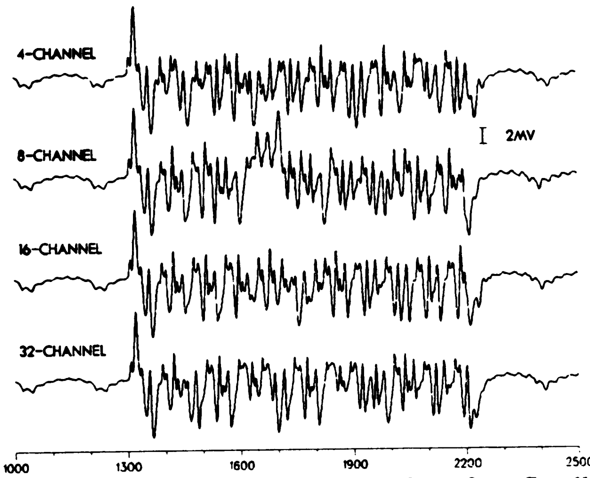


Fig.3. The first trace is from one of the four G-cells in a 4 channel case; the others are 8 channel, 16 channel, and 32 channel cases. The system dimensions of these cases are 58, 98, 178, and 338, respectively. Here there is no template built into the system. However, this scaling invariant property also holds for the system with templates.

5. FRACTAL AND 1/f NOISE

Physiological experiments show that the spectra of brain waves are broad with low and variable peaks consistent with 1/f noise [1], and the correlation dimension of the waves is fractal [7].

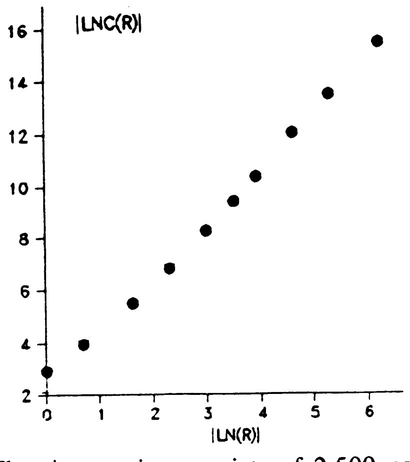
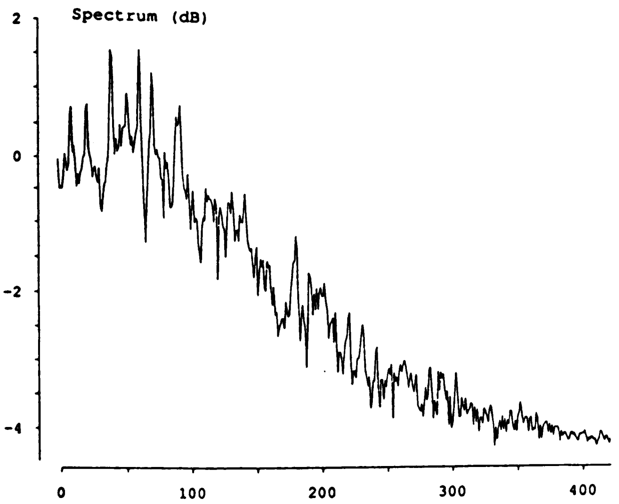


Fig.4. The time series consists of 2,500 points sampling from the basal activity in Fig.1. a) The slope of the linear region is the measurement of the correlation dimension of the time series.



b) The power spectrum of the time series fluctuates on a straight line, which implies a "1/f noise".

The spectrum analysis and the correlation analysis provide similar results with respect to the time series of the model (see Fig.4).

6. INDUSTRIAL DATA RECOGNITION

Let us consider an industrial data recognition problem [8]. There are 20 phase vectors in 64 dimensional feature space corresponding to 20 screw-driver heads. The classification results by using a minimum distance classifier is listed in Table I.

Apex: No involvement of the models with the geometrical method

input #	class #	D_bad	D_good	labeled #
1	0	8.32	28.76	0
2	0	13.12	13.56	0
3	0	6.92	29.76	0
4	0	10.92	27.76	0
5	0	11.12	15.56	0
6	0	19.92	13.56	1
7	0	9.52	27.96	0
8	0	7.12	29.96	0
9	0	14.72	18.36	0
10	0	21.72	11.76	1
11	1	27.72	6.16	1
12	1	25.52	6.36	1
13	1	29.12	5.16	1
14	1	24.92	7.36	1
15	1	8.92	26.16	0
16	1	8.52	28.56	0
17	1	22.32	9.16	1
18	1	13.32	25.36	0
19	1	8.92	28.96	0
20	1	15.32	23.36	0

D=3.61 SD_bad=0.70 SD_good=1.36
80% for the bad, 50% for the good

The results are substantially improved after the feature enhancement with the olfactory model (see Table II).

Apex: Using the olfactory model with the geometrical method				
input #	class #	D_bad	D_good	labeled #
1	0	6.35	628.55	0
2	0	67.48	389.19	0
3	0	6.88	622.57	0
4	0	7.14	621.40	0
5	0	2.72	578.83	0
6	0	52.21	401.00	0
7	0	11.55	670.96	0
8	0	5.38	617.93	0
9	0	6.77	618.99	0
10	0	7.72	583.80	0
11	1	68.02	299.62	0
12	1	75.33	285.80	0
13	1	1180.63	124.51	1
14	1	1356.28	194.72	1
15	1	1075.30	100.92	1
16	1	1015.71	90.85	1
17	1	1559.51	280.66	1
18	1	1091.19	107.37	1
19	1	1013.78	91.43	1
20	1	1021.81	93.64	1
D=23.45 SD_bad=2.23 SD_good=3.34 100% for the bad, 80% for the good				

7. CONCLUSION

Our computer simulation results provide support for the following properties of the distributed olfactory model. First, with the change of its "adaptive" parameters, the model can store different patterns. These patterns are retrievable by using partial information. Second, there is a global chaotic attractor in the system. Its stability is shown by the reproducibility of its geometric form in state space. The attractor may be described as composed of a central part and multiple wings. The transition back and forth between the wings and the central part stands for phase transition in the sense of physics and for pattern recognition in the sense of neural networks. Third its scaling is an invariant property, which implies the plausibility of the model. A fourth is the independence of initial conditions, which implies that each wing has its own basin. This may make it possible to do successive pattern recognition as well as pattern completion. A fifth is that the olfactory system reveals a spatial coherence across the array, which is expressed in the reduced dimension of model states, i.e. the dimensions of the model activities are much lower than the number of the related elements. The dimensions are fractal and may vary with the complexity of on-going behavior. In the above five respects the model simulates the corresponding properties of the olfactory system. In two important respects the model fails to do so. First, in the model the response component that best serves to describe the convergence to a reproducible spatial pattern is the baseline shift, whereas in bulb's output it is the amplitude of the burst oscillation and not the baseline shift. Second, whereas in the bulb the transition time required to go from an interburst state to a burst state is only a few milliseconds, in the model the transition often required up to 50 msec.

REFERENCE

[1] Freeman, W.J. (1975), Mass action in the nervous system. *New York, Academic Press.*

[2] Freeman, W.J. (1987), Analytic techniques used in the search for the physiological basis of the EEG. *in "Methods of analysis of brain electrical and magnetic signals, EEG Handbook", A.S. Gevins and A. Remond (EDS), Elsevier Science Publishers.*

[3] Yao, Y. and Freeman, W.J. (1989), Model of biological pattern recognition with spatially chaotic dynamics. *Neural Networks in press.*

[4] Yao, Y. and Freeman, W.J. (1989) Pattern recognition in olfactory systems: modeling and simulation. *An oral presentation in the 1989 Int. Joint Conf. on Neural Networks, Washington D.C. (see the Proceeding of IJCNN89).*

[5] Freeman, W.J. (1987), Simulation of chaotic EEG patterns with a dynamic model of the olfactory system, *Biological Cybernetics 56: 139-150.*

[6] Freeman, W.J., Yao, Y. and Burke, B. (1988), "Central pattern generating and recognizing in olfactory bulb: a correlation learning rule" *Neural Networks 1: 277-288.*

[7] Freeman, W.J. (1988) "Strange attractors that govern mammalian brain dynamics shown by trajectories of electroencephalographic (EEG) potential", *IEEE Trans. CAS-35, 7:781-783.*

[8] Yao, Y., Freeman, W.J., Burke, B. and Yang, Q. (1989), Pattern recognition in a layer-distributed neural network: an industrial application. *Submitted to Neural Networks.*

Acknowledgement: Support from grants AFOSR-87-0317, MH06686 from NIMH and Cray Sponsored University Research and Development Grant Program is gratefully acknowledged. The authors wish to thank Dr. Shawn Buckley from Cochlea Corporation in San Jose, California for kindly providing the industrial data.

Feature Linking by Synchronization in a Two Dimensional Network

G. Hartmann, S. Drüe *)
Fachbereich Elektrotechnik, Universität Paderborn
Pohlweg 47-49, D4790 Paderborn

Abstract

By our simulations we can show, that large numbers of neurons can be synchronized by a fully distributed mechanism. We used a network with exclusively local interconnections. A two dimensional example shows, that an exploding number of meaningful combinations between neurons can be linked by a constant number of interconnections per neuron. There are first hints, that this interconnections may be learnt.

Introduction

Models for feature linking by temporal codes have been proposed by several groups [1], [2], [3], and neurophysiological experiments [3], [4], [5] are supporting this concept. Synchronisation between neurones, representing similar or matching features, provides a powerful tool for neuronal information processing. This mechanism is not restricted to processing of continuous lines as in our example.

The Synchronization Mechanism

Our model neurons are of the well known type, described by French and Stein [6]. In detail, the incoming spike signals are multiplied by a synaptic weight and temporally integrated by a leaky integrator. The signals of all the integrators are summed up to yield the membrane potential, which is compared with a dynamic threshold. As soon as the membrane potential exceeds the threshold, a spike is generated, and the threshold is increased by a fixed step. The threshold decayes exponentially to its normal value within a refractory period.
To explain the synchronizing mechanism, we restrict our two-dimensional network to a chain for a moment (fig. 1). The neurones are driven by afferent signals at feeding inputs. The synaptic weights at these inputs are relatively small and the time constants are relatively long. As a result, a simple spike will not cause a major change of the membran potential and activation will result from the temporal integral over many spikes. In addition to the afferent signals, each neuron receives signals from its next neighbours (fig. 1) at trigger inputs. The synaptic weights at these inputs are higher and a single spike may increase the membrane potential significantly. So a neuron may be stimulated by a single spike at a trigger input if its membrane potential is close to the threshold. In other words, a neuron can only be triggered, if it receives signals also at its feeding inputs. A neuron, however, will not be activated only by triggering-signals independend of the rate. This is due to a very short time constant at the triggering inputs, preventing significant temporal integration.

*) This work was supported by a BMFT-Grant.

Now we can try to understand the behaviour of the simple network in fig. 1. Suppose, all neurons of the chain receive feeding input and neuron 1 happens to be active first. It will trigger neuron 2, and after a short delay of 1ms, neuron 2 will send a spike to neuron 3 etc. This situation is explained by fig. 2a and one can easily see the cumulation of delays. The total delay may be reduced, if a neuron in the middle of the chain happens to fire first, and if neighbours are triggered at both sides (fig. 2b). But these simple examples do not take into account the stochastic nature of the system. A neuron may fire input-driven, it may fire due to stimulation by its neigbour, or it may be in its absolute refractory period during stimulation. In fig. 2c, neurons 3, 6 and 10 are firing input driven, all the other neurones are firing due to stimulation. Neuron 3 triggers its neighbours 2 and 4, neuron 2 triggers neuron 1, and neuron 4 would like to trigger its neighbour 5. But neuron 5 has been triggered just before by 6 and so it is in its absolute refractory period. Simularly, a neuron can never be re-triggered by that neighbour, which was triggered by it before.
In a chain with n neurons, i neurons will fire input-driven and s=n-i will fire due to stimulation. From each of the i input driven neurons, two wave fronts of triggering signals are starting, travelling up and down the chain. A wave front stops, as soon as it collides with another front, travelling in the opposite direction (fig. 2c). The mean number of neurons, triggered by one wave front is n/2i, which is obviously independent of the number n of neurons in the chain. This number n/2i multiplied by the delay between triggering signal and output spike, is a good measure for the time interval, within which all neurons of a chain are firing.
We adjusted the parameters of our model neurons to values compatible with biological neurons. The total input rate, summed up over all feeding inputs of one neuron was 200 spikes/second. The increase of membrane potential was adjusted to 4 mV per spike at a feeding input and 10 mV per spike at a trigger input. The time constant was 30 ms at feeding inputs and 2 ms at trigger inputs. The delay time between a triggering spike and a stimulated spike was adjusted to 1ms. With these parameters, we immediately achieved good synchronization (fig. 3). With different parameter settings we could show, that the synchronization effect is not limited to a small parameter space.

Continuous Contours: an Application of Synchronization

To show the computational power of synchronized networks, we decided to simulate a two-dimensional network, representing a small visual field. This visual field was subdivided into 16x16 subfields, arranged in a hexagonal grid (fig. 4). For each subfield there was a complete set of detector neurons with oriented receptive fields. These detectors are described more precisely in previous publications (Hartmann [7]). In our simulations we have simplified the feeding inputs and we have also reduced the complexity of the detector set by omitting detectors with highly curved receptive fields. We have added, however, the interconnections, necessary for synchronization.

Suppose a bright line is running through subfield A, B, C, D, E, F, G in fig. 4. The line shall fit to the receptive fields of neuron 1, 2, 3, 4, 5, 6, and 7, so that these neurons are excited. As in our chain configuration, neuron 1 and 2, 2 and 3, and all the other adjacent pairs shall be mutually interconnected. As we have seen in the last chapter, all the neurons of this chain will synchronize their spikes.
Now we change the input pattern to a slightly different contour, encoded by the neurons 1, 2, 8, 9, 10, and 7 (fig. 4). In this case neuron 2 must be connected with neuron 8 instead of neuron 3, and similarly neuron 7 with neuron 10 instead of neuron 6. There may be other lines, requiring connections between neuron 2 and 11 or 12 or 13. Generally spoken, each neuron in a subfield must be mutually connected with two groups of fitting neurons in two adjacent subfields.
As all neurons with "fitting" receptive fields are mutually linked now, also those neurons will always be connected, which are activated by any arbitrary continuous line. But the above discussed chain is just a subset of the complete interconnection, and so we only have to prove, that the additional connections will not disturb the synchronization of the chain. We have to discuss two cases. Firstly, a neuron with full interconnection will not only send triggering spikes to its active neighbours in the chain, but also to inactive fitting neighbours. But a triggering spike can not excite a neuron without sufficient feeding input, and no neurons outside the chain will be synchronized. Secondly, a neuron with full interconnection can not only receive triggering spikes from its neighbours, but also from its neighbours outside the chain. These outside neighbours, however, are not active and so they will not influence synchronization.
We have simulated this two dimensional network and presented different continuous lines activating up to 137 neurons. Synchronization was as good as in the case of simple chains. The synchronization was not significantly influenced, when we simulated spontaneous activity of "outside neighbours".
Self organization of synchronizing networks could start with a full interconnection between all neurons of neighbouring subfields. According to the Hebbian rule, weights of connections between those pairs of neurons would decrease, which are never driven simultaneously by a continuous stimulus.

References:

[1] von der Malsburg, C.: The correlation theory of brainfunction. Internal report 81-2, Dpt. Neurobiology, Max Planck Institute for Biophysical Chemistry (1981)

[2] Koenderinck, J.: The concept of local sign. In A. J. van Doorn et al., Eds. Limits in Perception, VNU Sci. Press. 495-549 (1984)

[3] Eckhorn, R. et al.: Feature linking via stimulus-evoked oscillations: Experimental results form cat visual cortex and funcional implications from a network model. Proc. IJCNN89, IEEE, 1.723-1.730 (1989)

[4] Freeman, W. J.: Mass action in the nervous system. Academic Press New York (1975)

[5] Gray, C. M., Singer, W.: Stimulus specific neuronal oscillations in the cat visual cortex: a cortical functional unit. Soc. Neurosc. abstr. 404.3 (1987)

[6] French, A. S., Stein, R. B.: A flexible neuronal analog using integrated circuits. IEEE Trans. Biomed. Eng., 17, 248-253 (1970)

[7] Hartmann, G.: Processing of continuous lines and edges by the visual system. Biol. Cybern. 47, 43-50 (1953)

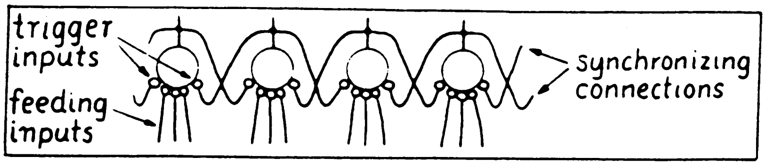


Fig. 1: The synchronizing connections in a chain of neurones.

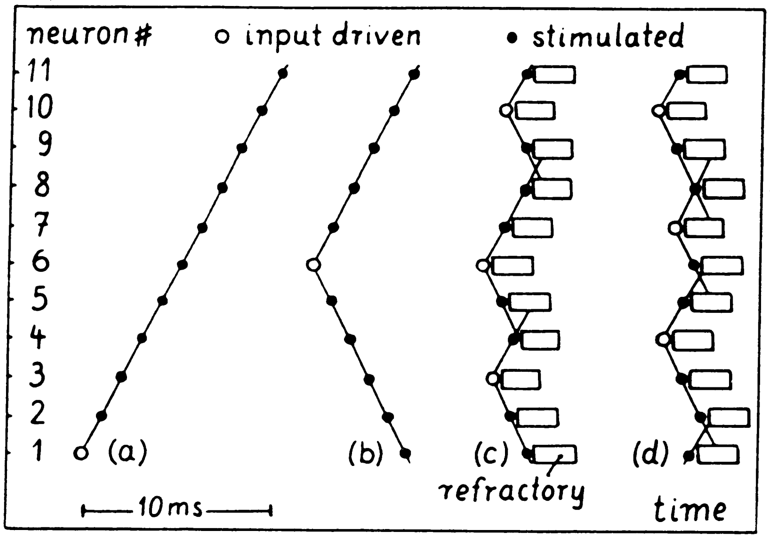


Fig. 2: The synchronization mechanism in a chain of neurones. Without input driven activity of the stimulated neurones, delay would cumulate proportional to the length of the chain (a,b). With input driven activity, wave fronts start from different points of the chain and stop at refractory neurones (c), (d).

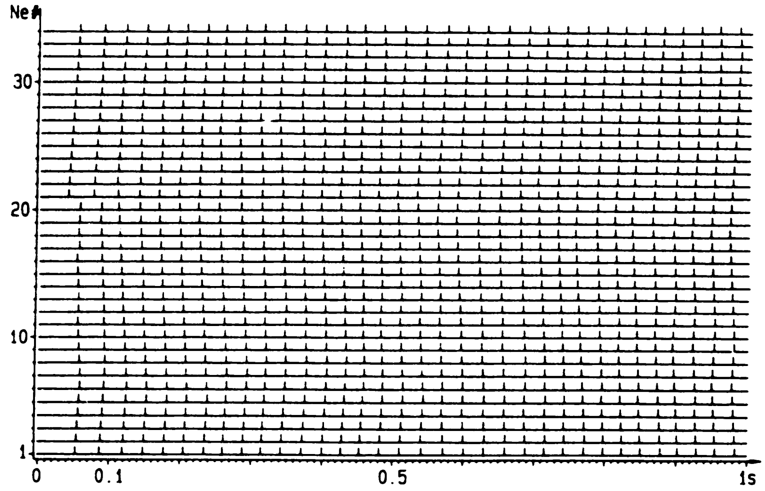


Fig. 3: Result of a simulation with 34 neurones. Notice, that the whole assembly is and remains synchronous after three spikes, when the stimulus is presented.

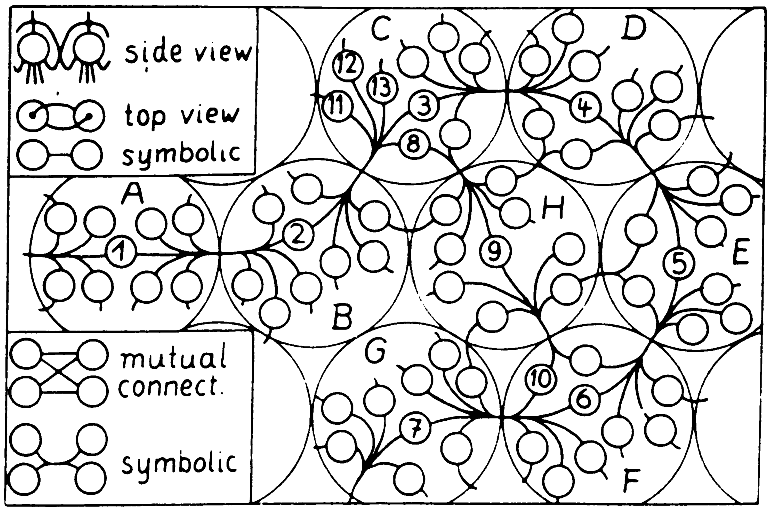


Fig. 4: The synchronizing connections in a two dimensional assembly of neurones. Only part of neurones is shown for clearness. Also for clearness, the two connections between a pair of neurones are symbolized by one (box top left). Similarly mutual connections between groups of neurones are symbolized (box bottom left).

SOME SIMILARITIES BETWEEN SINGLE-CELL RECORDINGS OF THE MOTOR CORTEX AND NEURAL NETWORKS: BROAD TUNING AND (POSSIBLY) TASK-MODULATED CHANGES IN NEURONAL OUTPUT

Larrie Hutton and Vincent Sigillito
The Johns Hopkins University Applied Physics; Laurel, MD 20707
Phone (301) 953-6242

James Sims
The Johns Hopkins University Space Telescope Science Institute
Baltimore, MD

Abstract

A neural net was trained to produce a 2-dimensional output vector that either matched an input vector or was displaced from it by 90°. The behavior of the elements of the system paralleled the output of neurons in the motor cortex of monkeys engaged in a task requiring directional judgments in 2 important ways: (1) neurons from both the neural network and the motor cortex acted as broadly tuned orientation detectors; and (2) the time-dependent changes in activity of the output neurons of the neural network reflected the time-dependent changes in the angle encoded by the population vector of cells in the motor cortex. In addition, we found that the amplitude of the response from output neurons was modulated by the task requirement even though the hidden nodes continued to retain their directional specificity.

Introduction

The neural representation of an input vector, and the pattern of neural activity generated as an organism responds to that input, are classic problems in neuroscience. In this paper we describe some parallels between the activity observed in individual neurons of the motor cortex of a monkey engaged in a visual-motor task, and the activity of individual neurons at the hidden and output layers of an artificial neural network.

The development of the neural network model was inspired by the work of Georgopoulos, *et al.* In one study (Georgopoulos, Schwartz, and Kettner, 1986), rhesus monkeys were trained to place their hand in the center of a sphere and then, when a light indicated the appropriate position, to move their hand to one of eight points that were strategically located on the interior surface of the sphere. The monkeys were able to accomplish this simple task quickly and accurately. What is important for our purposes is that single-cell recordings from the motor cortex indicated that a large number (224 were sampled) of neurons were active preceding the behavioral movement, that these neurons were quite broadly tuned (i.e., the frequency of firing for any particular neuron was a function of the angle of movement, but increasing deviations from the angle of maximum activity still resulted in substantial but declining firing rates), and that trigonometrically combining the individual activities resulted in a population vector that accurately predicted the final direction of the animal's arm.

In a later, related study (Georgopoulos, Lurito, Petrides, Schwartz, and Massey, 1989), rhesus monkeys were trained to move their arm in a direction counterclockwise and perpendicular to a line defined by the current position of the animal's hand and a stimulus light. Single-cell recordings from the motor cortex indicated that the combined population vector, as before, accurately predicted the final movement. In addition, the changes in the population vector during the time between the stimulus onset and movement (i.e., the reaction time) provided physiological evidence for the mental rotation model of Shepard and Cooper (1982).

In the first of the two current experiments, we trained a supervised neural network to perform a simple task: reproduce a 2-dimensional input from a distributed representation in the hidden layer (which corresponded to cells in the motor cortex) that had been built up during a training procedure. Because the input and output representations were isomorphic, this seems like an easy problem. In fact, it is easy to show that this "problem" can be solved without a hidden layer at all if the network is linear and the inputs are linearly separable. The fact that we had a third, "context node" for both the input and output representations, however, required the use of a hidden layer. The third node was used to indicate the presence or absence of the task variable: if that node were "off," the target output was identical to the input vector, and if "on" to rotate counterclockwise by 90°.

Our hypothesis was that the broadly tuned neurons seen in the animal studies would emerge here as broadly tuned feature detectors in the hidden layer of the experimental network, and that the broad tuning would persist with changes in the number of hidden nodes.

A second experiment was conducted in an attempt to replicate the "mental rotation" aspect of the second Georgopoulos, *et al.* (1989) experiment. In the second experiment, we used a recurrent network during the testing phase.

Method

We trained an ordinary multilayer feedforward network (Rumelhart, Hinton, and Williams, 1986) using back propagation to learn to predict a 2.5-dimensional output vector given a 2.5-dimensional input vector. (The "half-dimension" is the context node that indicated the presence or absence of the task variable. We avoid using the term "3-dimensional" because that would imply three spatial dimensions, which we did not investigate.) Two of the three elements of the input vector represented the horizontal and vertical components of a modified unit vector that was normalized and centered at (0.5, 0.5). Two of the three elements of the output vector (and therefore the target vector as well) were interpreted in the same way. Thus, (0.0, 0.5) would represent a half-unit vector pointing directly to the left, and the point (0.146, 0.146) would represent a half-unit vector directed to a point 135° counterclockwise from the vertical. These values were chosen so that neither the input vectors nor the targets used during training would fall outside the "unit square universe."

The third input node (the half dimension) was used to represent a task-specific context. When this node was 0.0, the net was trained merely to reproduce the input at the output level. When the context node was set to 1.0, the network was trained to move to a point 90° counterclockwise from the input. Without exception, the third target node was always identical to the third input node.

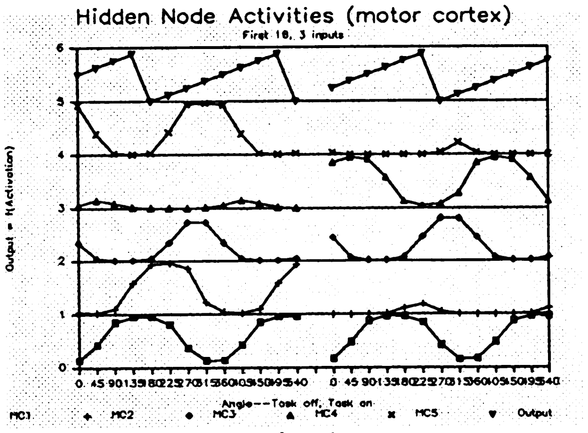
The training set for the conditions above consisted of 32 patterns--16 with the context node turned off and 16 with the context node turned on. The 16 points in each of the two major conditions were fairly evenly-spaced points (approximately 22.5° apart) on the modified unit circle described above. We investigated the behavior of the network with 1, 2, 4, 5, 8, 16, 32, 64, and 128 hidden nodes.

In a second experiment, the context node was ignored. The target was always a point 22.5° closer to the goal (which was, arbitrarily, the 0° point) from the current point. Details of this procedure, which involved a recurrent feedback loop during the testing phase, can be found in Hutton, Sigillito, and Egeth (1989).

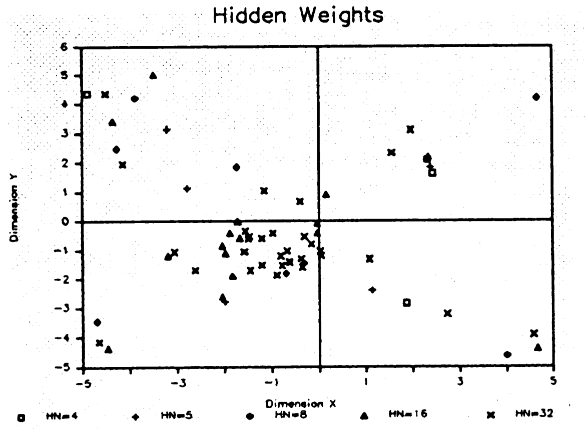
Results and Discussion

The results are summarized in the first figure, in which we show the outputs of the hidden nodes in the 5-hidden node case for both the no-task (left half of figure) and task-imposed (right half) condition. The 10 (5 hidden nodes per condition, times 2 conditions) bottommost figures are the output values (after the logistic function had been applied to the respective activations) for each of the 5 hidden nodes. The topmost pair of figures are the arctan transformations of the transformed output neurons. The horizontal axis ranges over an angle slightly larger than the full unit circle (i.e., it begins to double over) to more clearly show the peaks for each hidden node. Note that the horizontal axis is presented twice, once each for the two major conditions in the first experiment.

It might be noted that the hidden nodes arranged themselves approximately equally around the unit circle: one hidden node was "dedicated" to an angular area centered approximately every 70°. We found that was not true in general: as we increased the number of hidden nodes, they tended to space themselves out unequally.



Three significant features are evident from the first graph. The first, as noted, is that the hidden node neurons are very broadly tuned: although there are regions of relatively high activity, no neurons were quiescent at any point. Second, note that the effect of the task variable was to modulate the activity of a hidden node across' its range, but not to shift the point of maximum sensitivity. Although this is not too surprising in retrospect, this observation was not reported in the motor cortex studies. If a similar finding is verified, that may imply that individual neurons in the motor cortex are integrating both position and task information. Third, although the hidden nodes are "fuzzy" distributed representations, they can be (and are) accurately integrated by the output neurons (see the top two lines in the first figure) across more than one context. Thus, the internal representation is in principle available to any output vector that can exploit that representation. Here we formed the internal representation through the delta rule. There is of course no reason why an unsupervised network (competitive learning or a Kohonen network, for example) could not have been responsible for forming the weights between input and hidden layers.



The second figure shows the effect of increasing the number of hidden nodes. Each point represents the horizontal and vertical weight component for one hidden node in a particular condition. (For example, there are four points shown for the 4 hidden-node condition. Each of those points represents the weights from the horizontal and vertical input node for one of the four hidden nodes.) The figure shows data for conditions with 4, 5, 8, 16, and 32 hidden nodes; all were run for 1200 iterations. Although we expected the tuning to sharpen, this did not happen. The standard deviations for the weight vectors going to a particular hidden node from the two input nodes did not change in a systematic way for the cases (4, 5, 8, 16, 32, 64, and 128 hidden nodes) that we checked. If there is any trend to be discerned from our data, it is that increasing the number of hidden nodes (and thereby more closely approximating the situation found in the motor cortex) actually *broadens* the tuning curves. The second figure also indicates that most of the hidden nodes are relatively uncommitted to a particular orientation. (The "degree of commitment" can be estimated by the magnitude of the vector emanating from the center, at which point the horizontal and vertical components are respectively zero.) We also found that a larger number of hidden nodes required a much larger number of input presentations to learn the task. With 4, 8, and 16 hidden nodes, the proportion of variance accounted for in the target data approached 100% within 100 iterations; with 128 hidden nodes, the network had not learned when we stopped at 1200 iterations. It is reasonable to distinguish between *producing* an efficient representation, which our network needed to do, and *recruiting* an existing representation, which would be the case for a biological neural network. On average, our hidden nodes were less active (but neither consistently silent nor consistently active) if they were members of a larger hidden layer.

The effect of training to make gradual movements on the unit circle to a predetermined point produced two interesting results. First, the system, which recurrently fed back its own output as the sole input to the network (after being given an initial position) learned to move to a fixed point (the goal) regardless of its initial position on the unit circle. Second, initial positions outside the unit vector quickly resulted in movements back to the unit circle and subsequently toward the fixed point to which training had been directed. Thus, the net showed evidence of "habit formation" that permitted more rapid solutions to dynamic tasks.

Conclusions

The network described here clearly is too simple to capture the details of movement in any task approaching the complexity of even very simple motor tasks. Nevertheless, there are nontrivial similarities between the output of individual neurons in the motor cortex and the output of individual nodes in the hidden layer of our simple network: the broad tuning, the effective recruitment of a large percentage of the available cells, and the gradual changes in the output representation as an input vector was "mentally" rotated to an ultimate destination. We also found that the effect of introducing contextual information was to modulate the activity of directionally sensitive neurons in the hidden layer of our neural network. It would be interesting to see if this effect is found for neurons in the motor cortex of monkeys who are engaged in appropriate tasks.

References

Georgopoulos, A. P., Lurito, J. T., Petrides, M., Schwartz, A. B., and Massey, J. T. (1989). Mental rotation of the neuronal population vector. *Science*, 243, 234-236.

Georgopoulos, A. P., Schwartz, A. B., and Kettner, R. E. (1986). Neuronal population coding of movement direction. *Science*, 233, 1416-1419.

Hutton, L., Sigillito, V., and Egeth, H. (1990). Experiments on constructing a cognitive map: A neural network model of a robot that daydreams, APL Technical Report, RMI.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E., and McClelland, J. L. (Eds.), *Parallel Distributed Processing* (pp. 318-362). Cambridge, MA: MIT Press.

Shepard, R. N., and Cooper, L. A. (1982). *Mental Images and Their Transformations.* Cambridge, MA: MIT Press.

Neural Computation in a Vertebrate Adaptive Reflex System

W. T. Rogers, S. C. Dembinski, E. B. Graves[†]*, K. M. Spyer*[‡]*, A. R. Moser, and J. S. Schwaber*

E. I. DuPont Co., Computational Neurobiology Group, Imaging Systems Department and Engineering Physics Laboratory, Wilmington DE 19880-0357,
†College of William and Mary, Williamsburg VA, and
‡Royal Free Hospital, Dept. of Physiology, London, England

Introduction

The premise of this work is that computational principles evolved by biological systems are applicable to engineering devices. Our approach is based on detailed neurobiology, and we have chosen to study the baroreceptor vagal reflex. We experimentally determine facts of neural architecture and connection, and the biophysics of neural computation. These facts become models, and are run as computer simulations. Based on observation of real and simulated neuronal networks, it is our conviction that the brain is a highly nonlinear *dynamical* system that "computes" by virtue of its time-dependent traversal of state space; we analyze and interpret network function in these terms. Inasmuch as the model network incorporates the present knowledge of the baroreflex, we are using results of the simulation to guide new biological experiments. Thus, by continually refining our model of the reflex, we aim to distill the computational essence of the network.

System Description

The mammalian baroreceptor vagal reflex is of interest because it performs non-linear, analog, dynamical, adaptive signal processing, and is accessible to detailed experimentation. Three factors make this an ideal model system: (1) anatomical and functional access to inputs *and* outputs via peripheral nerves, (2) a relatively small number of neurons comprising the system, and (3) effective *in vitro* as well as *in vivo* experimental preparations.

In vertebrates, heart rate and contractility are under beat-to-beat regulation via the vagal baroreflex. Anatomically, the reflex circuitry lies in the brainstem, principally within the Nucleus Tractus Solitarii (NTS). First order neurons (cells incorporating stretch receptors in the major blood vessels) convey blood pressure and pulse rate information via peripheral nerves directly to second order neurons in the NTS. Neurons within the NTS project to the Nucleus Ambiguus (NA) which contains most of the output neurons to the heart. Altogether, there are perhaps several hundred neurons comprising the baroreceptor vagal reflex.

Biological Experiments

The anatomy of the reflex forms the framework which guides the rest of the experimental approach. The majority of the reflex circuitry lies within restricted levels of the brainstem, near the level of the obex [1,2]. Anatomical studies [3] have refined the location of the first order inputs to a distinct subdivision of the NTS. Within this subdivision, we are determining the pattern of connection of first order to second order neurons, and the size and extent of the second order population.

Based on the anatomical information, we are conducting electrophysiological experiments to characterize the biophysics and functional responses of second order neurons. The electrophysiological experiments are performed in an *in vitro* NTS slice preparation [4]. The biophysical characteristics measured include (a) resting membrane potential, (b) membrane resistance, (c) membrane time constant, and (d) voltage-dependent ionic conductances (such as post-inhibitory rebound currents, delayed rectifier currents, etc.). Network architecture and dynamical behavior are obtained from analysis of the time variation of intracellular events in response to various artificial stimuli. In particular, the fibers which encode blood pressure and heart rate [5] are accessible to stimulation in the slice using an extracellular electrode. Figure 1 shows intracellular recordings from an individual dorsomedial NTS cell. The upper trace shows response to stimulation of input fibers, while the lower trace shows response to intracellular depolarization. The excess of response in the case of extrinsic as compared with intrinsic stimulation is a network property, and analysis can, in principle, yield connectivity and dynamical information on the reflex network.

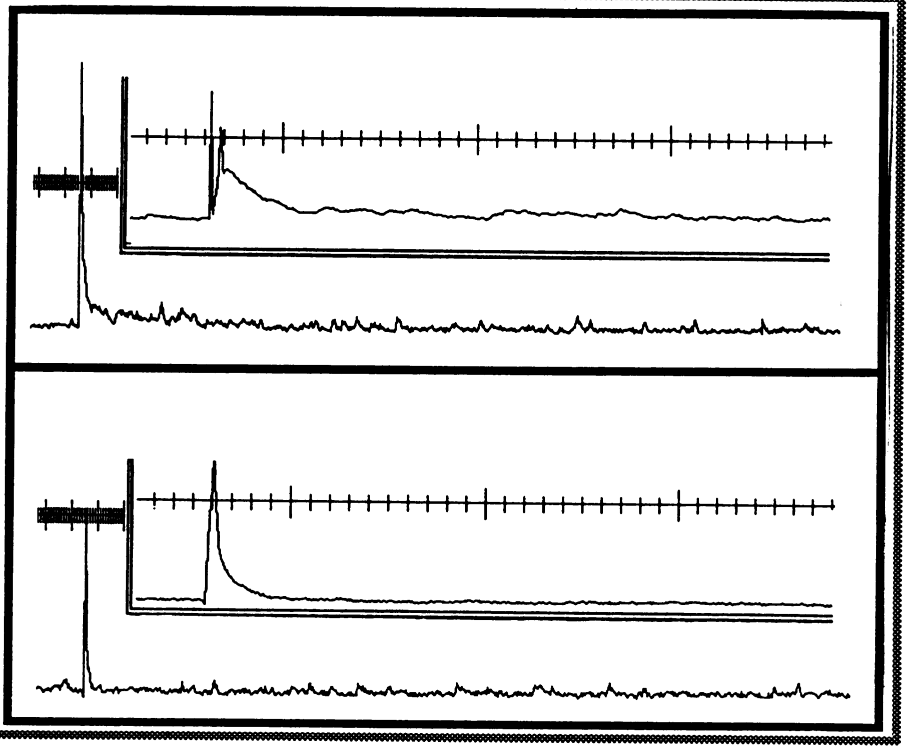


Figure 1. Intracellular recordings from the NTS slice. Upper trace: stimulation of fibers in the tractus solitarius with bipolar electrode. Lower trace: intrinsic depolarization to threshold. Time scale is 10 ms per small tic. Insets show the same data on an expanded time scale. Note the difference in the two recordings, particularly within the first several hundred milliseconds poststimulus.

Simulation

Simulation determines the computational meaning of biological data in the network model. Indeed, simulation allows us to extract computationally relevant aspects of network structure and function. Further, some experiments are infeasible in a biological preparation.

Simulations are written in an object-oriented language (C++), which provides for the encapsulation of the behavior of individual objects in a convenient way. Within this paradigm, arbitrarily complex collections of interacting objects are easily assembled. In our case, the basic objects are *Neurons* and *Synapses*. Since the second order cells we study receive primarily somatic inputs (as noted above), we presently use a single-compartment model [6], with intrinsic voltage-dependent sodium and potassium conductances generating action potentials (Fig. 2). Synaptic conductances take the same numerical form, but depend on a presynaptic action potential rather than intrinsic membrane potential for initiation. All conductances are assumed to follow a $\beta te^{-\alpha t}$ time course upon initiation [7].

Simulation starts with building up instances of objects whose intrinsic parameters correspond in a statistical sense to their biological counterparts. The coupled differential equations describing the ensemble of objects is then automatically solved by updating the state of each element. We drive the biological and simulated systems in similar ways and look for correspondence of behavior. An example of such a simulation is seen in Figure 3. In this case, the purpose of the simula-

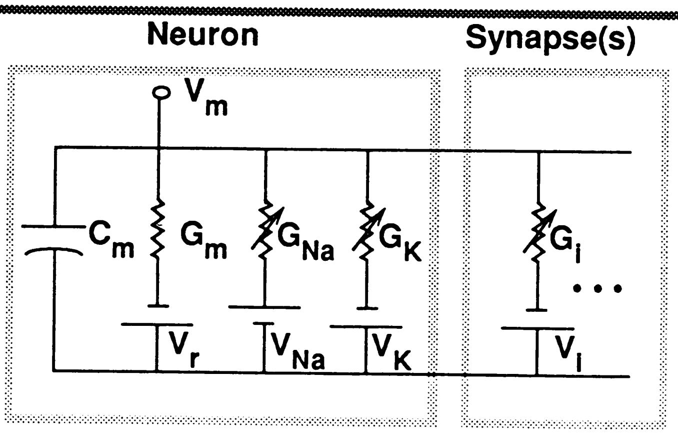


Figure 2: Circuit diagram of an individual model neuron. Excitatory vs. inhibitory synapses differentiated by the sign of the V_i.

tion was to determine the influence of the amount of interconnectivity within the various neuronal pools. The integrated activity of the second order pool was monitored as a function of the number of random synaptic contacts between second order neurons, as well as the number of synapses connecting the second order pool to a discrete population of inhibitory interneurons. Afferent to second order synapses were held constant at a ratio of 1.

Analysis

Our results strongly suggest that biological neural networks compute by virtue of their nonlinear dynamical properties. Individual neurons (simulated as well as biological) are intrinsically highly nonlinear due

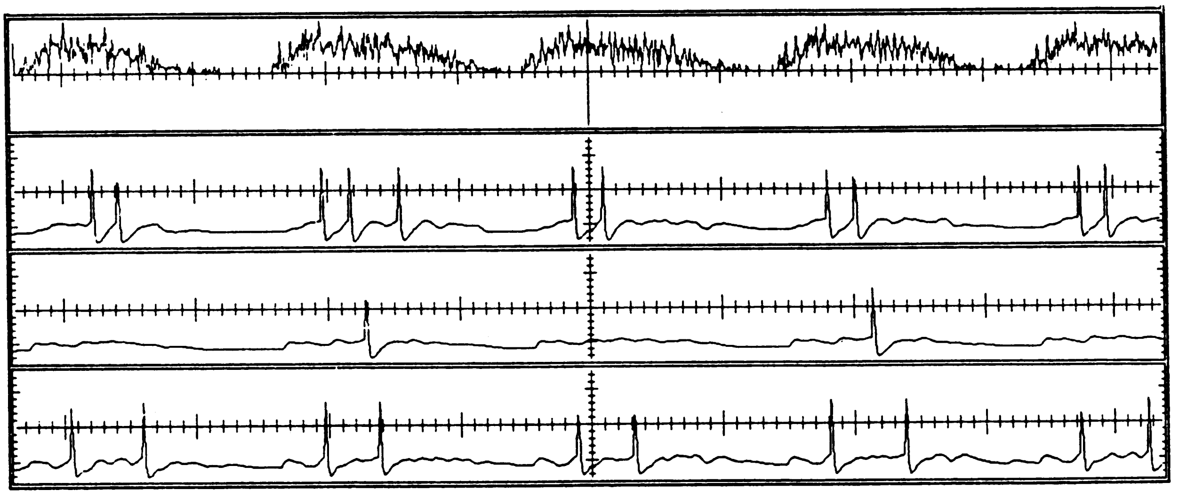


Figure 3. Simulation of Intracellular activity.
Simulated network consists of 200 afferent neurons, 200 2nd order neurons, 10% of which are spontaneously active, and 30 inhibitory interneurons. Total of 4600 synapses, 4000 of which are lateral connects within the 2nd order pool. Upper trace is simulated "field potential". Lower traces are intracellular recordings from randomly chosen 2nd order neurons. Each small tic on the horizontal axis is 10 ms, and on the vertical axis is 10 mV. Simulated afferent shock applied approximately every 200 ms.

to active processes inherent in their membrane biophysics. Dynamical behavior of an ensemble of such devices is dominated by the high dimensionality of the state space created by their coupling.

We are trying to characterize the dynamical behavior of neurons using methods from the field of chaos. In particular, phase plots, such as seen in Figure 4, permit visualization of the time evolution of the state of the system. Loops in phase space correspond to post-synaptic potentials; in general clockwise loops belong to excitatory post-synaptic potentials (epsp's), while counter-clockwise means inhibitory (ipsp) events. While the two phase plots seem qualitatively similar, they are different in important respects, indicating to some extent lack of correspondence between the dynamics of the simulation and the slice. The phase space trajectories developed in such a signal can be quantitatively described by the dimensions and entropies of their underlying attractor [8]. The order-2 information dimension (D_2) of a slice neuron is 2.5 0.2 and that of a model neuron is 3.1 0.2. D_2 for several model neurons was found to be indeterminate. We are currently refining the simulation parameters to improve quantitative correspondence.

Discussion and Conclusion

This paper discusses our initial efforts to understand computational principles, mechanisms, and architectures found in the vertebrate brain by modeling and simulation of a neuronal system of intermediate complexity: the baroreceptor vagal reflex. We propose that, unlike most artificial neural network implementations, the brain is a highly nonlinear *dynamical* system that "computes" by virtue of its time-dependent traversal of state space. Many network models created with the goal of understanding their biological counterparts are highly abstract, and succeed in elucidating limited aspects of biological systems. The present

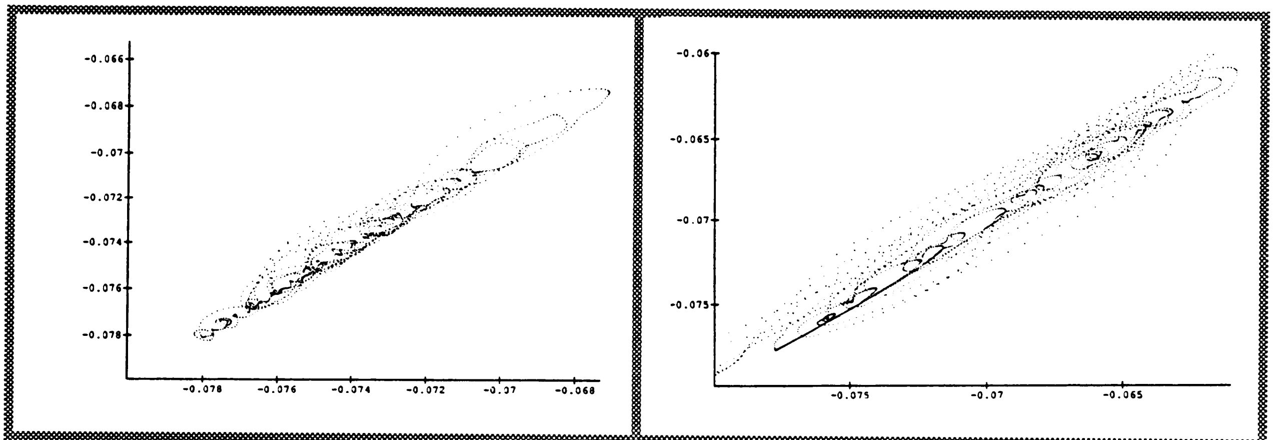


Figure 4. Phase plots of (left) 2nd order NTS neuron electrophysiologically recorded in the slice, and (right) a simulated 2nd order neuron. In both cases, the vertical axis is the intracellular membrane potential at time t+τ, while the horizontal axis is the same potential at time t, where τ is chosen to be 1.0 ms. The simulation parameters are described in the caption of Fig. 3.

alternative is to model biological systems with a significant degree of realism, and via computer simulation and VLSI emulation, elucidate the essential features of brain function within the context of specific functional circuits.

In general, any neuronal network constantly receives many parallel inputs, and continuously maps them to a set of parallel outputs. The relationship between inputs and outputs is often complex, and a task in emulating biological networks is to first find this relationship, and then understand the dynamical computational mechanisms underlying it. Once this is achieved, simulation provides a powerful method for investigating mechanisms and principles of computation not heretofore possible. With insights thus developed, we believe that systems can be constructed in silicon that embody dynamical properties similar to the model neuronal system.

References

1. Schwaber, J. S. Neuroanatomical substrates of cardiovascular and emotional-autonomic regulation. *In: Central and Peripheral Mechanisms in Cardiovascular Regulation. A. Magro, W. Osswald, D. Reis and P. Vanhoutte eds. Plenum,* 1986, 353-384.
2. Cox, G. E., Jordan, D. Moruzzi, P, Schwaber, J. S., Spyer, K. M., and Turner, S. A. Amygdaloid influences on brain-stem neurons in the rabbit. 1986. *J. Physiol.* **381** (Lond) pp.135-148 (1986).
3. Bradd J., Dubin J., Due B., Miselis R. R., Montor S., Rogers W. T., Spyer K. M., Schwaber J. S., Mapping of carotid sinus inputs and vagal cardiac outputs in the rat, *Soc. for Neurosci. Abstracts* (1989), in press.
4. Dekin M. S., Getting P. A., and Johnson S. M., *In Vitro* characterization of neurons in the ventral part of the Nucleus Tractus Solitarius, *J. Neurophysiol.* **58**(1), pp. 195-229 (1987).
5. Abboud F. M., and Chapleau A. W., Effects of pulse frequency on single-unit baroreceptor activity during sine-wave and natural pulses in dogs, *J. Physiol.* **401**, pp. 295-308 (1988).
6. Shephard, G. M., *Neurobiology*, 2nd ed., Oxford Press, 1988, pp. 134-138.
7. Traub R. D., Miles R., and Wong R. K. S., Model of the origin of rythmic population ascillations in the hippocampal slice, *Science* **243**, pp.1319-1325 (1989).
8. Albano A. M. et. al., Lasers and brains; Complex systems with low-dimensional attractors, in *Dimensions and Entropies of Chaotic Systems*, ed. G. Mayer-Kress, Springer-Verlag,1986, pp. 231-240.

IDENTIFICATION OF SYNAPTIC CONNECTIVITY USING A HIDDEN MARKOV MODEL

Xiaowei Yang * and Shihab A. Shamma †
Systems Research Center *†, and Electrical Engineering Department *†,
and the University of Maryland Institute for Advanced Computer Studies †
University of Maryland, College Park, Maryland 20742

Abstract: *A hidden Markov model is employed to identify the synaptic connectivity in neural networks. This new approach can substantially reduce the computational burden involved in the conventional correlation methods, and is suitable for either nonstationary or stationary neuronal firings. In the first phase, a postsynaptic spike train is used to estimate hidden parameters of the model such as the initial state probabilities and the state transition matrix, where the state is a variable representing the accumulated membrane potential. The optimal state path (the membrane potential process) is then estimated using a modified Viterbi algorithm, again, based on the post-synaptic spike train. In the second phase, a presynaptic spike train is transformed into a continuous postsynaptic activity. This activity becomes the standard against which the hidden Markov model estimate of postsynaptic activity is compared. The results of this comparison determine the connectivity parameters.*

1. Introduction

Correlation analysis of simultaneously recorded spike trains provides the information necessary to measure the internal structure of the biological neural networks. The conventional methodology used for this purpose includes cross-interval histograms, cross-correlation histograms, cross-covariance histograms and joint post-stimulus-time (PST) histograms [1]. By carefully choosing a normalization procedure, one can use the joint PST histogram to quantitatively measure the synaptic connectivity between a pair of neurons for a given model [3]. Nevertheless, the computational complexity involved is very large. The object of this paper is to present more efficient algorithms for computing pairwise connectivity.

In this report, we propose a new two step approach to determine connectivity in biological neural networks, an approach which can tremendously reduce the computational burden. This method, which is universal for stationary or nonstationary firings, uses a hidden Markov model to estimate the intensity process (membrane potential) of a doubly stochastic process (spike train) so that the synaptic connectivity between a neuron pair can be readily revealed.

Many signals can be modeled as probabilistic functions of Markov chains in which the observed signal is a random variable (or vector) whose probability density function depends on the current state of an underlying Markov chain. Because the current state of the underlying chain can not be determined by observation, such models are called *Hidden Markov Models* (HMMs). In other words, an HMM is a doubly stochastic process associated with an underlying Markov chain which is not observable, and each state in the chain is associated with a probabilistic function.

Our major concern here is to use the HMM to model a neuron so that we can estimate the membrane potential (state) from the observed firing sequence and, hence, identify the connectivity. In the first phase, we use a hidden Markov chain to model the behavior of the neuron so that the membrane potential sequence can be estimated based on the observed postsynaptic spike train. In the second phase, we analyze correlations between the estimated membrane potential and the presynaptic spike train to obtain the connectivity information. This procedure is described in Fig. 1, where $\{T_1, T_2, \cdots\}$ is the presynaptic spike train and $\{Y_0, Y_2, \cdots, Y_{T-1}\}$ is the postsynaptic firings; V_t represents the membrane potential and $h(t,s)$ is the synaptic connectivity with $\hat{V}_t$ and $\hat{h}(t,s)$ denoting the estimators of V_t and $h(t,s)$, respectively.

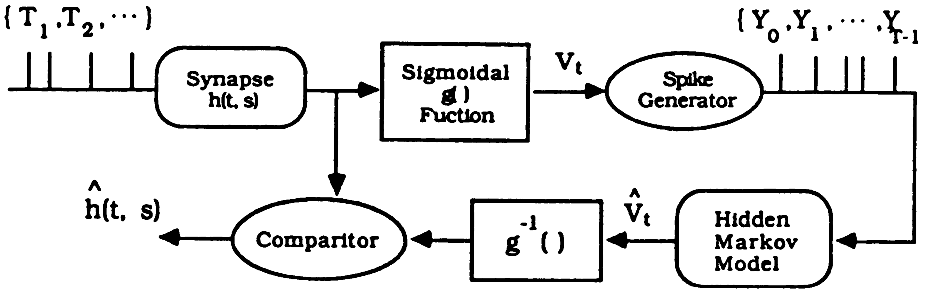


Figure 1. A systematic diagram of the identification scheme.

2. A Hidden Markov Model for Neurons

In order to analyze the observed spike train, discrete time bins are imposed so that the train may be represented as a point process. The bin width is chosen to be very small so that the spike train is converted into 0–1 process.

Let T be the length of the observation sequence. The membrane potential is discretized into N levels on $\mathbf{Q} = \{q_0, q_1, \cdots, q_{N-1}\}$. Therefore, a Markov chain of N states is associated with the discrete 0–1 process $\{Y_0, Y_1, \cdots, Y_{T-1}\}$. Since the membrane potential in extracellular recording is an unobservable quantity, the Markov chain is hidden. Denote by X_t the state of the chain at time t, the state transition probability distribution matrix is expressed as $A = \{a_{ij}\}$, where

$$a_{ij} = P_r(X_{t+1} = q_j | X_t = q_i) \tag{1}$$

with the initial state distribution

$$\pi_i = P_r(X_0 = q_i). \tag{2}$$

Let $b_j(k)$ be the observation probability distribution in state j, then

$$b_j(k) = P_r(Y_t = k | X_t = q_j). \tag{3}$$

Since the spike train is represented as a doubly stochastic process, the outcome depends fully on the current state, namely,

$$P_r(Y_0, Y_1, \cdots, Y_{T-1} | X_0, X_1, \cdots, X_{T-1}) = \prod_{t=0}^{T-1} P_r(Y_t | X_t) \tag{4}$$

with

$$P_r(Y_t = k | X_t) = \begin{cases} e^{-X_t}, & k = 0 \\ 1 - e^{-X_t}, & k = 1 \end{cases} \tag{5}$$

where the state of chain $X_t = \int_{t_0}^{t} V_\tau d\tau$ represents the accumulated membrane potential with t_0 denoted as the occurrence instant of the previous spike. Formulas (4) and (5) imply that the outcome sequence is generated based on information about the membrane potential process received via a "memoryless channel". The complete derivation of the doubly stochastic point process neuron model can be found in [4].

Our purpose at this stage is to find a path of the membrane potential process, or equivalently, to find the state path based on a given pattern of the spike train process. We can establish this by using the Viterbi algorithm [2] for finding the best state path, if the criterion is to individually optimize each state.

Let a realization of the membrane potential process be $(X_0, X_1, \cdots, X_{T-1}) = (q_{i_0}, q_{i_1}, \cdots, q_{i_{T-1}})$. The Viterbi algorithm can be used to find the best state path $(i_0^*, i_1^*, \cdots, i_{T-1}^*)$ in the sense of

$$P_r(\mathbf{Y}, \mathbf{X} = (q_{i_0^*}, q_{i_1^*}, \cdots, q_{i_{T-1}^*})) \geq P_r(\mathbf{Y}, \mathbf{X}), \ \forall\, \mathbf{X} \in \mathbf{Q}^T. \tag{6}$$

Before the Viterbi algorithm is implemented, the hidden parameters, $\{\pi_i\}$ and $\{a_{ij}\}$, have to be estimated. This estimation is carried out by the forward-backward procedure and the Baum-Welch reestimation formulas [2]. We found that the forward-backward procedure can be implemented by an artificial recurrent neural network.

The general Markov chain has essentially a full state transition probability matrix. However, the Markov chain structure of our neuron model is a left-to-right model. During the interval between successive spikes, the state evolves monotonically from a low level to a high level until a spike is triggered. Immediately after the spike, the state returns to the lowest level q_0. Thus the transition matrix has the form

$$\mathbf{A} = \begin{pmatrix} a_{00} & a_{01} & \cdots & a_{0,N-1} \\ a_{10} & a_{11} & & a_{1,N-1} \\ \vdots & & \ddots & \vdots \\ a_{N-1,0} & \mathbf{O} & & a_{N-1,N-1} \end{pmatrix} \tag{7}$$

Since $X_0 = 0$, we have $q_0 = 0$. This implies $b_0(Y_t = 0) = 1$, i.e., the neuron does not generate spikes at $X_t = q_0$, which reflects the absolute refractory effect.

Because a spike in the postsynaptic neuron (described as $Y_t = 1$) means that the accumulated membrane potential at the next instant returns to the resting level (described as $X_{t+1} = 0$), the Viterbi algorithm should be modified for this application.

3. Identification of Synaptic Connectivity

Let us denote by $\{T_1, T_2, \cdots\}$ the observed presynaptic spike train with T_k representing the time of the occurrence of the k-th spike. The associated counting process $N(t)$ represents the number of spikes in the time interval $(0, t]$. Let Δt be the time bin width. Then a discrete version of the state can be described as

$$X_t = \sum_{k=0}^{t} V_k \Delta t \tag{8}$$

so that

$$X_{t+1} = \begin{cases} X_t + g(\sum_{k=1}^{N(t)} h(t,T_k))\Delta t, & Y_t = 0 \\ 0, & Y_t = 1 \end{cases} \tag{9}$$

where the sigmoidal function $g(u)$ is quantized to take values on $\{0, q_1/\Delta t, q_2/\Delta t, \cdots, q_{N-1}/\Delta t\}$. Since the accumulative nature of X_t, the value of the largest state level q_{N-1} is not fixed, rather, it has a dynamic range of $[q_{N-1}, \infty)$. This means that whenever $X_t \geq q_{N-1}$, X_t is considered to be at state q_{N-1}.

After the optimal state path is determined by the modified Viterbi algorithm, the membrane potential sequence is evaluated as

$$V_t = g(\sum_{k=1}^{N(t)} h(t,T_k)) = \frac{X_{t+1} - X_t}{\Delta t}, \; for\; Y_t = 0,\; t = 0, 1, \cdots, T-2. \tag{10}$$

Suppose that the connectivity has a specific form, for instance, $h(t,s) = we^{-\sigma(t-s)}$, where the time constant σ is sufficiently large. Then we can identify the connectivity parameters by the estimators

$$\hat{w} = \sum_{k:Y_{T_k}=0} g^{-1}(\frac{X_{T_k+1} - X_{T_k}}{\Delta t}) \tag{11}$$

and

$$\hat{\sigma} = \sum_{t:Y_t=1} \frac{-1}{t+1-T_{i_t}} \log \frac{1}{w} g^{-1}(\frac{X_{t+2}}{\Delta t}) \tag{12}$$

where $T_{i_t} = \max(T_k : T_k < t+1)$. This example suggests that the nonlinearity g can reduce the number of computations required. We are now investigating whether this method can be generalized to estimate pairwise connectivities for larger groups of neurons.

Acknowledgments

We wish to thank Dr. James W. Fleshman for his valuable comments and suggestions. This research was funded in part by a grant from the Whitaker Foundation.

References

[1] G.L. Gerstein, "Functional association of neurons: detection and interpretation," In: F.O. Schmitt (ed) *The Neurosciences Second Study Program*, Rockefeller Univ. Press, New York, 1970, pp. 648–661.

[2] L.R. Rabiner and B.H. Juang, "An introduction to hidden Markov models," *IEEE ASSP Magazine*, Jan. 1986, pp. 4–16.

[3] X. Yang and S.A. Shamma, "Identification of connectivity in neural networks," *Tech. Res. Report*, Systems Research Center, University of Maryland, TR–89–36, 1989.

[4] X. Yang, "Detection and classification of neural signals and identification of neural networks," *Ph.D. Dissertation*, Electrical Engineering Department, University of Maryland, 1989.

Pattern Recognition and Analysis of Network Dynamics

WHY TWO HIDDEN LAYERS ARE BETTER THAN ONE

Daniel L. Chester
Dept. of Computer and Information Sciences
University of Delaware
Newark, DE 19716

1 Introduction

With the introduction of the back propagation algorithm by Rumelhart, Hinton and Williams [5], the feedforward neural net has become a popular architecture for practical applications. This architecture, consisting of an input layer, one or more hidden layers and an output layer, is widely believed to be more powerful than perceptrons, which have no hidden layer. This belief was strengthened when several people ([2], [4]) pointed out that a net with two hidden layers can compute any continuous mapping. Building on a result by Irie and Miyake [3], Funahashi [1] proved that any continuous mapping can be approximated by a net with one hidden layer, assuming only that the transfer function computed by a neuron is nonconstant, bounded, continuous and monotone increasing.

One might interpret this result to mean that one hidden layer is sufficient for any practical purpose and there is no need for more than one hidden layer. This interpretation, however, is wrong. Funahashi assumes that the function to be approximated is given and that an unlimited number of neurons are available. Both of these assumptions are violated in most practical applications of feedforward nets. This paper shows that when a net is trained on a finite sampling of points in the domain of a function to be approximated, the resulting net may be very different from the nets referred to in Funahashi's theorems. And even if it were similar, the number of neurons in a net with one hidden layer might have to grow without bound to improve accuracy, while a simple net with two hidden layers might achieve any level of accuracy with just the adjustment of a few parameters.

For our purposes we will assume that a neuron with n inputs computes a function $\phi(a\mathbf{D}^T\mathbf{X} + b)$, where a and b are scalars, $\mathbf{X}$ is a vector in (Euclidean) n-space, $\mathbf{D}$ is a unit vector in n-space and ϕ is a nonconstant, bounded, monotone increasing function. (Note that continuity is not assumed.) In a multilayer net, the input layer just provides the input $\mathbf{X}$. The neurons in the first hidden layer compute on this input, producing a vector of outputs, which is the input to the neurons in the next hidden layer, and so on. Without loss of generality we can assume that the output layer just forms a linear combination of the neural outputs in the last hidden layer. Thus, the function computed by a net with one hidden layer can be represented by

$$y(\mathbf{X}) = d + \sum_{i=1}^{N} c_i\phi_i(a_i\mathbf{D}_i^T\mathbf{X} + b_i)$$

where N is the number of neurons in the hidden layer and d is a scalar. (Note here that each neuron can have its own transfer function ϕ. They need not be all the same.)

Feedforward nets are usually trained by presenting them with sample inputs and corresponding values called **targets** indicating the desired outputs. If $\mathbf{X}$ is a sample input, the corresponding target will be denoted by t. A finite set of input-target pairs is usually called a **training set**. These sets may be constructed by hand in simple experiments (e.g., the classic XOR problem), but the more interesting case is when training sets are constructed from empirical data, when the desired function is unknown (e.g., stock prices as a function of various parameters). (The fact that the data is empirical raises the issue of noise in the data, but that will be ignored in this paper.)

Suppose a net computes a function y and ϵ is any number > 0. For any given training set, we shall say that the net **ϵ-computes** the set if for each input $\mathbf{X}$ in the set, $|y(\mathbf{X}) - t| < \epsilon$.

2 The Illusion of Accuracy

What kind of net will we get if we train it on a given training set? Consider the case when the net has one hidden layer and no restriction is placed in advance on the number of neurons in the hidden layer. We first consider the case when the input space is one-dimensional. Let us say that a net **ϵ-interpolates** a training set if it ϵ-computes the set and, assuming x_1 is the smallest input and x_k is the largest input in the set, the output $y(x)$ for the net satisfies $|y(x) - t_1| < \epsilon$ for $x < x_1$ and $|y(x) - t_k| < \epsilon$ for $x > x_k$. It is obvious that if there are only two input-target pairs in the training set, only one neuron is needed in the hidden layer and for any $\epsilon > 0$ the parameters a, b, c and d can be found to ϵ-interpolate the set. Suppose that for any $\epsilon > 0$ and any training set having $k \geq 2$ input-target pairs there are nets with one hidden layer that ϵ-interpolate the training set. Then for a given training set S having k input-target pairs, there is a net $\mathcal{N}_k$ that $\epsilon/2$-interpolates the set. The function computed by $\mathcal{N}_k$ can be assumed to be

$$y_k(x) = d_k + \sum_{i=1}^{k-1} c_i\phi_i(a_ix + b_i).$$

Assuming without loss of generality that the input x_{k+1}, with corresponding target t_{k+1}, is larger than x_k, there is a net with one neuron in its hidden layer that $\epsilon/2$-interpolates the two-pair training set which pairs t_k with input x_k and t_{k+1} with input x_{k+1}. Let the function computed by this net be represented by

$$y(x) = d + c_k\phi_k(a_kx + b_k).$$

Then this net can be combined with $\mathcal{N}_k$ to form a net $\mathcal{N}_{k+1}$ that ϵ-interpolates the training set formed by adding the k + 1st input-target pair to S and computes the function

$$y_{k+1}(x) = d_k + d - t_k + \sum_{i=1}^{k} c_i\phi_i(a_ix + b_i).$$

Thus, by the principle of induction, we have proven that for any $\epsilon > 0$ and any training set containing $k \geq 2$ input-target pairs, there is a net that ϵ-interpolates the set and which contains $k - 1$ neurons in its one hidden layer.

The multi-dimensional case can be reduced to the one-dimensional case by noting that for a given finite set of points, a line can always be found onto which the points can be uniquely projected. If these points are the inputs in a training set S, the corresponding targets can be paired with the projections of the points on the line to make a training set S' with one-dimensional inputs. For any $\epsilon > 0$ there is a net with one-dimensional inputs that ϵ-interpolates S'. Assuming, without loss of generality, that the map from the input space to the line is the function $\mathbf{D}^T\mathbf{X}$, this net can be transformed into the required net with multi-dimensional inputs by substituting $\mathbf{D}^T\mathbf{X}$ for x in its functional description. We have thus proved the following

Theorem 1 *For any $\epsilon > 0$ and any training set containing k input-target pairs, there is a net having one hidden layer and $k - 1$ neurons within that layer that ϵ-computes the training set, that is, if y is the function computed by the net, then for any input $\mathbf{X}$ in the training set with corresponding target t, $|y(\mathbf{X}) - t| < \epsilon$.*

If a feedforward net that has been trained on a particular training set is only given inputs that appear in the training set, the net can calculate the targets to any required degree of accuracy; the above theorem states that one neuron for each input-target pair in the set is sufficient to do this. As a practical matter, however, a table lookup would be a more efficient way to compute such a set if that many neurons are needed. A neural net is potentially worthwhile, however, if the number of neurons in it is much less than the number of pairs in the training set and the net is going to be applied to new inputs not in the training set.

The above theorem suggests that the input-target pairs in a training set can be computed to any required degree of accuracy, but this accuracy is an illusion as soon as new inputs not in the set are fed to a net. The theorem says nothing about what values will be returned on such inputs, and, as will be shown, any value can be returned when the net has only one hidden layer. Let us call a training set a **pinnacle set** if its inputs are two-dimensional, one input is $[0,0]^T$ for which the corresponding target is 1, all other inputs are outside the unit circle and their corresponding target is 0. We will show that for all $\epsilon > 0$, all pinnacle sets can be ϵ-computed by a net having a single hidden layer that consists of two neurons. Moreover, there will be inputs outside a pinnacle set, and outside the unit circle, for which the net will evaluate to 1, no matter how the inputs in the pinnacle set are arranged.

Suppose a pinnacle set S and $\epsilon > 0$ are given and two neurons have the same transfer function ϕ. Let b be such that $\phi(b)-\phi(b+1)$ is maximal. Let c be such that $c(\phi(b)-\phi(b+1)) = 1$. Now let $\mathbf{D}$ be a unit vector pointing in the direction of a line passing through $[0,0]^T$ but not through any other input in S. Finally, let $\mathcal{N}$ be the net constructed from these two neurons and described by

$$y(\mathbf{X}) = c(\phi(a\mathbf{D}^T\mathbf{X} + b) - \phi(a\mathbf{D}^T\mathbf{X} + b + 1)).$$

Note that $y([0,0]^T) = 1$ and for large $a\mathbf{D}^T\mathbf{X}$, $y(\mathbf{X})$ approaches 0. Since the line associated with $\mathbf{D}$ does not pass through any input in S other than $[0,0]^T$, the scalar a can be increased until $\mathcal{N}$ ϵ-computes S. But notice that for any input $\mathbf{X}$ on the line perpendicular to $\mathbf{D}$, $\mathbf{D}^T\mathbf{X} = 0$ and hence $y(\mathbf{X}) = 1$ everywhere on that line. Thus, no matter how the inputs in S are arranged, or how many there are, $\mathcal{N}$ can be adjusted to ϵ-compute S and yet there will still be a ridge where the output of $\mathcal{N}$ rises from near 0 all the way to 1. So even if our intuition tells us that the only reasonable smooth functions that compute the input-output pairs in S are those that peak at $[0,0]^T$ and are 0 everywhere outside of the unit circle, $\mathcal{N}$ will be nowhere close to approximating any of them. Hence the accuracy that appears to be achieved by tuning $\mathcal{N}$ until it ϵ-computes a training set is illusory.

3 The High Cost of a Single Hidden Layer

Let us call a smooth function peaking at $[0,0]^T$ and evaluating to 0 everywhere outside of the unit circle a **pinnacle function**. If two neurons are not enough to approximate a pinnacle function, how many neurons must be added to do so? To find the answer, consider a net $\mathcal{N}$ having one hidden layer. Each neuron in the hidden layer has an associated direction vector $\mathbf{D}$. All the neurons with the same vector $\mathbf{D}$ (or $-\mathbf{D}$) can be lumped together and thought of as computing some univariate function f applied to the projection of $\mathbf{X}$ onto the line having direction $\mathbf{D}$. Thus the output of $\mathcal{N}$ can be described by

$$y(\mathbf{X}) = d + \sum_{i=1}^{k} f_i(\mathbf{D}_i^T\mathbf{X})$$

where k is the number of distinct lumps of neurons. Each function f_i has the characteristic that it varies when its argument is somewhere near 0, but approaches a limit as its argument approaches either $+\infty$ or $-\infty$. Since each direction $\mathbf{D}_i$ is distinct, if $\mathbf{X}$ is sufficiently far out from the line having direction $\mathbf{D}_i$ while still being nearly perpendicular to it, the local variation in $y(\mathbf{X})$ will be due to the variation in f_i and not to variations in the other functions. Letting *maxvar* be a functional that evaluates to the maximum variation in a function, we note that

$$1 \leq maxvar(y) \leq \sum_{i=1}^{k} maxvar(f_i)$$

Consequently, the largest value of $maxvar(f_i)$ must be at least $1/k$. If, for a given $\epsilon > 0$, we want this to be $< \epsilon$, k must be at least $1/\epsilon$. But there must be at least k neurons in the hidden layer of $\mathcal{N}$, so there must be at least $1/\epsilon$ neurons in a single hidden layer if the net approximates a pinnacle function to within ϵ of the required values.

4 The Advantage of a Second Hidden Layer

The advantage of a second hidden layer may now be seen; while increasingly better approximations to pinnacle functions can only be achieved by adding more neurons to a net with one hidden layer, a net with two hidden layers can approximate pinnacle functions with arbitrary accuracy (at the center and outside of the unit circle) with just four neurons in the hidden layers. Let $\mathbf{D}_1$, $\mathbf{D}_2$ and $\mathbf{D}_3$ be three unit vectors in 2-space that point in directions $120°$ from each other and suppose that four neurons have the transfer function ϕ such that the output range of ϕ is $[0,1]$. Three of these neurons can be put in the first hidden layer and the fourth can be put in the second hidden layer so that they compute the function

$$y(\mathbf{X}) = \phi(a(.5 - \sum_{i=1}^{3} \phi(a(\mathbf{D}_i^T\mathbf{X} - .1))))$$

where a is an adjustable parameter. For any $\epsilon > 0$, a can be made large enough so that for most of the points $\mathbf{X}$ in the equilateral triangle defined by $\mathbf{D}_i^T\mathbf{X} \leq .1$, $i = 1,2,3$, the neurons in the first layer output 0 approximately, while for most of the points outside of that triangle one or more of them outputs 1 approximately, so that $|y([0,0]^T) - 1| < \epsilon$ and for any point $\mathbf{X}$ outside the unit circle, $|y(\mathbf{X})| < \epsilon$.

5 Conclusion

The problem with a single hidden layer is that the neurons therein interact with each other globally, making it difficult to improve an approximation at one point without worsening it elsewhere. With two hidden layers it is possible to have some of the neurons in the first layer partition the input space into small regions (e.g., the equilateral triangle above) and other neurons therein compute the desired function within those regions; then for each region a neuron in the second hidden layer can combine the outputs of corresponding first hidden layer neurons so that it computes the desired function within that region and outputs 0 everywhere else. In this way the effects of the neurons are isolated and the approximations in different regions can be adjusted independently of each other, much as is done in the Finite Element Method for solving partial differential equations or the spline technique for fitting curves.

References

[1] Ken-ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–192, 1989.

[2] R. Hecht-Nielson. Kolmogorov mapping neural network existence theorem. In *IEEE First International Conference on Neural Networks*, pages III(11–13), 1987.

[3] B. Irie and S. Miyake. Capabilities of three-layered perceptrons. In *IEEE International Conference on Neural Networks*, pages I(641–648), 1988.

[4] T. Poggio. Visual algorithms. In O. J . Braddock and A. C. Sleigh, editors, *Physical and Biological Processing of Images*, pages 128–135, Springer-Verlag, New York, 1983.

[5] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In David E. Rumelhart, James L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing*, chapter 8, pages 318–362, The MIT Press, Cambridge, Massachusetts, 1986.

On the Optimality of the Sigmoid Perceptron

Bill Horne and Don Hush
Dept. of Electrical and Computer Engineering
University of New Mexico
Albuquerque, NM 87131

1.0 Introduction

In this paper we show that a second order perceptron with sigmoid nonlinearity can implement the exact form of the a posteriori probability for the two class Gaussian problem. Further, we show that if the weights of the perceptron are chosen to minimize the mean squared error performance criterion, the mapping that results is exactly the a posteriori probability. This has two important implications. First, this means that the sigmoid perceptron can be trained to implement the optimal Bayes classifier. That is, for the two class Gaussian problem, minimizing the MSE is equivalent to minimizing the classification error. Second, unlike other perceptron models, the sigmoid perceptron, in computing the a posteriori probability, provides a statistically accurate measure of the confidence in the classification assignment.

The relationship between neural networks and a posteriori probabilities has been discussed by Asoh and Otsu [1]; however, we are not aware of any formal proofs like those given in this paper.

2.0 Background

For a two class classification problem, Bayes classification rule is given by,

$$P(\omega_1|\mathbf{x}) \overset{\omega_1}{\underset{\omega_2}{\gtrless}} P(\omega_2|\mathbf{x}) \qquad (1)$$

where $P(\omega_i|\mathbf{x})$ is the a posteriori probability that a given data sample $\mathbf{x}$ belongs to class ω_i. The a posteriori probability can be viewed as a measure of confidence of class membership.

Simplification of (1) often leads to linear or quadratic discriminant functions. Linear discriminant functions are of the form,

$$d(\mathbf{x}) = \mathbf{w}^T\mathbf{x} \qquad (2)$$

where $\mathbf{x}$ is an *augmented* data vector given by $\mathbf{x} = (x_1 \; x_2 \; \dots \; x_n \; 1)^T$ and $\mathbf{w}$ is a vector of scalar weights. A decision boundary is formed when $d(\mathbf{x}) = 0$ which partitions the pattern space into two regions with an n-dimensional hyperplane.

Similarly, quadratic discriminant functions are of the form,

$$d(\mathbf{x}) = \mathbf{x}^T A\mathbf{x} + \mathbf{b}^T\mathbf{x} + c \qquad (3)$$

where $\mathbf{x}$ is not augmented, A is a symmetric matrix, $\mathbf{b}$ is a vector, and c is a scalar bias weight. A quadratic decision boundary is formed when $d(\mathbf{x}) = 0$. Although (3) is quadratic in $\mathbf{x}$ it is linear in the weights and thus can be expresses in a form similar to (2) given by,

$$d(\mathbf{x}) = \mathbf{w}^T\mathbf{z} \qquad (4)$$

where,

$$\mathbf{z} = (x_1^2 \;\; x_1x_2 \;\; \dots \;\; x_1x_n \;\; x_2^2 \;\; \dots \;\; x_n^2 \;\; x_1 \;\; x_2 \;\; \dots \;\; x_n \;\; 1)^T \qquad (5)$$

and,

$$\mathbf{w} = (A_{11} \;\; (A_{12} + A_{21}) \;\; \dots \;\; (A_{1n} + A_{n1}) \;\; A_{22} \;\; \dots \;\; A_{nn} \;\; b_1 \;\; b_2 \;\; \dots \;\; b_n \;\; c)^T$$

For the sake of clarity it will be necessary to use both the form in (3) and in (4) for quadratic discriminant functions during the foregoing discussion. Note that the appropriate equation can be identified by the notation being used.

This work was supported in part by Sandia National Laboratories, Albuquerque, New Mexico, under contract number 05-8801.

The most common nonlinearity used in perceptrons today is the *sigmoid* function,

$$f(x) = \left[1 + e^{-x}\right]^{-1}$$

Justification for using this function is that it is computationally convenient, motivated by biological systems, and simply that it is a continuous function that closely resembles a hard limiter. All of these may be true, but the goal of this paper is to show that this function has important statistical properties as well.

We will define the *sigmoid perceptron* as a second order perceptron [2, 3] which computes a single output, u, from a vector of inputs, **x**. The transfer function is given by,

$$u(\mathbf{x}) = f(\mathbf{x}^T A \mathbf{x} + \mathbf{b}^T \mathbf{x} + c)$$

or equivalently,

$$u(\mathbf{x}) = f(\mathbf{w}^T \mathbf{z})$$

where $f(\cdot)$ is the sigmoid function.

The mean squared error criterion function is given by,

$$J(\mathbf{w},\mathbf{x}) = \tfrac{1}{2} E\left\{ (r(\mathbf{x}) - u(\mathbf{x}))^2 \right\}$$

Its gradient is given by,

$$\nabla_{\mathbf{w}} J(\mathbf{w},\mathbf{x}) = E\{(r(\mathbf{x}) - u(\mathbf{x}))\, u(\mathbf{x})(1 - u(\mathbf{x}))\mathbf{z}\} \tag{6}$$

where,

$$r(\mathbf{x}) = \begin{cases} 1 & x \in \omega_1 \\ 0 & x \in \omega_2 \end{cases}$$

3.0 Two Class Gaussian a posteriori Probability

The conditional probability density function of a Gaussian function is given by,

$$p(\mathbf{x} \mid \omega_i) = \frac{1}{(2\pi)^{\frac{n}{2}} \mid C_i \mid^{\frac{1}{2}}} \exp\left[-\tfrac{1}{2} (\mathbf{x} - \mathbf{m}_i)^T C_i^{-1} (\mathbf{x} - \mathbf{m}_i)\right] \tag{7}$$

where n is the dimension if the data, $\mathbf{m}_i$ is the mean and C_i is the covariance matrix for class ωi. For the two class Gaussian problem we can apply Bayes Rule to determine the a posteriori probability of class one as follows,

$$P(\omega_1 | \mathbf{x}) = \frac{P(\omega_1)\, p(\mathbf{x} \mid \omega_1)}{P(\omega_1)\, p(\mathbf{x} \mid \omega_1) + P(\omega_2)\, p(\mathbf{x} \mid \omega_2)} = \left[1 + \frac{P(\omega_2)\, p(\mathbf{x} \mid \omega_2)}{P(\omega_1)\, p(\mathbf{x} \mid \omega_1)}\right]^{-1} \tag{8}$$

substituting (7), yields,

$$P(\omega_1 | \mathbf{x}) = \Big[1 + \exp\Big[-\tfrac{1}{2} (\mathbf{x} - \mathbf{m}_2)^T C_2^{-1} (\mathbf{x} - \mathbf{m}_2) + \tfrac{1}{2} (\mathbf{x} - \mathbf{m}_1)^T C_1^{-1} (\mathbf{x} - \mathbf{m}_1)$$
$$+ \ln\left[P(\omega_2) \mid C_1 \mid^{\frac{1}{2}}\right] - \ln\left[P(\omega_1) \mid C_2 \mid^{\frac{1}{2}}\right]\Big]\Big]^{-1}$$

This equation corresponds to the form of a sigmoid perceptron where,

$$A = \tfrac{1}{2}\left(C_2^{-1} - C_1^{-1}\right)$$

$$\mathbf{b} = C_1^{-1}\mathbf{m}_1 - C_2^{-1}\mathbf{m}_2$$

$$c = \tfrac{1}{2}\left(\mathbf{m}_2^T C_2^{-1}\mathbf{m}_2 - \mathbf{m}_1^T C_1^{-1}\mathbf{m}_1 - \ln\left[P(\omega_2) \mid C_2 \mid^{\frac{1}{2}}\right] + \ln\left[P(\omega_1) \mid C_1 \mid^{\frac{1}{2}}\right]\right) \tag{9}$$

When the covariances are equal, the problem reduces to a first order perceptron. Note also that these weights, when used in equation (3), provide optimal classification.

4.0 Bayesian Weights and the Expected Squared Error Criterion Function

The previous section showed that a second order perceptron has the correct form for computing the a posteriori probabilities in two class Gaussian classification problems. This section shows that minimizing the expected squared error criterion function results in the weights given in (9). To check the first order necessary condition, we must set the gradient with respect to the weights equal to zero and solve for the weights. Substituting for $r(\mathbf{x})$ in (6) gives,

$$\begin{aligned}\nabla_{\mathbf{w}} J(\mathbf{w},\mathbf{x}) = {} & P(\omega_1)\, E\{[1-u(\mathbf{x})]\, u(\mathbf{x})[1-u(\mathbf{x})]\, \mathbf{z} \mid \mathbf{x}\in\omega_1\} \\ & + P(\omega_2)\, E\{-u(\mathbf{x})u(\mathbf{x})[1-u(\mathbf{x})]\, \mathbf{z} \mid \mathbf{x}\in\omega_2\}\end{aligned}$$

substituting the sigmoid yields,

$$\nabla_{\mathbf{w}} J(\mathbf{w},\mathbf{x}) = P(\omega_1)\, E\left\{\frac{e^{-2\mathbf{w}^T\mathbf{z}}}{\left(1+e^{-\mathbf{w}^T\mathbf{z}}\right)^3}\mathbf{z} \mid \mathbf{x}\in\omega_1\right\} - P(\omega_2)\, E\left\{\frac{e^{-\mathbf{w}^T\mathbf{z}}}{\left(1+e^{-\mathbf{w}^T\mathbf{z}}\right)^3}\mathbf{z} \mid \mathbf{x}\in\omega_2\right\}$$

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} \frac{P(\omega_1)e^{-2\mathbf{w}^T\mathbf{z}}\, p(\mathbf{x}\mid\omega_1) - P(\omega_2)e^{-\mathbf{w}^T\mathbf{z}}\, p(\mathbf{x}\mid\omega_2)}{\left(1+e^{-\mathbf{w}^T\mathbf{z}}\right)^3}\, \mathbf{z}\; dx_1 dx_2 \ldots dx_n$$

This integral evaluates to zero when,

$$e^{-\mathbf{w}^T\mathbf{z}} = \frac{P(\omega_2)\, p(\mathbf{x}\mid\omega_2)}{P(\omega_1)\, p(\mathbf{x}\mid\omega_1)} \tag{10}$$

So that,

$$\left[1+e^{-\mathbf{w}^T\mathbf{z}}\right]^{-1} = \left[1+\frac{P(\omega_2)\, p(\mathbf{x}\mid\omega_2)}{P(\omega_1)\, p(\mathbf{x}\mid\omega_1)}\right]^{-1}$$

which is equivalent to the expression in (8). Thus, when the weights of the perceptron are set according to (9), the integral evaluates to the zero vector satisfying the first order necessary condition.

Next, to show that this value is actually a minimum (as opposed to a maximum or saddle point), the second order sufficient condition must be proved by showing that the Hessian is positive definite. Differentiating (6) gives,

$$\nabla^2_{\mathbf{w}} J(\mathbf{w},\mathbf{x}) = -2E\left\{\left[r(\mathbf{x}) - 2u(\mathbf{x}) - 2r(\mathbf{x})u(\mathbf{x}) + 3u(\mathbf{x})^2\right] u(\mathbf{x})[1-u(\mathbf{x})]\, \mathbf{z}\mathbf{z}^T\right\}$$

Substituting for r(x) gives,

$$\begin{aligned}\nabla^2_{\mathbf{w}} J(\mathbf{w},\mathbf{x}) = {} & -2E\left\{\left[1-4u(\mathbf{x})+3u(\mathbf{x})^2\right] u(\mathbf{x})[1-u(\mathbf{x})]\, \mathbf{z}\mathbf{z}^T \mid \mathbf{x}\in\omega_1\right\} \\ & -2E\left\{\left[-2u(\mathbf{x})+3u(\mathbf{x})^2\right] u(\mathbf{x})[1-u(\mathbf{x})]\, \mathbf{z}\mathbf{z}^T \mid \mathbf{x}\in\omega_2\right\}\end{aligned}$$

Substituting the sigmoid and simplifying yields,

$$\begin{aligned}\nabla^2_{\mathbf{w}} J(\mathbf{w},\mathbf{x}) = {} & -2P(\omega_1)E\left\{\frac{e^{-2\mathbf{w}^T\mathbf{z}}\left(e^{-\mathbf{w}^T\mathbf{z}}-2\right)}{\left(1+e^{-\mathbf{w}^T\mathbf{z}}\right)^4}\, \mathbf{z}\mathbf{z}^T \mid \mathbf{x}\in\omega_1\right\} \\ & -2P(\omega_2)\, E\left\{\frac{e^{-\mathbf{w}^T\mathbf{z}}\left(1-2e^{-\mathbf{w}^T\mathbf{z}}\right)}{\left(1+e^{-\mathbf{w}^T\mathbf{z}}\right)^4}\, \mathbf{z}\mathbf{z}^T \mid \mathbf{x}\in\omega_2\right\}\end{aligned}$$

which can be expressed as,

$$\nabla^2_{\mathbf{w}} J(\mathbf{w},\mathbf{x}) = 2\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} \alpha(\mathbf{x})\,\mathbf{z}\mathbf{z}^T\; p(\mathbf{x})\; dx_1 dx_2 \ldots dx_n = 2E\left\{ \alpha(\mathbf{x})\,\mathbf{z}\mathbf{z}^T \right\} \qquad (11)$$

To show that this matrix integral is positive definite, we must show that,

$$\xi^T\left[\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} \alpha(\mathbf{x})\,\mathbf{z}\mathbf{z}^T\; p(\mathbf{x})\; dx_1 dx_2 \ldots dx_n\right]\xi = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty} \alpha(\mathbf{x})\left(\xi^T\mathbf{z}\right)^2 p(\mathbf{x})\; dx_1 dx_2 \ldots dx_n > 0$$

Since $\xi^T\mathbf{z}$ defines a manifold of dimension less than $\mathbf{x}$, $\xi^T\mathbf{z}$ is almost never zero for an arbitrary vector ξ. Since we know p(x) is never identically equal to zero, the problem reduces to showing that $\alpha(x)$ is never identically zero. $\alpha(x)$ in (11) can be written as,

$$\alpha(\mathbf{x}) = -\frac{e^{-\mathbf{w}^T\mathbf{z}}\left[P(\omega_1)\,p(\mathbf{x}\,|\,\omega_1)e^{-\mathbf{w}^T\mathbf{z}}\left(e^{-\mathbf{w}^T\mathbf{z}} - 2\right) + P(\omega_2)\,p(\mathbf{x}\,|\,\omega_2)\left(1 - 2e^{-\mathbf{w}^T\mathbf{z}}\right)\right]}{\left(P(\omega_1)\,p(\mathbf{x}\,|\,\omega_1) + P(\omega_2)\,p(\mathbf{x}\,|\,\omega_2)\right)\left(1 + e^{-\mathbf{w}^T\mathbf{z}}\right)^4}$$

$$= -\frac{e^{-\mathbf{w}^T\mathbf{z}}\left[e^{-\mathbf{w}^T\mathbf{z}}\left(e^{-\mathbf{w}^T\mathbf{z}} - 2\right) + \dfrac{P(\omega_2)\,p(\mathbf{x}\,|\,\omega_2)}{P(\omega_1)\,p(\mathbf{x}\,|\,\omega_1)}\left(1 - 2e^{-\mathbf{w}^T\mathbf{z}}\right)\right]}{\left(1 + \dfrac{P(\omega_2)\,p(\mathbf{x}\,|\,\omega_2)}{P(\omega_1)\,p(\mathbf{x}\,|\,\omega_1)}\right)\left(1 + e^{-\mathbf{w}^T\mathbf{z}}\right)^4}$$

Using the result in (10), this expression simplifies to,

$$\alpha(\mathbf{x}) = \frac{e^{-2\mathbf{w}^T\mathbf{z}}}{\left(1 + e^{-\mathbf{w}^T\mathbf{z}}\right)^4} = \left[P(\omega_1|\mathbf{x})P(\omega_2|\mathbf{x})\right]^2$$

which we know is not identically zero for all nontrivial problems. Thus (11) is always positive definite satisfying the second order sufficient condition.

5.0 Summary

We have shown that minimizing the Mean Squared Error (e.g. by using back-propagation) for a second order perceptron with sigmoid nonlinearity is equivalent to minimizing the Bayes classification error for the two class Gaussian problem. This solution corresponds to the perceptron computing the exact a posteriori probability of class membership giving the optimal measure of classification confidence.

6.0 Acknowledgements

The authors would like to thank Peter Dorato for his assistance in developing some of the proofs presented in this paper.

7.0 References

[1] Asoh, H. and Otsu, N. (1989) "Nonlinear Data Analysis and Multilayer Perceptrons", *Proc. IEEE Int. Joint Conf. on Neural Networks*, June 18-22.

[2] Giles, L. and Maxwell, T. (1987) "Learning, Invariance, and Generalization in High-Order Neural Networks", *Applied Optics*, v. 26, n. 23, pp. 4972-4978.

[3] Giles, C., Griffin, R., and Maxwell, T. (1988) "Encoding Geometric Invariances in High-Order Neural Networks", Neural Information Processing Systems, pp. 301-309, Ed: D. Anderson, AIP, N.Y.

Recognition of Spatio-temporal Patterns with a Hierarchical Neural Network

Takayuki ITO   Kunihiko FUKUSHIMA *
Visual Science Research Division
NHK Science and Technical Research Laboratories
1-10-11, Kinuta, Setagaya-ku, Tokyo, 157 Japan.

1 Introduction

We discuss here a model which has an ability to recognize spatio-temporal patterns like formant patterns of speech signals. Many factors such as difference of speakers, speed of speaking, and noise in surroundings, disturb the aspect of speech signals and they make the speech recognition very difficult. Several neural networks have been developed for recognition of speech and show good performances[1],[2] but they seem not to have overcome these factors fully yet.

We propose a model which recognizes spatio-temporal patterns with an ability to tolerate shift in frequency, deformation in shape, and noise.

2 The Structure of the Model

Fig.1 shows the structure of the model. The model is a multi-layered neural network in which each layer consists of several one-dimensional arrays of neurons along spatial (or frequency) axis. We call the array a "cell-plane". In Fig.1, rectangles drawn with broken lines represent the layers and thin rectangles with thick lines represent cell-planes. The model has two sub-networks, temporal feature extraction sub-network and recognition sub-network.

2.1 Temporal feature extraction Sub-network

The temporal feature extraction sub-network extracts several kinds of features which are specific to temporal patterns such as onsets and offsets of signals, or frequency modulated (FM) portion in the formant pattern. Neurons arranged along spatial or frequency axis (vertical axis in Fig.1) analyze temporal signals in their own position or frequency and extract these features. These kinds of neurons has been found in the early auditory neural

*Present address: Department of Biophysical Engineering, Faculty of Engineering Science, Osaka University, Toyonaka, Osaka 560, Japan

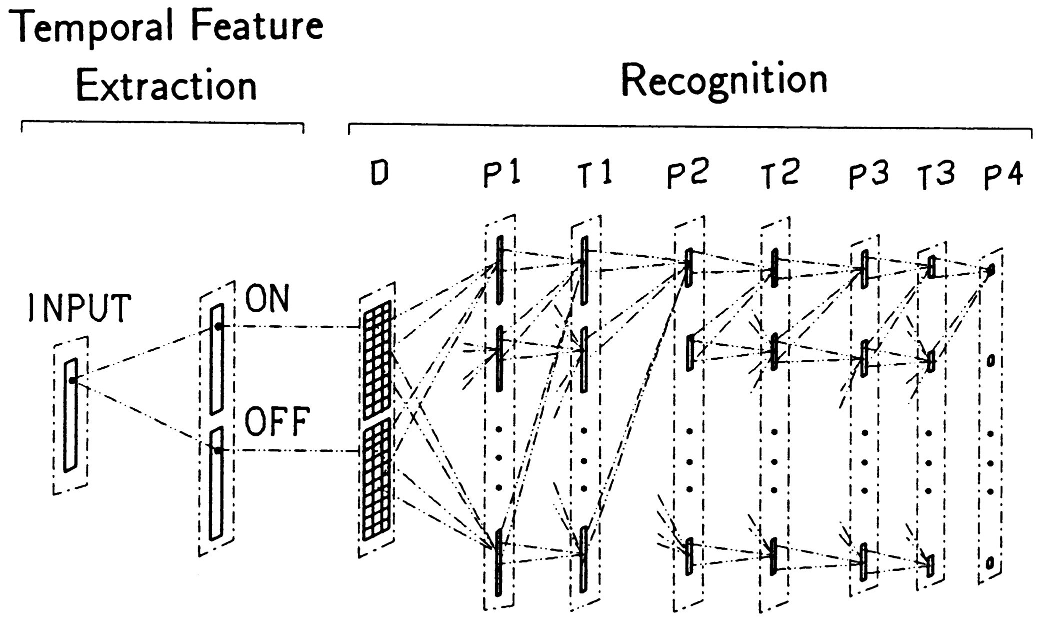


Figure 1: The structure of the model.

pathway[3],[4]. and, based on these physiological data, we have proposed a neural network model which can extract these features from voice signals[5].

In this model, we use a simplified version of the model which extracts just onset and offset of input patterns (Another feature, FM portion will be extracted later in the front end of the recognition sub-network).

2.2 Recognition Sub-network

The recognition sub-network has similar structure to that of a neural network model of visual pattern recognition "Neocognitron"[6]. The input layer is an array of delay-lines which acts as an narrow window to the flow of spatio-temporal patterns, which is denoted by "D" in Fig.1 (the width of the window is three time-units). Then follows a structure in which two types of layers are alternately connected in cascade.

2.2.1 P-cell

One of the two types of layers is a feature extraction layer named "P-cell layer", neurons in which extract spatial local features at each moment. All neurons in a cell-plane have identical connections. This makes it possible that at least one of the neurons extracts the local feature corresponding to the connections whichever spatial position the feature appears. This type of cell is similar to an S-cell in the Neocognitron.

Neurons in the first P-cell layer have connections of size 3 by 3 between input delay-line layer, which can extract local slant lines or frequency modulated signals in spatio-temporal patterns.

2.2.2 T-cell

Another type of layer is a temporal summation layer named "T-cell layer", which has the same number of cell-planes as those of the preceding P-cell planes, corresponding one to one. Neurons in this layer has two functions.

One is to receive responses from neurons of the preceding P-cell plane. Because of these connections of neurons, responses of preceding P-cell plane will be blurred in the T-cell plane, which enables a succeeding P-cell to tolerate positional errors of local features extracted in the preceding P-cell layer.

The other function is to prolong the responses of P-cells with adaptive decay time. It is realized by extra connections between a T-cell and preceding P-cells in other cell-planes. A T-cell shows constant decaying response when the P-cells in other planes have not been showing any responses. If the P-cells have been showing any responses, to the contrary, the T-cell will show bigger and therefore prolonged response because of these extra connections. Because the response of the T-cell itself gates the input from those connections and therefore this enhancement of response happens only when the T-cell itself has been showing some response.

3 Computer simulation

Fig.2 shows a set of training patterns which are moving continuously from right to left and the time course of responses of five neurons in the final layer after finished training. The network was trained layer by layer from lower to higher, presenting the patterns twice for each layer. The learning algorithm used here is a "winner-take-all" rule on which connections of a neuron showing the maximal output will be reinforced and all other neurons in the same cell-plane will get the copy of them. Fig.2 shows that each of the five neurons in the final layer correctly responds to each of the five patterns.

Some examples of responses for deformed patterns are shown in Fig.3. The neurons in the final layer show correct responses to the patterns although they are prolonged or shortened in time domain, or are shifted in frequency (spatial axis), or are contaminated with noise.

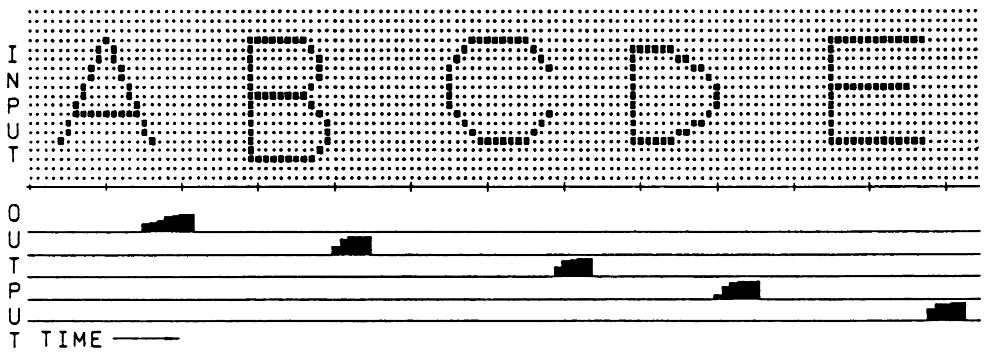


Figure 2: Training patterns and responses of the five neurons in the final layer.

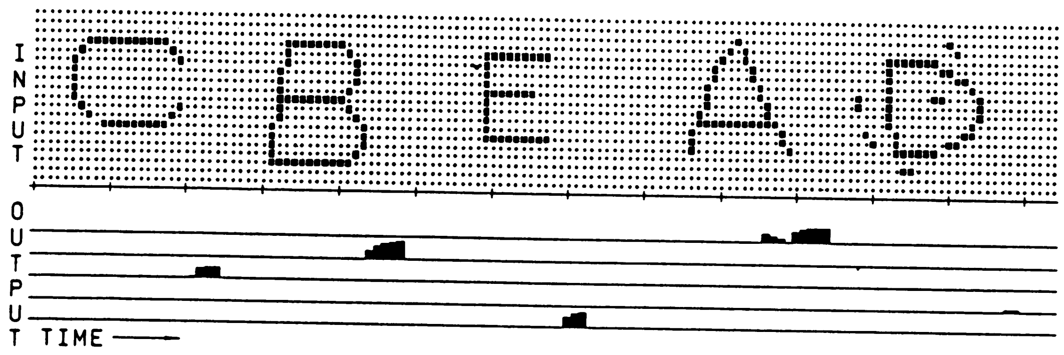


Figure 3: Responses of the model to deformed patterns.

4 Discussion

The model shows an ability to recognize spatio-temporal patterns even though they are shifted in frequency, distorted in shape or contaminated with noise compared with training patterns. The prolonged responses of neurons in T-cell layers and integration in the succeeding P-cell layer endows the network with the ability to recognize temporal patterns even with a very narrow window. Instead these mechanisms might cause a problem of confusing the order of appearing local features. In most cases, this confusion is avoided because the network extracts temporal features such as onsets, offsets, and frequency modulated portions of signals, which detect the cause and effect of signals.

We think that the flexible recognition ability of this model is suitable for speaker independent speech recognition.

References

[1] A.Waibel. Modular construction of time-delay neural networks for speech recognition. *Neural Computation*, 1(1):39–46, 1989.

[2] J.L.McClelland and D.E.Rumelhart. *Parallel Distributed Processing I,II.* MIT Press, 1986.

[3] Y. Katsuki N. Suga and Y. Kanno. Neural mechanism of the peripheral and central auditory system in monkeys. *J. of the Acoustical Society of America*, 34(8):1396–1410, 1962.

[4] M. Nomoto. Discharge pattern of the primary auditory cortex in cats. *Jpn. J. Physiol.*, 15(5):427–447, 1988.

[5] Takayuki Ito and Kunihiko Fukushima. A neural network model extracting features from speech signals. *Systems and Communications in Japan*, 19(3):32–45, 1988.

[6] Kunihiko Fukushima. Neocognitron: a hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1(1):119–130, 1988.

CLUSTERING TAXONOMIC DATA WITH NEURAL NETWORKS

Behzad Kamgar-Parsi
Center for Automation Research
University of Maryland
College Park, MD 20742

J. Anthony Gualtieri
Code 635
NASA GSFC
Greenbelt, MD 20771

1 Introduction

Clustering is an important problem in many field, such as image analysis, astrophysics, biology, etc. By clustering we mean partitioning N points (or objects) into K groups, such that the members of each group are more similar to one another than to those of other groups. The problem can be stated more formally if the objects are represented as points in a D-dimensional metric space: Partition N points among K clusters such that the sum of *within cluster variances* is minimum. Since there are approximately $K^N/K!$ ways of partitioning N points into K clusters [1], when N is large finding the best solution through exhaustive search becomes impractical. Therefore, many heuristic techniques have been developed to tackle this problem (see [5]).

In this paper we present a method for solving the clustering problem with a neural net and investigate its performance. The intuitive motivation for studying the feasibility of neural nets for solving this problem is that clustering involves collecive decision making, as the points must decide among themselves how best to partition themselves, and a neural net with its many connections among the neurons provides a technique for this. Another reason is that by using an analog neural net we embed a discrete optimization problem in a continuous space, the state space of the analog network. This embedding generally helps in finding better solutions.

Our approach here is similar to the Hopfield and Tank solution of the Traveling Salesman Problem (TSP) [4]. This approach has been criticised [11]. We have also found that neural nets are ill-suited for solving the TSP for which the network must satisfy strongly conflicting constraints [7], but in solving the clustering problem, where costraints are much easier to satisfy, the results have been quite successful [6].

Here we apply our neural net algorithm to clustering Anderson's data (tabulated in [2]) on three species of Iris. There each Iris flower is characterized by four attributes, the width and length of petals and the width and length of sepals. This 4-dimensional data set is frequently used as a test of clustering algorithms. We compare the neural net results with those obtained with an often used K-means algorithm proposed by Forgy [3].

2 The Neural Net

We want to partition a set of N points in a D-dimensional space into the *best* K clusters – best in the sense that sum of the squares of the distances of the points from their respective cluster centroids (i.e. sum of *within cluster variances*) is minimized. We formulate the problem in terms of minimizing an energy function, which can be mapped on a neural net.

The energy function will consist of two parts: (i) constraint terms which make certain a point, at the end of the search, belongs to one and only one cluster; and (ii) the cost term which is the sum of the residuals and is the function we actually wish to minimize. We assign K neurons to each data point, i.e. the network has a total of $n = K \times N$ neurons. We denote the neurons by two indices (p, i), where

$p = 1, \cdots, K$ and $i = 1, \cdots, N$ refer to clusters and points, respectively. The activity V_{pi} of neuron (p, i) then represents the strength of the hypothesis that point i belongs to cluster p. The energy of a neural net that can solve this clustering problem can be written as (for more details see [6])

$$E = \frac{A}{2}\sum_{i=1}^{N}\sum_{p=1}^{K}\sum_{q\neq p}^{K} V_{pi}V_{qi} + \frac{B}{2}\sum_{i=1}^{N}(\sum_{p=1}^{K} V_{pi} - 1)^2 + \frac{C}{2}\sum_{p=1}^{K}\sum_{i=1}^{N} R_{pi}V_{pi}^2. \tag{1}$$

Here the coefficients A, B, and C are positive constants. The A- and B-terms together enforce the constraints (or syntax). During the network's search for a solution all points belong to all clusters because in general all neurons are partially active, i.e. $0 < V_{pi} < 1$. However, at the end of the search the network must reach a state that unambiguously decides which point belongs to which cluster, i.e. if point i belongs to cluster p then $V_{pi} = 1$ and the rest $V_{qi} = 0$, $q \neq p$. When a valid solution that satisfies the syntax is found the A- and B-terms become zero, and the energy E reduces to just the C-term, which is the cost and is the function we actually wish to minimize. Therefore the deep minima of E correspond to good solutions, and the deepest minimum to the best solution. Here R_{pi} denotes the square of the distance of point i from the centroid of cluster p (i.e. the residual) and is given by

$$R_{pi} = \sum_{\alpha=1}^{D}(x_{\alpha i} - X_{\alpha p})^2 \quad \text{with} \quad X_{\alpha p} = \sum_{i=1}^{N} x_{\alpha i}V_{pi} / \sum_{i=1}^{N} V_{pi}, \tag{2}$$

where $x_{\alpha i}$ are the coordinates of point i, and $X_{\alpha p}$ are those of the centroid of cluster p. Here we have chosen the Euclidean distance as the metric, but one can define any metric one wants.

The network dynamics, obtained from $-\partial E/\partial V_{pi}$, are described by

$$\frac{du_{pi}}{dt} = -\frac{1}{\tau}u_{pi} - A\sum_{q\neq p}^{K} V_{qi} - B(\sum_{q=1}^{K} V_{qi} - 1) - CR_{pi}V_{pi} + I_{pi}. \tag{3}$$

Here u_{pi} is the input signal of neuron (p, i), τ is the self-decay time of the neuron and I_{pi} is the external bias. For the gain function of the neurons we have taken the standard sigmoid function which is

$$V_{pi} = \frac{1}{2}(1 + \tanh\frac{u_{pi}}{u_0}), \tag{4}$$

where u_0 is the steepness of gain.

3 Results and Discussion

To simulate the performance of the network in finding solutions we choose an intial state at random and let the network evolve according to the equations of motion (3). Since the energy of an analog neural net is Lyapunov the network evolves towards states of lower energy and stops eventually once it finds a solution. Among the variety of techniques for solving initial value ordinary differential equations (e.g., see [8]) we have used the Euler method.

For simulations we let $\tau = 1$, so that the convergence times are measured in units of τ. We have chosen the following values for the parameters of the energy function: $A = B = 1$, $C = 1/R_{avg}$, all $I_{pi} = 1$. These parameter values are obtained from analyzing the stability of valid solutions. Scaling the parameter C with the average residual R_{avg} is necessary to ensure good solutions, because as the network evolves, the residuals become generally smaller and the cost term becomes less effective in driving the network toward good solutions. This rescaling of parameter C keeps the cost term of the same order of magnitude as the syntax terms. The results are not overly sensitive to these parameter values. However, the network appears to be more sensitive to the steepness of the gain function, u_0. The results we give here are with $u_0 = 0.01$.

In computationally hard problems one usually finds several solutions and then takes the best one as *the solution*. Since with a neural net one cannot still guarantee finding the best solution (although it enhances the chance of finding good solutions or the best solution), one has to run the network several times. The results given in the Table below are obtained from 100 trials. K-means refers to Forgy's iterative algorithm [3]. We have used the 4-dimensional Anderson data on three species of Irises [2].

	Neural Net				K-means		
K	Best	χ^2_{avg}	Synt%	Time	Best	χ^2_{avg}	Iter
2	100	152.3	100	0.9	100	152.3	6
3	100	78.9	85	2.4	78	93.7	9
4	92	58.1	93	1.5	7	65.3	11
5	7	49.1	78	3.0	0	54.4	9

The results obtained with neural net are consistently better than those found by the K-means algorithm in two respects: (i) The network finds the best solution more frequently (Best column in the Table is the percentage of the 100 trials yielding the best solution), and (ii) the average sum of within cluster variances, χ^2_{avg}, are significantly lower for $K = 3,4,5$ for the neural net indicating that it has a better chance of finding good solutions. For $K = 2$ the clusters found are well separated and both methods find identical solutions.

The Time column in the Table gives the average convergence times in units of τ, the characteristic decay time of neurons. These are order of magnitude estimates, because they depend on various parameters such as the time step size in Euler method, and the convergence criterion. Note that the convergence time of the network should not be compared directly to the average number of iteration for K-means. For example, when $K = 3$ the convergence time of 2.4 requires 2400 network updates (since time step was 0.001), while K-means needs only 9 iterations. Hence, the neural net algorithm is much more expensive to run on regular machines than the conventional K-means. They become practical only when they are mapped on analog VLSI neural nets, which are expected to have processing time $\tau = 10^{-3}$ to 10^{-6} sec [9].

The Anderson data consists of 150 samples, 50 samples from each of the 3 Iris species. In Figure 1 we show two 3-D projections of the unclustered points and in Figure 2 we show the best cluster found by the network for $K = 3$, also as 3-D projections. The data points in Figure 2 are connected to their respective cluster centroids. The samples from one of the species are well clustered, but the other two clusters are close to each other and there is partial overlap between them. Because of the overlap correct classification of the samples has been problematic. For example, Fisher [2] uses the information about the original data to define a different metric, and Tukey and Tukey [10] use the projection pursuit technique, both with limited success. Our method, too, fails to find the true clusters, for the reason that the true clusters have $\chi^2 = 89.3$ which is not the optimum solution found by the network with $\chi^2 = 78.9$. To find the true clusters one has to have additional information about the shapes of clusters. In the absence of such information we have assumed the simplest shape possible, i.e. convex compact clusters as reflected in the metric (2). In case shape information is available one can add additional terms in the energy function to encourage clusters of a particular shape.

Another important question is: How many clusters? We had found previously [6] that the percentage of times the syntax is satisfied is a a good indicator of the presence of background noise. Our motivation here was to see if there exists a trend in the syntax percentage as we vary the number of clusters. There does not appear to be such an indicator. However, efforts are under way to incorporate the number of clusters directly into the energy function.

References

[1] W. Feller, *An Introduction to Probability Theory and Its Applications* (John Wiley, 1959) Vol. 1, p. 58.

[2] R.A. Fisher, *The use of multiple measurements in taxonomic problems*, Annals of Eugenics **7**, 179 (1936).

[3] E.W. Forgy, *Cluster analysis of multivariate data: efficiency versus interpretability of classifications*, Biometric Soc. Meetings, Riverside, California. Abstract in *Biometrics*, **21**, 768 (1965).

[4] J.J. Hopfield and D.W. Tank, *Neural computation of decisions in optimization problems*, Biological Cybernetics **52**, 141 (1985).

[5] A.K. Jain and R.C. Dubes, *Algorithms for Clustering* (Prentice-Hall, 1988).

[6] B. Kamgar-Parsi, J.A. Gualtieri, J.E. Devaney, and B. Kamgar-Parsi, *Clustering in parallel with neural networks*, submitted to Biological Cybernetics (1989).

[7] B. Kamgar-Parsi and B. Kamgar-Parsi, *On problem solving with Hopfield neural nets*, submitted to Biological Cybernetics (1989).

[8] M.V. Mascagni, *Numerical methods for neuronal modeling*, in *Methods in Neuronal Modeling*, edited by C. Koch and I. Segev (MIT Press, 1989), Ch. 13.

[9] C. Mead, *Analog VLSI and Neural Systems* (Addison-Wesley, 1989).

[10] P.A. Tukey and J.W. Tukey, *Data-driven view selection; agglomeration and sharpening*, in *Interpreting Multivariate Data*, edited by V. Barnett (John Wiley, 1981) p. 215.

[11] G.V. Wilson and G.S. Pawley, *On the stability of the Travelling Salesman Problem algorithm of Hopfield and Tank*, Biological Cybernetics **57**, 63 (1988).

Fig. 1

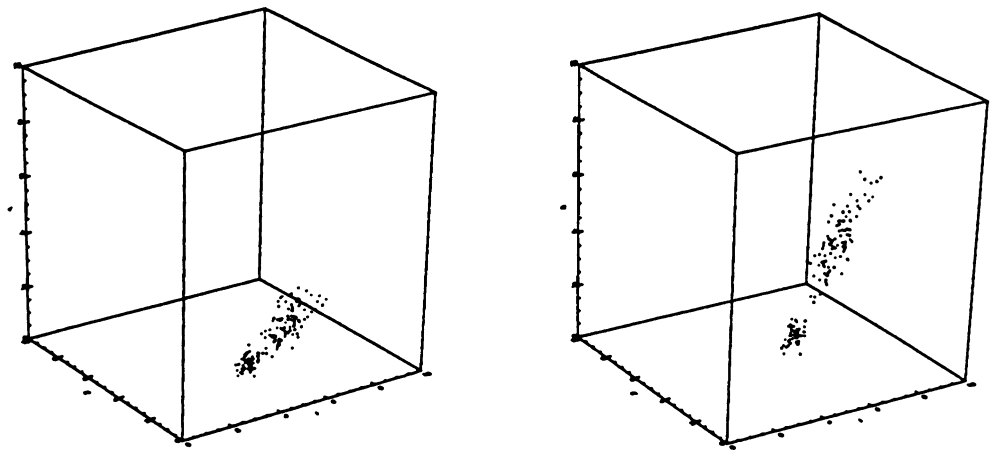
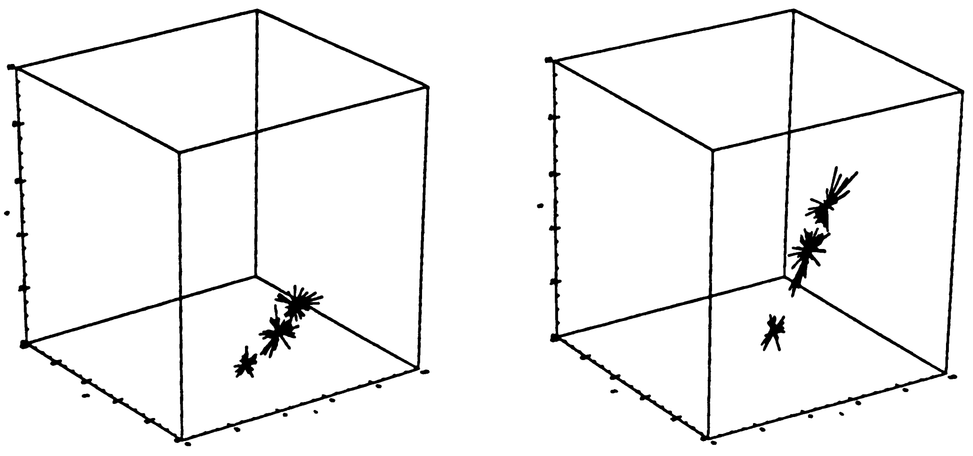


Fig. 2



Reproducing Infinite Boolean Sequences: an Application of Hidden Markov Models to Connectionist Learning

A. Kehagias
Division of Applied Mathematics
Brown University
Providence, RI 02912

E-Mail Address: st401843@brownvm.bitnet

Abstract

Given the first few terms of an infinite sequence of 0's and 1's, build a network that reproduces the rest of the sequence. To accomodate this learning task, a framework is developed for learning, general enough to include learning of finite-length input, finite-length output as well. The similarities between the finite and infinite learning tasks are considered and parallels are drawn to the speech recognition problem. Several properties of infinite length learning are obtained, using Hidden Markov Model theory.

1. Introduction: Consider the following learning problem: build a network that, given the first few terms of an infinite sequence of $0's$ and $1's$ (call such a sequence a **Boolean** sequence), will output the rest of the sequence. The network, will be trained on one or several finite (but possibly quite long) valid sequences.

This problem is similar to a standard connectionist learning problem: given input/output pairs train a network that will, when presented with an input, produce the correct output. In this paper I will propose a learning framework that accommodates both the finite and infinite case; also stochastic and deterministic learning by connectionist networks (henceforth CN) can both be tretated. I will argue that the stochastic case is more general and interesting.

An apology is due: because of the very limited space, I have only presented the general rationale behind the theory and some of the theoretical results, but without proof. The proofs and very encouraging computational results will be reported separately.

2. Learning by Computing Probabilities: The pattern learning problem can be phrased as follows: given a **learning universe** $L \subset U \times Y$ (i.e. a set of input/output pairs (u, y)), build a network (and train it) that, when given an input $u \in U$, will output the 'correct' $y \in Y$.

The concept of 'correctness' is generally not unambiguous and, in fact, this is an advantage. E.g., given an input, it is possible that more than one outputs would pass as 'correct' in some sense. Also, given an unambiguous training set $T = \{(u, y)\}$, such that we can build a network that will reproduce the training pairs exactly but will generalize poorly [Huy]. In fact, the power of generalization that CN's exhibit can be partly attributed to their tolerance of errors.

Let us then change the learning problem so that more than one correct answer is possible. The entity to be learned is a conditional probability measure $P(\mathbf{y} \mid \mathbf{u})$, where $\mathbf{u}$ is a $\mathcal{U}$-measurable random variable $\mathbf{y}$ is a $\mathcal{Y}$-measurable random variable, $\mathcal{U}$ is a σ-algebra on U, $\mathcal{Y}$ is a σ-algebra on Y. $\mathbf{u}$, $\mathbf{y}$ are the input and output random variables; the idea is that, given some input, the output is a random variable with probability conditioned on the input. This prompts the following:

Definition 1 *Given random variables* $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ *on an appropriate probability space.* $\mathbf{x}$ *and* $\mathbf{y}$ *are said to be* $\mathbf{z}$ **input equivalent** *iff* $Prob(\mathbf{x} \mid \mathbf{z}) = Prob(\mathbf{y} \mid \mathbf{z})$.

3. Network Architecture: We will consider connectionist networks, that is, networks of numbers of simple, identical, interconnected units. The building units will be **Stochastic Threshold Units** (STU). These units have the following input/output law:

$$Prob(\mathbf{y} = 0 \mid \mathbf{u} = u) = Prob(\eta > u).$$

Here η is a random variable distributed uniformly in $[a, b]$. By proper choice of a, b we can get deterministic output. A network built of such units accepts an input u and produces output y with probability $Q(\mathbf{y} \mid \mathbf{u})$. As the output is exclusively 0/1, call such a network a **Boolean** CN. The network is, in general, stochastic. Of course, nothing precludes that Q is a trivial measure, concentrating mass 1 to one element y and zero to all others (deterministic case). It is useful to think of the connectionist network itself as equivalent to the conditional probability measure $Q(\mathbf{y} \mid \mathbf{u})$.

4. Learning Finite Output: The finite input/output problem has been studied by many researchers, usually as a deterministic problem. (But see [Bar], [Pea], [Lin], [Sol] for probability learning approaches.) In essence we want to learn a Boolean function from a subset of its graph. The introduction of deterministic **Linearly Graded Units** (LGU's) offers more flexible solutions but still an exponentially large number of units and exemplars, is needed for perfect learning [Huy]. Once again, the argument for stochastic learning presents itself: if exact learning requires oversized networks, learning the statistical properties of the output (which ispossible on a smaller network) is preferrable. Indeed, I submit that the most succesful case of discrete input/output learning is done by connectionist methods but outside the traditional connectionist research community. The example I have in mind is speech recognition by Hidden Markov Models (henceforth HMM) [Lev]. Here exact learning of the input/output relationship is out of the question (because the input/output function is unknown and the training set is too large for exact, deterministic learning). The HMM modellers have resorted to a probabilistic model, which can produce rich input/output behavior with few connections exactly because each unit can produce probabilistically many different outputs. On the other hand, I have argued elsewhere [Keh] that the HMM is exactly analogous to a connectionist network and can be implemented as one. Set the intial state of a HMM to **u** and let it run for a finite time N. The state sequence of the underlying Markov Chain $\mathbf{y} = \mathbf{y}_1, ..., \mathbf{y}_N$ is a random variable. By identifying sequences of outputs of a Boolean CN with the (finite) number of states of a HMM, we can easily prove the following:

Theorem 1 *To every Hidden Markov Model corresponds a* **u** *input equivalent connectionist network of STU's.*

As for the training algorithm: back propagation is an option, the backward-forward HMM algorithm is another [1]. In fact any kind of descent algorithm that utilizes gradient information will benefit from a forward-backward derivative-computation. This follows from the staged character of the architectures we have proposed and the use of Lagrange multipliers optimization [Keh]. It is also true of time-evolving networks which we will use for the infinite output case.

5. Learning Infinite Output: Consider the infinite output case. The STU-networks that we have been considering can be in an on or off state. It follows that the complete state of an M-unit network can be characterized at any given moment by an M-long **state** vector of 0's and 1's, indicating which units are on and which off. Evolution of the state vector in time is described by the network acting on the initial state vector[2]. Time is not of great importance in the finite case, because the network is **static**. However, if we want an infinite output, we must build an infinitely large network or else trade space for time and obtain the output sequence as the output of a time evolving network[3]. The problem of learning an infinite sequence is the following: we are given the initial part of one or more valid sequences and we want to build a network that when presented with the beginning of a sequence will produce correct (infinitely long) output. The natural way to think about the problem is that of building a network that will accept an input and then will run on its own until infinity, producing output. This is like a dynamical network where we specify the

[1] Generalization of the Backward-Forward algorithm to CN's is possible: results will be reported elsewhere.

[2] Another useful state-network characterization is the one based on the state probability vector. This will not be presented here, for brevity.

[3] Such networks are called **recurrent**; they have been considered in the literature [Pin], but usually only as they evolve to a steady state, that is, as all units settle down to either 0 or 1 output for all time.

initial conditions[4]. In the proposed general framework, the input is the initial states. The learning task is, in general, to learn a probability measure $P(\cdot \mid u)$, where u is the input. It would be nice if the output y_n at time n depended on a finite number of past outputs.

$$Prob(\mathbf{y}_n = \tau \mid \mathbf{y}_{n-1} = \sigma_1, ..., \mathbf{y}_1 = \sigma_{n-1}) = f(\sigma_1, ..., \sigma_m) \tag{1}$$

Here f is a Boolean function of m arguments. This is a highly desirable situation, as we can take this sequence as the output of a STU-network with, at most, However its also conceivable that

$$Prob(\mathbf{y}_n = \tau \mid \mathbf{y}_{n-1} = \sigma_1, ..., \mathbf{y}_1 = \sigma_{n-1}) = f(\sigma_1, ..., \sigma_{n-1}) \tag{2}$$

The network as described by equation (1) is an m-th order Markov chain, equivalent to an HMM, which can be trained by the powerful Backward-Forward algorithm. It can also be implemented as a STU-network with at most $O(2^m)$ units. We will also use the term **Boolean Dynamic Network**. The next question is whether the sequence is determinstic or stochastic; consider the case where we are given a finite part of a sequence and must compute the next terms. We do not know whether it is stochastic or deterministic. Of course the following fact is true:

Theorem 2 *For every deterministic Boolean dynamical network with output $y_1, y_2, ...$, $\exists$ N such that y_{N+1}, y_{N+2}, ... is periodic*[5].

However, the period may turn to be exponentially long, so given a relatively short training sequence lack of periodic behavior does not determine whther the inifinite sequence is deterministic or stochastic. That is, we may see any finite substring of 0's and 1's followed by either a 0 or a 1; this would still not show that the sequence is stochastic. In short, we have no basis to decide whether the sequence is deterministic or stochastic and the reasonable assumption is to try to model it as stochastic.

Definition 2 *A* **aggregate** *of a stochastic process $\{X_n\}$ is another process $\{Y_n\}$ s.t. $Y_n = f(X_n)$.*

A HMM is a aggregate of a Markov chain; the output of a connectionist network is a aggregate of the state vector and, since the state vector is a Markov chain, the output of the connectionist network is the observable of a Hidden Markov Model.

Definition 3 *A* **Boolean stochastic process** *is a probability measure on the set of sets of infinite sequences of* 0*'s and* 1*'s (call every such sequence a* **realization** *of the stochastic process), that is, $P(\mathbf{x} = s_1 s_2 ....)$.*

With each realization we can associate a number s in the interval $[0, 1]$; we do this by taking $s = .s_1 s_2 ...$ in binary notation. [6] The measure P is thus well defined on dyadic intervals and can be extended by Kolmogorov's Theorem on the Borel sets in $[0, 1]$. Hence the stochastic process $\mathbf{x}$ is equivalent to a a random variable $\mathbf{x}$ taking values in $[0, 1]$, or, which is really the same thing, to a probability measure on $[0, 1]$. As the CN produces a stochastic process depending on the initial conditions, by the same process as the one outlined above, we have a conditional probability measure $P(\cdot \mid u)$ associated with each input u.

Now we can apply the same steps as in the finite learning task: we must find a probability measure $Q(\cdot \mid \cdot)$ that will approximate $P(\cdot \mid \cdot)$. Many approximation schemes are available, but it is useful to establish first that an arbitrarily close approximation is possible.

Theorem 3 *Given a Stochastic Process, that is, a measure on $[0, 1]$, it can be weakly approximated by a sequence of Hidden Markov Models.*

[4]We can go up one degree of generality: Imagine an infinite input as well an infinite output. This would correspond to a network where we have input at every time instant. We can certainly build such a network. I do not give details here, for lack of space, but also because it turns out many of the properties of the input network will be similar to the no-input network. However there are interesting peculiarities, too. Roughly, the no-input network corresponds to a free dynamical network and the input network corresponds to a control network.

[5]There is a subtlety here: the actual evolution will depend on the initial conditions. So for different initial conditions the network may get attracted to a different periodic sequence .This is similar with limit cycles in real-valued nonlinear dynamical systems, as well as with non-ergodic Markov chains.

[6]The correspondence is not one-to-one. However, for all interesting cases, we can turn it to one-to-one by dropping out realizations of probability 0.

Given that there exist HMM's and hence CN's that can approximate a stochastic Boolean sequence arbitrarily close, how does one go about finding them and, probably most important, how does one train them? I have used several schemes; one, call it **structured**, is the following: the training sequence is used to estimate the joint probabilities of $x_1, x_2, \ldots, x_{m+1}$, for some prechosen m; then from these the transition probabilities $P(x_{n+m} \mid x_{n+m-1}, \ldots, x_n)$ are computed. It is easy to implement a CN that has $2^m + m$ STU's and the above computed transition probabilities. The training of such a network consists in updating the estimates of the transition probabilities as new data is collected; when these probabilities are known the weights of the connection can be immediately computed. In this respect it is important to have ergodicity of the teaching sequence, or else samples from all the ergodic elements. This architecture yields large but easily trainable networks; yet output is pretty good even with rather small values of m. Another possibility explored is less structured architectures where a more or less random choice of connectivity is made and then weights are computed by some backpropagating algorithm. The actual criterion used was either cross entropy between desired and actual probability function, or the following weighted quadratic criterion:

$$J = \sum_{n=1}^{N} \frac{(P(z_n) - Q(z_n))^2}{P(z_n)}. \tag{3}$$

Results were again quite good, with no noticeable differences across learning algorithms and criteria.

One last point must be raised: we are essentially looking at high order sequences and attempt to approximate them by aggregates. This alleviates certain problems (estimation, size of networks) but, inevitably, produces some faulty sequences. The degradation in terms of spurious sequences being produced by the approximating network can be quantified in terms of the entropy H of the model as compared to the entropy of the original sequence. This is the subject of the following Theorem; remember that an approximation is a aggregate of the original sequence.

Theorem 4 *If a stochastic process* $\mathbf{x}$ *is a aggregate of a stochastic process* $\mathbf{y}$*, then* $H(\mathbf{x}) \geq H(\mathbf{y})$. *If* $\mathbf{y}_n \rightarrow \mathbf{y}$*, then* $H(\mathbf{y}_n) \downarrow H(\mathbf{y})$.

This is yet another way of saying that there are approximating networks that will approximate the original sequence arbitrarily well.

6. Conclusion: I have presented an outline of a framework for learning that involves infinitely large learning universes. The basic idea is that as the learning(and training) universe get large it is not feasible to learn deterministically, especially when it is not clear that the objects being learned behave deterministically. Therefore stochastic learning methods are proposed; several theoretical results indicate that stchastic learning is feasible and practical. Computational results are also very encouraging, but were not presented for lack of space.

REFERENCES

[Bar] Barto A.G. et al., *Pattern-Recognizing Stochastic Learning Automata*, IEEE SMC, Vol.15, No.3, May 1985.

[Huy] Huy, K.A. and Horowitz, M.A., *Generalization in Connectionist Networks that Realize Functions*, Proc. of the Pittsburgh Connectionist School, Morgan Kauffman, 1988

[Keh] Keh, Ath., *Optimal Control for Training: The missing Link between Hidden Markov Models and Connectionist Training*, Brown University, Div. of Appl. Math. Preprint, 1989

[Lev] Levinson, S.E. et al., *An Introduction to the Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition*, Bell Sys. Tech. J., Vol. 62, No.4, April 1983

[Lin] Linsker, R. *Self-Organizing in Perception Networks*, IEEE Computer,Vol.21,No.3, March 1988

[Pea] Pearlmutter,B. and Hinton G., *G-Maximization: An Unsupervised Learning Procedure*, Neural Networks for Computing, AIP Conf. Proc. 151, 1986

[Pin] Pineda, F., *Generalization of Back-Propagation to Recurrent Neural Networks*, Phys. Rev. Let., Vol.59, No.19

[Sol] Solla, S. *Accelerated Learning Experiments in Layered Neural Networks*, Complex Systems, Vol. 2, 1988

GRAMMATICAL INFERENCE AND NEURAL NETWORK STATE MACHINES

Y.D. Liu, G.Z. Sun, H.H. Chen, Y.C. Lee
Laboratory for Plasma Research
Dept. of Physics and Astronomy
and
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742

C. Lee Giles
Air Force Office of Scientific Research

Recently, much attention has been paid to the studies of recurrent neural networks to perform sequential data processing tasks [1] [2] [6] such as sequential symbol predictions, isolated and continuous speech recognition, and robotic arms control etc. In general, it is believed that a recurrent neural network is much more powerful than a network without recurrent connections due to its ability to form internal states and to form transition loops among states. These internal states hold informations about the past history of the input strings and allow the recurrent neural network to react in accord with the full history of the input string regardless of its length. In this regard, the recurrent nueral network is acting like a sequential state machine [3] In models with discrete neuron states as in the McCulloch-Pitts threshold unit, the number of states is always finite and a neural network is equivalent to a finite state machine. Recurrent connections would then allow the finite state machine to have loops which would endow it the ability to process sequential data of arbitrary length. In the case of analog neurons however, it is not yet clear how many internal states could be represented by a fixed number of neurons. It neverthless is a crucial question needs to be answered to allow us a decision on the number of neurons and weights needed to simulate a particular state machine. On the otherhand, it is well known from the theory of computing [4] that finite state machine is but the simplest kind of a hierarchy of sequential machines. These more sophiscated machines contain a finite state machine as a controller with an additional infinite memory tape or stack.

The language recognized by a finite state machine is called a regular language. Other more involved language such as context free and context sensitive language would only be recongnized by a push down automaton or a Turing machine.

In this paper, we would like to discuss the possibility of using recurrent neural networks to conduct the task of regular grammar inference [5]. That is, from a finite set of positive and negative examples, the neural network has to learn the task of recognizing the legality of new examples, usually of arbitrary length. It is interesting to note that in case the neural network learned the inference task the generalization power would in effect be infinite since the size of the training set is finite and the test set could be infinite. This is usually the case after we quantized the learned weights and the internal states to obtain the finite state machine from the neural network.

As an example, let us consider the dual parity problem. A binary string of arbitrary length is considered legal and therefore accepted by a finite state automaton if it contains an even number of 0´s and an even number of 1´s. Otherwise it is an illegal string. The task for the neural network is to learn from a few examples to recognize the legality of new strings.

It is obvious that this problem cannot be handled by a feed forward network since it can only accomodate a time window of fixed size and thus cannot handle sequential patterns with various length. In a recrurrent network, the iteration of the neurons can be viewed as transitions among internal states represented by the activity patterns of the neurons. Starting with a fixed initial state, the network is iterated with an input string presented sequentially through a few additional input neurons to the recurrent state neuron. At the end of the input string an end symbol is presented and the activity of a particular state neurons is taken to indicate the "acceptance" (or rejection) of the input string if this activity is close to "1" (or "0").

Using second order connections, the recurrent formula for the neurons can be written as

$$S_i^{t+1} = g(\Sigma_{j,k} W_{ijk} S_j^t I_k^t + \theta_i) \quad (1)$$

where S_i^t is the activity of the i_{th} neuron at time step t, I_k^t is the k_{th} component of the input symbol at time t, and g is the sigmoid function for analog soft neurons. In our system, we use three state neurons and three input symbols (0,1 and "e" the end symbol).

An online real-time learning algorithm for the network similar to that of Williams and Zipser [6] was derived in which the weight change (error prediction) is propagated forward and updated at every time step. At the end of the input sequence, the predicted error is compared with the actual error obtained through the classification information and updating of the connection weight made only at this time. This procedure therefore departs significantly from other works that trained state machines with supervision at each time step.

In our opinion, we are training the neural network to self organize into a finite state machine structure (transition rules and internal states unknown beforehand). This is significantly different from simulating a given finite state machine by providing information regarding the number of states and their transition rules. As a matter of fact, there is a well-known receipe to construct a neural network to simulate a given finite state machine [3]. However, we should note that the constructed neural network is of higher order type as indicated in Eq. (1). We can show that networks with only binary connections cannot simulate "all" finite state machines. It is the main reason why we choose higher order network for our grammatical inference task.

For the training samples we begin with all the binary strings shorter than a length of 4 digit. There are 30 of them. After these were learned, we tested the system on some new strings with longer digits. Usually, this first round of learning is not perfect. We found a few incorrect recognitions of strings in length 6,7,8 etc. These strings were then added to the training set for the second round of training. After three or four rounds of training, the total number of training patterns would be accumulated to around fifty and the learned network performs almost always perfectly (systematically testing strings of length up to 20). Usually each round of training requires less than 500 sweeps of presentation to converge.

Since we use analog neurons and weights, it is actually impossible to conclude that the trained recurrent network would not make errors in its recognition of an arbitrarily long strings (with length >> 20, there are more than 10^8 binary strings with length 20). However, after we quantized the internal states to 1 bit acuracy, we found a discrete finite state machine with four states as shown in Fig. 1. Furthermore, if we also quantize the weights to 1 $\frac{1}{2}$ bit per weight, we found the network state machine remains intact. The learned network is rather robust indeed.

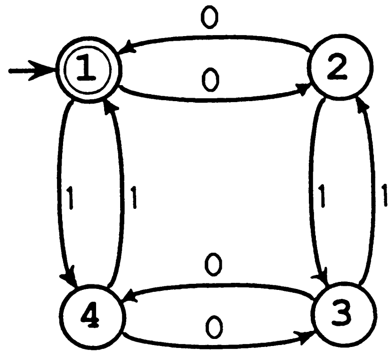


Fig. 1: A four state Machine learned from examples, state 1 is both the start and the final state.

As mentioned earlier, there is no guarantee that a recurrent network with "binary" connection weights would learn a finite grammar. As a matter of fact, one can easily show that the 4 state machine in Fig. 1 cannot be implemented with the "binary" network. Nevertheless we found that a binary network did learn to recognize the dual parity string. The machine has 16 states. It enjoys much less fault tolerance and requires much longer time to learn (about 4 or 5 times longer).

As a curiosity, we also tested the inference of nonregular grammars on the recurrent neural network without any external stack or tape memory. An example is the problem of parenthesis checking. The parenthesis checking problem is a problem of context free grammar which cannot be solved in general by any finite state machine. Nevertheless if we limit the length of the string then a finite state machine may be expected to handle it. Using the neural net with the same topology as for the dual parity problem, we found it can recognize almost all strings up to length 10 (with only two errors). However the internal states structures are vey irregular. We cannot extract a finite state machine from it.

In conclusion, we have trained recurrent neural network to do inference on regular grammar. A real-time learning algorithm can lead the network to form a finite state machine automatically. Second order connections are most natural for this problem since a constructive reciepe exists for a second order neural network to simulate any given finite state machine. First order network is not able to implement certain finite state structure and is also more difficult to train in general. Much more work are needed to clarify the question regarding the following: How large a finite state machine can be implemented in a given network? How many different finite state machine can be implemented in a given network topology? How to deal with inference problems with more sophiscated grammers such as context free and context sensitive grammers? etc. As for the last question we have constructed a neural network push down automaton to recognize simple context free grammar as reported in a separate paper [7].

Acknowledgment:

This work is sponsored in part by AFOSR and NSF.

References

(1) Jordan, M.I. Attractor Dynamics and Parallellelism in a Connectionist Sequential Machine. Proceeding 8th Annual Conf. Cognitive Science Soc. 531 (1987)

(2) Elmour J.L. Finding Structure in Time (CRL Technical Rep 8801) La Jolla, Univ. of Calif, San Diego, Center for Research in Language (1988).

(3) Minsky M.L. Computation: Finite and Infinite Machine, Prentice-Hall, N.J. (1967).

(4) Hopcroft J.E. and Ullman, J.P. Introduction to Antomata Theory, Languages, and Computation, Addison Wesley (1979).

(5) Fu K.S. and Booth T.L. Grammatical Inference: Introduction and Survey IEEE Transactions on Pattern Analysies and Machine Intelligence, Vol. 8, P.343 (1986).

(6) Williams R.J. and Zipser D. A Learning Algorithm for Continually Running Fully Recurrent Neural Network ICS Report 8805, Institute for Cognitive Science, Univ. of San Diego (1988).

(7) Sun G.Z., Chen H.H. , Giles C.L., Y.C. Lee, and Chen D. Connectionist Pushdown Automata that Learn Context-Free Grammars.

A Comparison of a Neural Network Based Estimator and Two Statistical Estimators in a Sparse and Noisy Data Environment*

Reza Shadmehr[1] and David Z. D'Argenio[2]

[1]Dept. of Computer Science & [2]Dept. of Biomedical Engineering
University of Southern California
Los Angeles, CA 90089-0782

Gorman and Sejnowski [3] demonstrated that a learning algorithm such as back-propagation in a multilayered neural network (NN) can be an alternative to traditional pattern classification techniques. In their study, a trained network's performance was compared with that of a nearest neighbor classifier, and they found that the NN could classify input signals somewhat better than the latter technique. Since the probability of correct classification for the nearest neighbor technique can be used to obtain upper and lower bounds on the Bayes probability of correct classification [1], the performance of the network may have approached that of a Bayes decision rule, which is an optimal classifier. This hypothesis is interesting since Bayesian estimators require *a priori* knowledge regarding the underlying statistical nature of the classification problem, and simplifying assumptions must be made in order to apply such estimators in a sparse data environment. A comparison of the two techniques would be valuable since NNs have the advantage of requiring less restrictive assumptions in representing an unknown system, making them potentially easier to apply in a sparse (and usually noisy) data environment.

It is in such an environment where we have compared the performance of a NN estimator to that of two statistical techniques: a maximum likelihood (ML) estimator, and a Bayesian maximum *a posteriori* probability (MAP) estimator. The estimation problem was that of predicting model parameters of a pharmacokinetic system (*e.g.*, systems that define absorption-elimination dynamics of drugs in patients). In this area, Bayesian estimators have been successfully applied [6], partially due to the fact that a prior distribution of the unknown parameters is generally available.

In addition to the parameter estimation problem, the estimators were asked to predict the noise present in each of the measured data samples. We compared the noise model developed by the NN with that of the ML and the Bayesian estimators. Our results show that the NN's performance in both the parameter estimation and the noise estimation problems was better than the ML estimator, and was comparable to the optimal Bayesian estimator.

Estimation Problem

In the process of acquiring FDA approval for a new drug, it is necessary for the developer to suggest a pharmacokinetic (PK) model that relates dosage to the drug's response in the patient (*e.g.*, a compartmental model that describes the dynamics of drug uptake and removal in various organs). One objective of such a model would be to estimate how much drug a physician should administer (and at what intervals) in order to achieve a given drug concentration in the blood stream of the patient. The distribution for the parameters of these PK models are calculated from data in a large number of patients. The developed population model is then used to regulate initial dosage to new patients after the drug has been approved by the FDA.

For many drugs that can produce life threatening toxicities, a patient's dosage regimen is often adjusted based on feedback from blood samples. This feedback is used for estimating the PK model parameters of the patient,

* *Supported by NIH grant P41-RR01861. R. S. is supported by a Doctoral Fellowship in Computer Science from I.B.M.*

given the prior distribution of parameters from the FDA approval studies. In the clinical setting, the particular patient's model parameters must be estimated from a limited number of blood samples: sampling procedures are generally expensive, and are influenced by errors associated with the drug assay techniques. In summary, the estimation problem is to predict the PK model parameters of a given patient based on a sparse set of noisy sampled data, in order to keep the plasma concentrations of a drug at a desired level.

Because of the sparse nature of this estimation problem, nonlinear state estimation algorithms are thought to be unsuitable [2]. However, Bayesian methods have been proposed [6] and successfully applied [7]. In our approach, we view the NN as an estimator which maps the noisy sampled data from a particular patient into that patient's PK model parameters. We also require the network to estimate the noise present in the sampled data. In what follows, a NN estimator is developed and its performance evaluated for a specific drug (theophylline, a bronchodilator) and clinical application (continuous intravenous therapy).

Pharmacokinetic Model

The PK model considered here has two parameters, a distribution volume V (liters), and an elimination clearance CL (liters/hr). Input to the model is the rate of drug dosage $r(t)$ (mg/hr). Model outputs are drug concentration $x(t)$ (μg) in the patient's body, and concentrations in the patient's blood $y(t)$ (μg/ml).

$$\frac{dx(t)}{dt} = -\frac{CL}{V}x(t) + r(t) \qquad y(t) = \frac{x(t)}{V}$$

Above model is used to describe the plasma concentrations following an initial loading (r_1) and maintenance (r_2) infusions of the drug. It is assumed that in the drug-approval studies, the distribution of the two model parameters V and CL are found to be log-normal with the mean, $\mu = \{ \mu_i \}$, i=1,2, $\mu_1 = 35$, $\mu_2 = 2.7$, respectively, and covariance, $\Omega = \{ \omega_{ij} \}$, i=j=1,2, $\omega_{11} = 125$, $\omega_{22} = 1.72$, $\omega_{21} = \omega_{12} = 7.34$.

For a particular patient, we would like to produce a concentration of 10 μg/ml at 30 minutes after the initial infusion of this drug, and keep this concentration constant for 24 hours. In order to calculate the dosage regimen for this patient, it is initially assumed that the patient's PK model parameters reside around the mean values of the population distribution. Based on this assumption, infusion rates of r_1 = 700 mg/hr over the interval of $0 < t <$ 30 minutes, and r_2 = 27 mg/hr for $0.5 < t < 24$ hours are selected.

To estimate the patient's actual PK model parameters, two plasma samples are taken and the drug's concentrations are measured by an assay technique. This technique adds an error term to the true concentrations of the drug: Measurements, $z(t_1)$ and $z(t_2)$, are related to the true concentrations of the drug, $y(t,\alpha)$ (μg/ml), as follows: $z(t) = y(t,\alpha) + e(t)$, where $\alpha = [V\ CL]^T$, and $e(t)$ is the error. The measurement error is assumed to be Gaussian with zero mean and a standard deviation of $\sigma_t(\alpha) = 0.15 * y(t,\alpha)$. It is assumed that the samples are taken at t_1 = 1.5 hrs and t_2 = 10.0 hrs. Given samples $z(t_1)$ and $z(t_2)$, the problem is to (1) estimate this individual's model parameters, and (2) estimate the noise present in each sample.

Estimation Procedure

Two traditional statistical approaches are used to solve such estimation problems for sparse data systems: maximum likelihood (ML) estimation and a Bayesian procedure which calculates the maximum *a posteriori* probability (MAP). The first is based on Gauss's suggestion that " ... the most probable value of the unknown quantities will be that in which the sum of squares of the differences between the actually observed and the computed values ... is a minimum." For the notation defined above, the ML estimate, α_{ML}, is defined as follows:

$$\alpha_{ML} = arg\ \{ \min_{\alpha} O_{ML} \} \qquad O_{ML} = \sum_{i=1}^{2} \frac{(z(t_i) - y(t_i,\alpha))^2}{\sigma_{t_i}^2(\alpha)} + ln\ \sigma_{t_i}^2(\alpha)$$

The second estimator, MAP, is a Bayesian point estimator which can be calculated in a computationally straightforward manner, given that certain distributional assumptions are met [2]. For the system model and error variance described in the previous section, the MAP estimator is given by α_{MAP} below, where $v = \{v_i\}$, $i = 1,2$, and $\Phi = \{\phi_{ij}\}$, $i=j=1,2$, and $v_i = ln\ \mu_i - \phi_{ii}/2$, $i = 1,2$, where $\phi_{ij} = ln\ (\omega_{ij}/\mu_{ij}\mu_{ij} + 1)$, $i,j = 1,2$. Details of above procedures are given in [2].

$$\alpha_{MAP} = arg \left\{ \min_{\alpha} O_{MAP} \right\} \qquad O_{MAP} = \sum_{i=1}^{2} \left[\frac{(z(t_i) - y(t_i, \alpha))^2}{\sigma_{t_i}^2(\alpha)} + ln\ \sigma_{t_i}^2(\alpha) \right] + [ln\ \alpha - v]^T \Phi^{-1} [ln\ \alpha - v] + 2\sum_{i=1}^{2} ln\ \alpha_i$$

A NN estimator was also designed and trained for this problem. The network was of the standard feedforward variety, consisting of three layers: two input units, seven hidden units, and four output units. The number of hidden units was arrived at empirically based on the change in the performance of the network as a function of the number of hidden units. The inputs to the network were the data samples $z(t_1)$ and $z(t_2)$. The four outputs of the network consisted of estimates of the PK model parameters $\hat{V}$ and $\hat{CL}$ for an individual patient, and estimates of the actual value of the data samples (noise-free) $\hat{y}(t_1)$ and $\hat{y}(t_2)$.

The training procedure for the NN estimator was as follows: From the log-normal population of V and CL, a *training set* of 150 pairs of model parameters were selected. For each parameter pair (representing one patient in the pre-approval studies), and for the given r_1 and r_2 and assumed PK and noise model, data samples $z(t_1)$ and $z(t_2)$ were given as input to the network. The errors in estimating model parameters and noise components of the sampled data were fed back using the back-propagation algorithm [5]. Both parameters of this algorithm were adjusted empirically based on rate of decline and asymptotic value of the sum of the absolute error. Increasing the size of the training set provided no appreciable improvements in estimation performance of the network.

Performance of the Estimators

To evaluate the performance of the trained NN estimator with that of the ML and the Bayesian (MAP) estimators, a *test set* consisting of 1000 measurement & parameter vectors was generated. The training and test sets had similar means and variances. To quantify the performance of these estimators, we calculated a prediction error term (pe_i) for each estimation point in the test set, *e.g.* for V, $pe_i = (\hat{V}_i - V_i)*100/V_i$, $i=1, \ldots, 1000$, where $\hat{V}_i$ and V_i are the estimated and true values for the i-th entry in the test set. Performance of the estimators over the entire test set was expressed as the mean and root mean square of the error (*Mpe* and *RMSpe*, respectively) for each predicted variable. *Mpe* reflects the bias of the estimator, and it is given below:

Table 1. Mean prediction error as a percentage of the actual value of the parameter, for the three estimators.

	$y(t_1)$	$y(t_2)$	CL	V
NN	0.6	2.3	3.8	4.7
ML	-1.1	-3.0	3.4	2.5
Bayesian	0.8	1.5	6.1	1.0

Results from Table 1 indicate that: (1) Biases of the three estimators were small for all variables, and (2) Biases were not significantly different among the three estimators (ANOVA $p > 0.05$). Since the mean error term is influenced by the cancelling effect of under- and over-estimation, we also examined the *RMSpe*. The results are plotted in Fig. 1. Based on this measure, the NN consistently performed better than the ML estimator, and approached that of the MAP estimator, which is an optimal classifier in terms of probability of correct classification.

We also compared the performance of the noise models developed by the NN and the Bayesian estimator. This is reflected in the *Mpe* of $y(t_1)$ and $y(t_2)$ in table 1, and *RMSpe* of $y(t_1)$ and $y(t_2)$ in Fig. 1. There appears to be little or no differences in the performance of these two noise models. We have plotted in Fig. 2 the 95% confidence interval for the three estimators: the error associated with the predicted value of a parameter will be bounded by this interval with a probability of 95%. This measure of performance is useful because it indicates how bad the worst predictions of these estimators were. For example, for CL, ML's worst estimates were about ±100% off the actual value. This compares with the NN's worst prediction errors of about ±60%, which is very

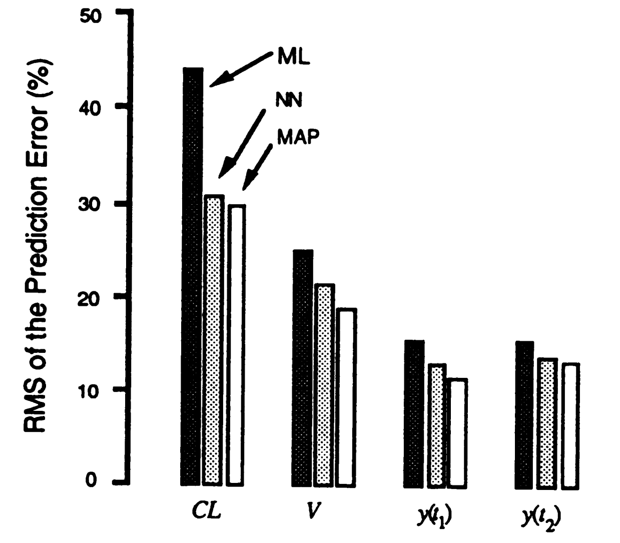


Figure 1. RMS of the error associated with predicting the variables, expressed as % of actual value, for the estimators.

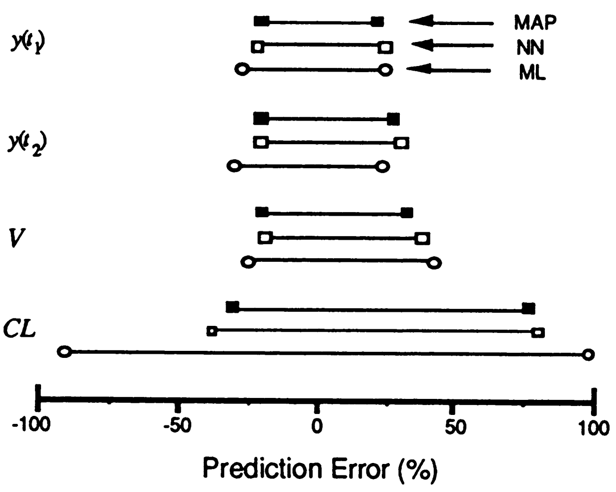


Figure 2. 95% confidence interval (see text) of the four variables by the three estimators.

similar to the Bayesian estimator's errors (Fig. 2). The entire test set and the predictions of the NN and Bayesian methods are plotted in Fig. 3 for the parameter V.

Arguments can be made that the process of supervised learning in a NN is a form of describing a Bayesian estimator. We have provided some evidence here that for the particular application of estimating parameters in a sparse and noisy data environment, the NN's performance approaches that of the optimal Bayesian MAP estimator. We now point out a potential advantage of the NN over Bayesian methods: More realistic kinetic models would require multiple compartments and a larger number of parameters. With only a few data samples available from such a system, significant simplifying assumptions are required in order for the Bayesian estimator to be constructed. It may be possible however to train a NN on simulations from a more complete model, while still producing estimates of a few unknown parameters that are of interest to the developer.

[1] Cover, T.M., & P.E. Hart (1967). Nearest neighbor pattern classification, IEEE Trans on Inform. Theory, IT-13, 21-27.

[2] D'Argenio, D.Z., & D.C. Maneval (1988). Estimation approaches for modeling sparse data systems, *IFAC Modelling and Control in Biomedical Systems*, Italy, 61-67.

[3] Gorman, R.P., & T.J. Sejnowski (1988). Analysis of hidden units in a layered network trained to classify sonar targets, *Neural Networks*, 1, 75-89.

[4] Mallet, A. (1986). A maximum likelihood estimation method for random coefficient regression models, *Biometrika*, 73:645-56.

[5] Rumelhart, R.E., G.E. Hinton & R.J. Williams (1986). Learning internal representations by error propagation, *Parallel Distributed Processing*, Eds: D.E. Rumelhart & J.L. McClelland, 318-362, MIT Press.

[6] Sheiner, L. B., H. Halkin *et al.* (1975). Improved computer-assisted digoxin therapy: a method using feedback of measured serum digoxin concentrations, *Ann Intern Med*, 82:619-627.

[7] Vozeh, S., G. Kewitz, & F. Follath (1980). Rapid prediction of steady state serum theophylline concentration in patients treated with intravenous aminophylline, *Eur J Clin Pharmacol*, 18:437-447.

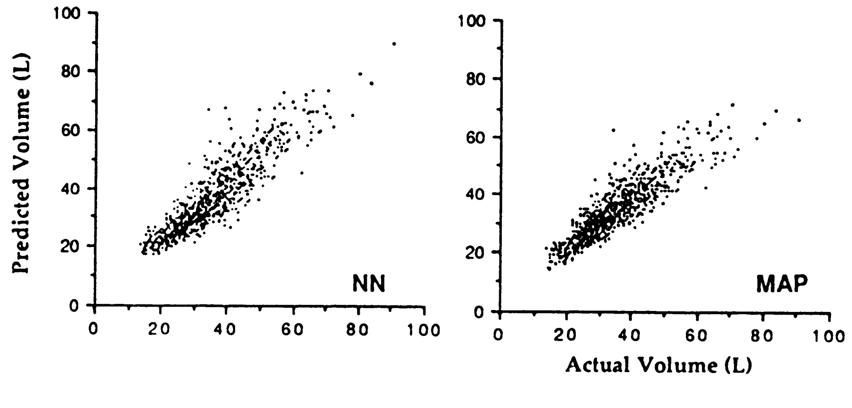


Figure 3. Predicted vs. actual values of the parameter V (Volume) for the neural network (NN) and the Bayesian (MAP) estimators.

Neural Networks Models for Linear Programming

Jean-Christophe Culioli and Vladimir Protopopescu
Engineering Physics and Mathematics Division
Charles L. Britton, Jr., and Milton N. Ericson
Instrumentation and Control Division
Oak Ridge National Laboratory, Oak Ridge, TN 37831-6364

Abstract: The purpose of this paper is to present a neural network that solves the general Linear Programming (LP) problem. In the first part, we recall Hopfield and Tank's circuit for LP and show that although it converges to stable states, it does not, in general, yield admissible solutions. This is due to the penalization treatment of the constraints. In the second part, we propose an approach based on Lagragrange multipliers that converges to primal and dual admissible solutions. We also show that the duality gap (measuring the optimality) can be rendered, in principle, as small as needed.

1. Introduction. The contribution of neural networks to Optimization Theory has been mainly dedicated to NP-complete problems so far, and in particular to the Travelling Salesman Problem [1] (see also [2] for an excellent account). Here, we consider "simpler" problems, like Linear Programming (LP) and its variants, for which low-order polynomial algorithms are already available [3]. Although some of these simpler problems are combinatorial by formulation (like the Assignment Problem [4]), their structure is inherently continuous and seems well adapted to a neural network solution.
The only attempts we are aware of, concerning continuous optimization problems are Hopfield and Tank's circuit for LP [5], its application to Analog Decoding [6], the work of Jeffrey and Rosner [7] for the solution of variational problems, and our recent application of sigmoidic functions to general optimization problems[8,9]. In the next Section, we give some properties of linear programs. In Section 3, we recall some background on Hopfield and Tank's LP network and discuss its connection with standard optimization methods. We note that it converges to stable states which are not, in general, admissible solutions of the problem. In Section 4, we propose a new network that, in handling the constraints, relies on duality instead of direct penalization. This network always converges to primal and dual admissible solutions, and the associated duality gap can be rendered as small as desired.

2. Linear Programming. We intend to solve the LP problem

$$\min < c, x > \quad subject \;\; to \;\; Ax \geq b, \tag{1}$$

where c and x are vectors in R^n, b is a vector in R^m, and A is an $m \times n$ matrix, with $m \leq n$. The brackets $< .,. >$ denote the scalar product in R^n. We assume that problem (1) has a bounded solution x^* and, for the purpose of the forthcoming derivations, that the rank of A is m. We define the operation $[\,.\,]^- : [y]^- = y \;\; if \;\; y < 0, \quad [y]^- = 0 \;\; if \;\; y \geq 0$, and apply it componentwise, if y is a vector. In the following, a vector x such that $Ax \geq b$ will be called admissible for (1). It is common, when dealing with LP problems to introduce their dual problems. The dual problem associated with (1) is

$$\max < b, p > \quad subject \;\; to \;\; A^T p = c, \quad and \quad p \geq 0, \tag{2}$$

where A^T denotes the transpose of matrix A. Note that we have also denoted the scalar product in R^m by $< .,. >$. The fundamental result of duality is (for a proof, see [10]):
Proposition 1. *If $\hat{x}$ is admissible for (1) and $\hat{p}$ is admissible for (2) (that is $A^T\hat{p} = c, \;\; \hat{p} \geq 0$), then the duality gap $\delta := < c, \hat{x} > - < b, \hat{p} >$ is positive. If $\delta = 0$, then $\hat{x}$ is a solution of (1), and $\hat{p}$ is a solution of (2).*

3. Hopfield and Tank's Neural Network for Linear Programming. The neural network proposed in [5] for the solution of (1), contains n neurons with internal states u_i, output values $x_i = g_\lambda(u_i)$, and time response τ. We denote them as neurons (u_i, x_i), $(i = 1, \ldots, n)$. The function g is assumed linear and increasing ($g_\lambda(u) = \lambda u, \quad \lambda > 0$). We also have m neurons (with no time response) with internal state $y_j = \sum_i A_{ji}x_i - b_j$ and output values $\psi_j = (y_j^-)^2$. There are no connections within the set of $\{(u_i, x_i)\}_i$ neurons nor within the set of $\{(y_j, \psi_j)\}_j$ neurons. On the other hand, each (u_i, x_i) neuron is connected, with connection strength A_{ji}, to the neuron (y_j, ψ_j) . The dynamics of the $\{(u_i, x_i)\}_i$ neurons is given by the equation (that we directly write in vector form for the whole set)

$$(3) \qquad \frac{du}{dt} = -c - \frac{u}{\tau} - A^T[Ax - b]^-.$$

An energy functional is associated with (3)

$$(4) \qquad E(x) =< c, x > +\frac{1}{2}||[Ax - b]^-||^2 + \frac{1}{2\lambda\tau}||x||^2.$$

We now prove the convergence of the network and address the optimality of the procedure.
Proposition 2. The functional $E(x)$ is a Lyapunov functional for (3).
Proof: $E(x)$ is bounded below since it contains a quadratic term in $||x||$. Also, we have

$$\frac{dE(x)}{dt} =< c, \frac{dx}{dt} > + < A^T[Ax - b]^-, \frac{dx}{dt} > +\frac{1}{\lambda\tau} < x, \frac{dx}{dt} >$$

$$=< c + A^T[Ax - b]^- + \frac{1}{\lambda\tau}x, \frac{dx}{dt} >= - < \frac{du}{dt}, \frac{dx}{dt} >= -\lambda||\frac{du}{dt}||^2 \leq 0.$$

Thus $E(x)$ is decreasing along the trajectories of (3) and $\frac{dE(x)}{dt} = 0$ implies that $\frac{du}{dt} = 0$. ∎

The network driven by (3) is designed to solve the problem $\min_x E(x)$, where one attempts to satisfy the constraint by penalization. We will show that, in general, this does not imply that the network solves the original problem (1). Let $(\bar{u}, \bar{x}, \bar{y}, \bar{\psi})$ be a stable state. Then $\bar{x}$ is solution of the fixed point equation

$$(5) \qquad c + \frac{1}{\lambda\tau}\bar{x} + A^T[A\bar{x} - b]^- = 0$$

We can state
Proposition 3. *Two necessary conditions (on $\lambda\tau$) for $\bar{x}$ to be admissible are*

$$(6) \qquad \lambda\tau \leq -\frac{< c, x^* >}{||c||^2}, \quad and \quad \lambda\tau Ac \leq -b.$$

Proof: Indeed, if $\bar{x}$ is admissible, then we get $\bar{x} = -\lambda\tau c$ from (5). Since the solution x^* is such that $< c, x^* > \ \leq \ < c, x >$ for every admissible x, we must have $< c, x^* > \ \leq -\lambda\tau||c||^2$. By applying the matrix A to (5) and by using the admissibility of $\bar{x}$, we get the other condition. ∎
Corollary. *In general, the solution of (5) is not admissible for (1).*
Indeed, since $\lambda\tau$ is strictly positive, whenever $< c, x^* >$ is positive or equal to zero, one cannot obtain an admissible $\bar{x}$. Even if $< c, x^* >$ is negative, the second condition of (6) implies $\lambda\tau < Ac, b > \ \leq \ -||b||^2$, which is not satisfied whenever $< Ac, b > \ > 0$. Counterexamples can be easily constructed.

4. A Primal-Dual Neural Network for Linear Programming. We now propose a neural network named Primal-Dual because it provides admissible primal ($\bar{x}$) and dual ($\bar{p}$) vectors . Moreover, we will show that the duality gap can be made as small as desired. We consider n neurons of the type (u, x) and m neurons of the type (v, p). We assume that

$$(7) \qquad x = R\ G_\lambda(u), \quad p = g_\mu(v), \quad with \quad \lambda > 0, \quad R > 0, \quad G'_\lambda(u) > 0, \quad g_\mu(v) \geq 0, \quad g'_\mu(v) > 0,$$

where R is a large bound on $||x||$ which has not to be known precisely. Although the following derivations do not use their explicit form, the functions G_λ and g_μ can be chosen as $G_\lambda(u) = \tanh(\lambda u)$ (with λ not too large in order to prevent instabilities) and $g_\mu(v) = \frac{e^{\mu v}}{\mu}$. The latter choice of g_μ was proposed in [6]. It may have, however, a natural tendancy to produce numerical difficulties.
The evolution equations of these neurons are assumed to be

$$(8) \qquad \frac{du}{dt} = -c + A^T p, \quad \frac{dv}{dt} = -\frac{p}{\mu\tau} - b + Ax.$$

The main differences between (8) and (3) are the following: (i) the neurons (u, x) have no time constant, and the relation between u and x is not totally linear, (ii) the constraints are "softly" penalized by the output values of the neurons (v, p), (iii) unlike the neurons (ψ, y) of the Hopfield and Tank model, the neurons (v, p) have an explicit evolution as independent state variables, and have a time constant. These differences do not affect significantly the feasibility of an analog implementation [11].
We now address the convergence of the network and the optimality of its fixed points.
We choose the Lyapunov functional

$$(9) \qquad E(x,p) =< c, x > - < p, Ax - b > + \frac{||p||^2}{2\mu\tau}.$$

Convergence: $E(x,p)$ is bounded below because $||x||$ is bounded by $\sqrt{n}R$ and E contains a quadratic term in $||p||$. It decreases along the trajectories of (8). Indeed,

$$\frac{dE}{dt} =< c - A^T p, \frac{dx}{dt} > - < \frac{dp}{dt}, Ax - b - \frac{p}{\mu\tau} >= -\lambda \sum_i G'_\lambda(u_i)(\frac{du_i}{dt})^2 - \sum_j g'_\mu(v_j)(\frac{dv_j}{dt})^2 \leq 0.$$

Thus, $\frac{dE(x)}{dt} = 0$ implies $\frac{du}{dt} = 0$ and $\frac{dv}{dt} = 0$. ∎
Consequently, the network defined by (7) and (8) converges to a stable state $(\bar{u}, \bar{x}, \bar{v}, \bar{p})$.
The admissibility and optimality of the fixed points is addressed in
Proposition 4. *The fixed point $\bar{x}$ is admissible for (1) and the fixed point $\bar{p}$ is admissible for (2). The associated duality gap $\delta =< c, \bar{x} > - < b, \bar{p} >$ is equal to $\frac{||\bar{p}||^2}{\mu\tau}$. Moreover, if the rank of A is equal to m, $\bar{p}$ is bounded and the duality gap has a magnitude of order $O(\frac{1}{\mu\tau})$.*
Proof: From the fixed point equations, one gets $A\bar{x} - b = \frac{\bar{p}}{\mu\tau} \geq 0$. Thus $\bar{x}$ is admissible for (1). Also, we have $A^T\bar{p} = c$ and since $\bar{p} = g_\mu(\bar{v}) \geq 0$, $\bar{p}$ is admissible for (2). If we multiply $A^T\bar{p} = c$ by $\bar{x}$, we get $< c, \bar{x} > - < b, \bar{p} >= \frac{1}{\mu\tau}||\bar{p}||^2$.
If the rank of A is equal to m, then (AA^T) is an $m \times m$ positive invertible matrix. Its eigenvalues (ordered by increasing size) $a_1, ..., a_m$ are stricly positive, and from $A^T\bar{p} = c$, one gets $||\bar{p}|| \leq \frac{1}{a_1}||c|| \leq M < +\infty$.
Thus, $< c, \bar{x} > - < b, \bar{p} > \leq \frac{1}{\mu\tau}M^2$. ∎

Application. In order to get an approximation of the optimal cost, $< c, x^* >$, with an absolute precision ϵ , one can choose $\mu_\epsilon = \frac{M^2}{\epsilon\tau}$. One might expect, however, that due to the form of g_μ, some bifurcation behavior can appear (See [9] for typical examples). It will thus be advisable to start the network with a moderate μ and increase it progressively to the value μ_ϵ. This procedure would bear some analogy with an "annealing" technique [1,9].

5. Conclusion. In this paper, we have studied two neural networks models for Linear Programming, the Hopfield and Tank network, and the Primal-Dual network. We have shown that the Primal-Dual network converges to admissible solutions and can be used to get a very good approximation of the optimal cost. Throughout the paper, we also addressed some implementation issues. Extensive numerical simulations and analog circuit implementation will be our next focus [11].

Acknowledgements

This research was partially sponsored by the U.S. Air Force Wright Aeronautical Laboratory under DOE Interagency Agreement, DOE-1570-1579-A1, by the Office of Basic Energy Sciences and the Exploratory Studies Program of Oak Ridge National Laboratory, U. S. Department of Energy, under contract # DE-AC05-84OR21400 with Martin Marietta Energy Systems, by DARPA under contract # 1868-A037-A1, and by an ORNL Postgraduate Research Appointment administered by Oak Ridge Associated Universities.

References

[1] J. J. Hopfield and D. W. Tank, "“Neural” Computation of Decisions Optimization Problems", Biological Cybern., vol. 52, pp. 141-152.

[2] A. B. Kahng,"Travelling Salesman Heuristics and Embedding Dimension in the Hopfield Model", Proceedings of the International Joint Conference on Neural Networks, Washington D. C., June 1989.

[3] N. K. Karmarkar, "A New Polynomial-Time Algorithm for Linear Programming", Combinatorica 4, Dec. 1984.

[4] J. C. Culioli, V. Protopopescu, C. Britton, and N. Ericson, "A Neural Network for Explicitly Bounded Linear Programming", this Conference Proceedings.

[5] J. J. Hopfield , and D. W. Tank, "Simple “Neural” Optimization Networks : an A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit", IEEE Trans. on Circuits and Systems, Vol. CAS-33, N. 5, May 1986.

[6] J. C. Platt and J. J. Hopfield, "Analog Decoding Using Neural Networks", Proc. of the AIP Conference, Snowbird, UT 1986.

[7] W. Jeffrey and R. Rosner, "Neural Network Processing as a Tool for Function Optimization", Proc. of the AIP Conference, Snowbird, UT 1986.

[8] J. C. Culioli and V. Protopopescu, "An Algorithm for Linear Programming That is Easy to Implement", Applied Mathematics Letters, Vol. 2, N. 2, pp.125-129, 1989.

[9] J. C. Culioli and V. Protopopescu, "Bifurcating Optimization Algorithms and their Possible Application", ORNL/TM-10976, Nov. 1988.

[10] G. B. Dantzig, Linear Programming and Extensions, Princeton University Press, 1964.

[11] J. C. Culioli, V. Protopopescu, C. Britton, and N. Ericson, ORNL Technical Memo, in preparation.

On The Amari-Takeuchi Theory of Category Formation

Morris W. Hirsch
Department of Mathematics
University of California Berkeley, CA 94720
September 19, 1989

Amari and Takeuchi (1978) presented an interesting theory of category formation for a single perceptron-like unit under the influence of changing environmental input and Hebb-like unsupervised learning. The present paper is a more rigorous foundation for a version of their theory, together with further analysis of the categories that can develop.

Our unit has *n* input lines and a weight vector $W \in \mathbf{R}^n$. There are *m* input patterns $X_\alpha \in \mathbf{R}^n$, $\alpha = 1, \dots, m$. The output is $g(W \cdot X_\alpha - \theta_\alpha)$ where g is a sigmoidal function and θ_α is a threshold (or bias) which can depend on the input. (This might arise from an inhibitory signal passed to the unit from another unit aroused by X_α, whose weights are constant in time.) In contrast to Amari and Takeuchi, ***we assume g is continuously differentiable,*** with positive derivative. For definiteness we assume $g(x) = \sigma(\gamma x)$ where $\sigma(x)$ is a standard sigmoid such as the logistic function, and $\gamma > 0$ is a system parameter called the ***gain***. The limiting values of g(x) as $x \rightarrow +\infty$ or $-\infty$ are respectively 1 and 0. (For Amari and Takeuchi the gain is infinite, that is, g is the discontinuous Heaviside function. This makes Equation (1) below easy to analyze; but (2) is then very difficult; moreover it is unclear whether (2) is a good approximation to (1).)

The weights evolve as follows. At presentation of the k'th input X_α let the weight vector be $W(k-1)$. After the output is computed, the weight changes according to the vector differential equation

$$\tfrac{1}{\lambda} dW/dt = -W + g(W \cdot X_\alpha - \theta_\alpha) X_\alpha \,, \quad k-1 \leq t \leq k. \qquad (1)$$

where the system parameter $\lambda > 0$ is the ***learning rate***. The new weight is W(k). We assume successive inputs are chosen according to an infinite sequence $\alpha(k)$ aking values in $\{1, \dots, m\}$, $k = 1, 2, \cdots$. Piecing together solutions of (1) for successive *k* gives a continuous $W(t)$, $0 \leq t < \infty$.

The basic assumption on the input sequence $\{X_{\alpha(k)}\}$ is that in the long run the proportion of k such that $\alpha(k)=j$ approaches a limit p_j, $0 \leq p_j \leq 1$, for each $j=1, \ldots, m$. *The $\alpha(k)$ may be chosen either deterministically or stochastically.* In the latter case, for many stochastic processes $\{\alpha(k)\}$ some form of the Law of Large Numbers, or an Ergodic Theorem, implies the existence almost surely of the limits p_j. Amari and Takeuchi require a deterministic sequence for their application to a model of development of feature detectors in the retinal cortex.

The key to analyzing equation (1) is to use the *Averaging Theorem*: Given $T>0$ and $\epsilon>0$ there exists $\lambda_0>0$ such that if $0<\lambda<\lambda_0$ then the solution to (1) is within ϵ of the solution with the same initial value to

$$\tfrac{1}{\lambda}d\hat{W}/dt = -\hat{W} + \sum_{\alpha=1}^{m} p_\alpha g(\hat{W}\cdot X_\alpha - \theta_\alpha)X_\alpha\ , \quad k-1 \leq t \leq k. \tag{2}$$

for $0 \leq t \leq T/\lambda$. (The proof requires that g be continuously differentiable.) From now on we study (2), identifying $\hat{W}$ with W. Note that if $p_\beta=0$ then X_β does not appear in (2). Thus inputs that are sufficiently infrequent have no effect on the long-term development of the weight vector. We might as well assume all $p_\alpha>0$.

Equation (2) is of gradient form: the right hand side is the gradient of the "energy" function

$$E(W) = -\tfrac{1}{2}\|W\|^2 + \sum_{\alpha=1}^{m} p_\alpha G(W\cdot X_\alpha - \theta_\alpha)$$

where $G(x)$ is any antiderivative of $g(x)$. It follows that every solution converges to an equilibrium, and under plausible "generic" assumptions almost every solution tends to some stable equilibrium, i.e. a local maximum point for E. It is easy to see that $W(t)\cdot Y \to 0$ exponentially for any solution $W(t)$ and any vector Y orthogonal to all the inputs.

Suppose a solution $W(t)$ of (2) converges to a stable equilibrium W_*. It is important to realize that the weight is not really frozen at W_*; in reality it changes according to system (1) as the different inputs are presented. But the actual solution $W(t)$ to (1) is close to W_* for long periods of time; and it can be proved that for high enough gain it is as close as desired ator all sufficiently large times t.

Insight into (2) is obtained by introducing the *recall variables* $u_\alpha = W \cdot X_\alpha$. Define a symmetric $m \times m$ matrix T by $T_{\beta\alpha} = X_\alpha \cdot X_\beta$, and sigmoids $g_\alpha(x) = p_\alpha g(x - \theta_\alpha)$. Dotting both sides of (2) with X_β gives the system

$$\tfrac{1}{\lambda} du_\beta/dt = -u_\beta + \sum_{\alpha=1}^{m} T_{\beta\alpha} g_\alpha(u_\alpha), \quad \beta = 1, \ldots, m. \tag{3}$$

This describes the *activation* dynamics (not the weight dynamics!) of the well known class of recurrent additive nets with symmetric weight matrices. The "net" corresponding to (3), with activations u_β and weights $T_{\alpha\beta}$, is only conceptual; but that does not prevent us from applying the crucial *Theorem of Hopfield* (1984): If the gain γ is sufficiently large then for any stable equilibrium u of (3), $g_\alpha(u_\alpha)$ is close to one of its limiting values. In terms of (2) this says: For sufficiently high gain, if W_* is a stable equilibrium then $g(W_* \cdot X_\alpha)$ is close to 0 or 1, for every α. This can also be proved directly from (2). Notice that when $m < n$, (3) has fewer equations than (2).

From now on we assume very high (but not infinite!) gain and very low learning rate λ. For each stable equilbrium W_* of (2), Amari and Takeuchi define the *category* $S(W_*) = S$ to be the set, *possibly empty*, of those X_α such that $g(W_* \cdot X_\alpha) \approx 1$. Thus $X_\alpha \in S$ means that when the long-term memory state is W_* then the unit fires whenever pattern X_α is input. We may say that in this case the unit "recognizes" X_α.

A category represents a possible mode of long-term development of the unit; Amari and Takeuchi interpret a category as a feature detector. Different initial values of W and different thresholds θ_α may lead to different categories. It is likely, but not known, that different stable equilibria can determine the same category. In any case, in functional terms it is the categories, not the stable equilibria, that characterize the developed network.

Amari and Takeuchi analyze the structure of categories in terms of various parameters. Analogs of some of their results can be obtained in the present setup. Here however we present a new result expressing a set-theoretic property of the set of all categories, under the special assumption, biologically plausible, that *the components of the vectors* X_α *are nonnegative*.

This makes (2) (also (1) and (3)) into a *cooperative system*: Denoting the right hand side of (2) by F(W), we have $\partial F_i/\partial W_j \geq 0$ for $i \neq j$. Using the order-preserving properties of the solution flow of a cooperative system (Hirsch 1982, 1989), the following result can be proved:

The union of two categories is contained in a category; the intersection of two categories contains a category.

As a consequence, if there is a long-term memory state W. recognizing X_α, and another recognizing X_β, then there is one recognizing both X_α and X_β. If every input pattern is recognized by some W, then the set comprising all inputs is a category, and so is the null set. In any case there is a unique maximal category and a unique minimal one.

Units of the type considered here can be cascaded to form a feed-forward net in which each unit develops its own category of those signal patterns, from earlier layers of the net, which it recognizes.

REFERENCES

Amari, S. & Takeuchi, A. 1978: Mathematical theory of category detecting nerve cells. Biol. Cybern. 29, 127-136

Hirsch, M. W. 1982: Systems of differential equations that are competitive or cooperative. I: Limit sets. SIAM J. Math. Anal. 13, pp. 167-179.

Hirsch, M. W. 1989: Convergent Activation Dynamics, to appear in Neural Networks, September issue.

Hopfield, J. J. 1984: Neurons with graded response have collective computational properties like those of two-state neurons. Proc. Nat. Acad. Sci. USA 81, pp. 3088-3092.

STATE EVALUATION FUNCTIONS FOR NEURAL NETWORKS AND POSSIBLE LYAPUNOV FUNCTIONS

Youichi Kobuchi
Department of Electronics and Informatics
Faculty of Science and Technology
Ryukoku University, Seta
Oe-cho Seta, Otsu-shi, 520-21 JAPAN

Hopfield[3] was the first to show that there is a Lyapunov function for a given *symmetric* neural network which operates *asynchronously* where each component model neuron is basically that of McCulloch and Pitts[5] or its continuous counterpart. It corresponds to the "energy" of the network which decreases as time elapses.(See, for example, Feldman[1].) We examine the alleged proposition that the neural networks should have symmetric weights to have Lyapunov functions. For that purpose, we start by defining a state evaluation function which is a mapping from the set of state configurations into the set of real numbers. Then the difference function of such state evaluation function is introduced and conveniently characterized. An application of the properties of difference functions reveals that there is a slightly extended class of *asymmetric* neural networks which have Lyapunov functions.

1. State Evaluation Functions and Their Difference Functions

Consider an arbitrary network composed of n elements each of which takes one of the two states $\{0,1\}$. A generic configuration is denoted as $s = (s_1,s_2,\ldots,s_n)$ where s_i is in $\{0,1\}$ for $i \in N_n$ and $N_n \equiv \{1,2,\ldots,n\}$. We first assume that there is a function E(s), called a *state evaluation function*, from $\{0,1\}^n$ to $\boldsymbol{R}$ where $\boldsymbol{R}$ is the set of real numbers. Note that the framework here is quite general that we are not concerned with any particular network dynamics at present. Let J be a subset of N_n . Then s_J denotes a configuration such that $s_J = (s'_1,s'_2,\ldots,s'_n)$ where $s'_i = \overline{s_i} \equiv 1 - s_i$ if i is in J and $s'_i = s_i$ otherwise. For any J , K $\subseteq N_n$ let $J \oplus K = \{ i \in N_n \mid (i \in J \text{ and } i \notin K)$ or $(i \in K \text{ and } i \notin J)\}$. Now we define a *difference function* as follows: For $J \subseteq N_n$, let $\Delta E_J(s) \equiv E(s) - E(s_J)$. In passing, note that $\Delta E_J(s)$ is a kind of state evaluation function with parameter J. A basic relation which must hold among these difference functions is that $\Delta E_J(s) = \Delta E_K(s) + \Delta E_{K\oplus J}(s_K)$ for any J , K $\subseteq N_n$. In fact, this relation is shown to hold if it holds for any $J \subseteq N_n$ and any singleton set $K \in N_n$. The relation also turns out to be a sufficient condition for the existence of a function E(s)such that $\Delta E_J(s) = E(s) - E(s_J)$. That is, consider a family of state evaluation functions $\Delta F_J(s)$ from $\{0,1\}^n$ into $\boldsymbol{R}$ with parameter $J \subseteq N_n$ such that the above relation for $\Delta E_J(s)$ holds. Then define a function E(s) : $\{0,1\}^n \to \boldsymbol{R}$ by $E(s) \equiv \Delta F_{I(s)}(s) + C$ where C is a constant in $\boldsymbol{R}$ and I(s) denotes the set of integers i such that $s_i = 1$: $I(s) = \{ i \in N_n \mid s_i = 1$ where $s = (s_1,s_2,\ldots,s_n) \}$. Note that $\Delta F_{I(s)}(s)$ is a function of s since the parameter part I(s) is also determined by s. Now we can reduce the basic relation still further to obtain the following theorem.

<u>Theorem 1.</u> Let $\Delta E_k(s) : \{0,1\}^n \to \boldsymbol{R}$ be given for each $k \in N_n$. The following preservation conditions are necessary and sufficient for the existence of a function $E(s) : \{0,1\}^n \to \boldsymbol{R}$ such that

$\Delta E_J(s) = E(s) - E(s_J)$ where $\Delta E_J(s) \equiv \sum_{r=1}^{m} \Delta E_{j_r}(s_{J_r})$ for $J_r = \{j_1 j_2 \ldots j_{r-1}\}$, $J_1 = \phi$ and $J = J_{m+1}$:

For any j and k in N_n, $\Delta E_j(s) + \Delta E_k(s_j) = \Delta E_k(s) + \Delta E_j(s_k)$ (2nd order condition),

$\Delta E_k(s) + \Delta E_k(s_k) = 0$ (1st order condition), and

$$\Delta E_{\phi}(s) = 0 \quad \text{(0th order condition).}$$

The theorem says that the preservation conditions of order less than or equal to two are sufficient for the basic relation to hold. The state evaluation function E(s) can be a Lyapunov function for a network with certain dynamics such that $\Delta E_k(s)$ is nonnegative whenever s changes to s_k.

2. Neural Network Definitions and Searching for Lyapunov Functions

Using the above theorem, we examine what are the conditions for neural networks to have Lyapunov functions, if any. Assume that there are n McCulloch-Pitts neurons numbered 1 through n in a network, and also assume an asynchronous operation for the time being. Let $s_i(t)$ be the state of neuron i at time t, taking the value 1 or 0, and let $s = (s_1(t), s_2(t), \ldots s_n(t))$. If we write the weight connecting i to j as w_{ji}, and the threshold value for i as θ_i, the next state of neuron k can be defined as follows:

Evaluate $d_k(s) \equiv \sum_{j=1}^{n} w_{kj}s_j(t) - \theta_k$, then $s_k(t+1)$ becomes 1 if $d_k(s)$ is positive, and 0 if it is negative. We have assumed as in Goles [2] that for every i, $\sum_{j=1}^{n} w_{ij}s_j - \theta_i \neq 0$ for any $s_j \in \{0,1\}$ $i, j = 1,2,\ldots,n$. The state transition rule means that s_k changes to $\overline{s_k}$ if and only if $\tilde{s_k}\ d_k(s) < 0$ where $\tilde{s_k} \equiv s_k - \overline{s_k}$. In order to have a Lyapunov function E(s) for this network, it is necessary that E value should decrease when s changes to s_k, which implies $\tilde{s_k}\ d_k(s) < 0$. That is, the difference function can be appropriately expressed as $\Delta E_k(s) = f_k(-\tilde{s_k}\ d_k(s)) = -\tilde{s_k}\ f_k(d_k(s))$ where f_k is a sign preserving function. Then the preserving conditions in the previous theorem yield the following relations.

For any l and k in N_n,

$$\tilde{s_k}(f_k(d_k(s)) - f_k(d_k(s) - w_{kl}\tilde{s_l})) = \tilde{s_l}(f_l(d_l(s)) - f_l(d_l(s) - w_{lk}\tilde{s_k})) \quad \text{(2nd order relation), and}$$

$$f_k(d_k(s)) = f_k(d_k(s) - \tilde{s_k}\ w_{kk}) \quad \text{(1st order relation).}$$

Now we analyse these relations in general terms. From the 1st order relation, we have the following necessary condition because f_k is a sign preserving function as required.

$$d_k(s)(d_k(s) - \tilde{s_k}\ w_{kk}) > 0.$$

From this inequality, we can have a condition on w_{kk} under which a network under consideration behaves just like a network where $w_{kk} = 0$. Conversely, $w_{kk} = 0$ trivially ensures the 1st order relation. In short, the 1st order condition requires something like a point-wise stability. We call the property $w_{kk} = 0$ ($k = 1,2,\ldots,n$) as *zero-diagonal* condition.

In the above expression, the 2nd order relation is required for every possible s. It can be easily seen, however, that the relation holds if it is satisfied for the cases where $s_k = s_l = 0$. If we write s_k and s_l component of s as $s(s_k, s_l)$ then the relation can be rewritten as

$$f_k(d_k(s(0,0))) - f_k(d_k(s(0,0)) + w_{kl}) = f_l(d_l(s(0,0))) - f_l(d_l(s(0,0)) + w_{lk}) .$$

To be more concrete, we consider the case where each f_k is a linear function, i.e., $f_k(x) = c_k x$ for a positive constant c_k. Then we have $c_k w_{kl} = c_l w_{lk}$. The relations $c_k w_{kl} = c_l w_{lk}$ (k, l = $1,2,\ldots,n$) can be written in matrix form as $CW = W^tC$ where $W = [w_{ij}]$ and C is a diagonal matrix whose i-i element is c_i for $i = 1,2,\ldots,n$. Since $C = C^t$, this means that CW is symmetric. We call a matrix W *quasi-symmetric* (with respect to C) if CW is symmetric for some diagonal matrix C. A diagonal matrix is said to be *positive* if all of its diagonal elements are positive. We give several characterizations of quasi-symmetric matrices.

Lemma. Let W and C be a matrix and a non-singular diagonal matrix, respectively. Then the following conditions are equivalent.

(1) CW is symmetric.
(2) $W = C^{-1}U$ for some symmetric matrix U.
(3) W = VC for some symmetric matrix V.
(4) W = BSA for some symmetric matrix S and non-singular diagonal matrices B and A.

To sum up, we have shown that, under the linear assumption, neural networks must satisfy both zero-diagonal condition and quasi-symmetric condition with respect to a positive diagonal matrix in order to have some noncontradictory energy function.

3. Lyapunov Functions for Quasi-Symmetric Neural Networks

Here we calculate the desired Lyapunov function for a neural network satisfying zero-diagonal condition and quasi-symmetric condition with respect to a positive diagonal matrix C $= [c_i]$. For a given $s = (s_1,s_2,...,s_n)$, let I(s) be the set of index i such that $s_i = 1$ as introduced before. Then $\Delta E_{I(s)}(s) = E(s) - E(s_{I(s)})$. By definition, $s_{I(s)} = \mathbf{0} = (0,0,...,0)$ and let $E(\mathbf{0})$ be the reference value in evaluating E(s). More simply put, assume that $E(\mathbf{0}) = 0$ and define E(s) $= \Delta E_{I(s)}(s)$. Let $\widetilde{w_{kl}} = \widetilde{s_k}\widetilde{s_l} w_{kl}$ for $k,l = 1,2,...,n$ then we have

$$\begin{aligned} E(s) &= \Delta E_{I(s)}(s) \\ &= \sum_{i\in I(s)} \Delta E_i(s) + \frac{1}{2}\sum_{i,j\in I(s)} c_i\widetilde{w_{ij}} \\ &= \sum_{i=1}^{n} \Delta E_i(s)\, s_i + \frac{1}{2}\sum_{i,j=1}^{n} c_i\widetilde{w_{ij}} s_i s_j \\ &= -\sum_{i=1}^{n} c_i d_i\, s_i + \frac{1}{2}\sum_{i,j=1}^{n} c_i w_{ij} s_i s_j. \end{aligned}$$

Substituting $d_i = \sum_{j=1}^{n} w_{ij}s_j - \theta_i$ we have $E(s) = -\frac{1}{2}\sum_{i,j=1}^{n} c_i w_{ij} s_i s_j + \sum_{i=1}^{n} c_i s_i \theta_i$.

Multiplying by two, for notational convenience, we have a desired Lyapunov function as below.

Theorem 2.

Let a connection matrix W be a zero-diagonal and quasi-symmetric with respect to a positive diagonal matrix C whose i-i element is c_i for $i = 1,2,...,n$. Then under asynchronous operation mode, the following function is monotone non-increasing.

$$E_a(t) = -\sum_{i,j=1}^{n} c_i w_{ij} s_i(t) s_j(t) + 2\sum_{i=1}^{n} c_i s_i(t)\theta_i \quad .$$

Following an argument in Goles[2], we can also show that the following is a Lyapunov function for *synchronous* quasi-symmetric neural networks.

$$E_s(t) = -\sum_{i,j=1}^{n} c_i w_{ij} s_i(t) s_j(t-1) + \sum_{i=1}^{n} c_i (s_i(t) + s_i(t-1))\theta_i \quad .$$

In fact, we have

$$E_s(t) - E_s(t-1) = - \sum_{i=1}^{n} c_i d_i(t-1)(s_i(t) - s_i(t-2)) \text{ where } d_i(t-1) = \sum_{j=1}^{n} w_{ij} s_j(t-1) - \theta_i.$$

If $d_i(t-1) > 0$ then $s_i(t) = 1$ which means $s_i(t) - s_i(t-2) \geq 0$. If $d_i(t-1) < 0$ then $s_i(t) = 0$ which means $s_i(t) - s_i(t-2) \leq 0$. In both cases, we have $c_i d_i(t-1)(s_i(t) - s_i(t-2)) \geq 0$ because c_i is positive. Thus $E_s(t)$ is a monotone non-increasing function of t and we have then that the cycle lengths of the corresponding global state transition are at most two.

4. Concluding Remarks

It has been known [3] that there is a particular Lyapunov function for asynchronous model neural networks with zero-diagonal and symmetric connection matrices. Our questions are : why these restrictions are necessary and why that particular form for Lyapunov function. To answer these questions we first characterized the existence of *state evaluation functions* through the properties on their difference functions. Then we have shown that there is a class of *asymmetric* neural networks which have Lyapunov functions. The class is called *quasi-symmetric* because the weight matrix W should be such that CW is symmetric for a positive diagonal matrix C. The result was obtained by assuming linear functions for the definition of difference functions. The other possibilities have to be examined in order to see if there were other classes of neural networks that have Lyapunov functions.

The analyses of possible forms of energy functions for asynchronously operating neural networks done here are also relevant when we want to speed up the network operation. That is, the formulas for the energy difference given here enable one to decide easily when it is possible to carry out parallel state transition keeping an energy function decreasing.

We have also shown that quasi-symmetric neural networks have Lyapunov functions under synchronous operation mode.

Although the class seems to be still rather restrictive, it has been shown elsewhere that an interesting class of neural networks with effector and receptor parameters has common elements with it[4].

References

[1] Feldman, J.A. (1987) Energy Methods in Connectionist Modelling. in *Pattern Recognition Theory and Applications* (Eds. Devijver, P.A. and Kittler, J.) Springer-Verlag 223-247.

[2] Goles, E. (1987) Lyapunov Functions Associated to Automata Networks in *Automata Networks in Computer Science* (Eds.Fogelman, F., Robert, Y., and Tchuente, M.) Manchester University Press.58-81.

[3] Hopfield, J.J. (1982) Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proc. Natl. Acad. Sci. USA* **79**, 2554-2558.

[4] Kobuchi, Y. (1989) Asymmetric Neural Networks with Effector and Receptor Parameters. in *Mathematical Topics in Biology.* RIMS Report No. 678.Kyoto Univ. 85-100.

[5] McCulloch, W.S. and Pitts, W. (1943) A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics* **5**, 115-133.

An Orthogonal Projection Type of Associative Memory

Kiyotoshi MATSUOKA

Division of Control Engineering, Kyushu Institute of Technology
Sensui-cho 1-1, Tobata, Kitakyushu, 804 Japan

1. Introduction

Associative memory has been one of the main subjects in the history of artificial neural networks. In its basic paradigm, a given set of pairs of patterns $(s^{(m)},y^{(m)})$ $(m=1,..,M)$, sometimes referred to as prototype patterns, are stored in the form of strength of connections linking neuron-like elements (neurons), and when the network is given a noisy or incomplete pattern of $s^{(m)}$ as a key pattern, it produces or recalls, ideally, the corresponding output $y^{(m)}$.

From the viewpoint of the principle of the association, associative networks can be classified into two types, the correlation type and the orthogonal projection type. It is well known that the latter type of networks have a desirable associative ability than the former type, but, in conventional models, the weights of connections cannot be determined by local calculation as correlation calculation for the prototype patterns, requiring inverse or pseudoinverse calculation. Although the connection weights can also be determined by learning based on some error correction paradigm, some sophisticated mechanism may be required to embody this in hardware.

This paper proposes a new model of associative networks in which the orthogonal projection operator is implemented in a particular structure of cross connections between neurons. As a result, this network has several advantageous properties compared to conventional models from practical points of view.

2. Proposed Model

If prototype pattern vectors $s^{(m)}$ $(m=1,..,M)$ are linearly independent, the optimal associative mapping for the pairs $(s^{(m)},y^{(m)})$ $(m=1,..,M)$ is given by

$$y = Y(S^TS)^{-1}S^Ts = YS^+s \tag{1}$$

[Kohonen 1987]. Here, s and y are a key (input) pattern vector and an output pattern vector, respectively. $Y \triangleq [y^{(1)},..,y^{(M)}]$, $S \triangleq [s^{(1)},..,s^{(M)}]$. "+" represents the pseudoinverse of a matrix. Obviously, (1) can be realized by a two-layered network with feedforward connections as shown in Fig.1.

The same mapping can be embodied by a dynamic network shown in Fig.2 (solid lines). It consists of four layers; input, recognition, supplementary and output layers. The function of the input neurons is only to sense external signal s and transmit it to the neurons on the recognition layer. The recognition neurons, whose number is the same as that of the prototype pairs, receive the signals from the sensing layer and from the supplementary layer with certain weights, and send the output x to the supplementary layer and to the output layer. The supplementary layer with the same number of neurons as the input layer receives the signal from the recognition layer and negatively feeds the output v back to the recognition layer. The output neurons sum up the signals from the recognition neurons and emit the output y.

The recognition neurons are integral elements and the others are all static linear elements. The network equations are

$$\tau\, dx/dt = S^Ts - S^Tv \tag{2}$$

$$v = Sx \tag{3}$$

$$y = Yx. \tag{4}$$

Here, τ is a time constant that specifies the response time of the recognition neurons. [In case that, for the recognition neurons, we adopt the familiar elements with time lag of first order, we have only to add self-excitation to them; $\tau dx/dt+x=S^Ts-S^Tv+x$.] x, v, and y converges, for constant s, to the following stationary state:

$$x(\infty) = S^+s \tag{5}$$

$$v(\infty) = SS^+s \tag{6}$$

$$y(\infty) = YS^+s = YS^+v(\infty) \tag{7}$$

If the time constant, τ, is small enough for the network state to converge quickly to the stationary state, this network may be considered to have virtually the same characteristic as (1). (6) shows that the final pattern of the supplementary layer $v(\infty)$ becomes the orthogonal projection of s onto Π_s, the space spanned by $s^{(m)}$ (m=1,..,M). From (7) one can see that the orthogonal component to Π_s in s, possibly noise, is eliminated by this operation.

An interesting aspect of this network is that the neurons on the recognition layer behave as the so-called "grandmother cells" for the prototype inputs. Indeed, substituting $s=s^{(m)}$ into (5) leads to

$$x(\infty) = \delta^{(m)}, \tag{8}$$

where $\delta^{(m)}$ is the vector the m-th element of which is unity and the others are zeros.

The total number of the connections of this network is M(3I+J) (M, I, J: numbers of the neurons in the recognition, input (supplementary), and output layers). If M is much smaller than I and J, the connection number is smaller than that of Fig.1 (IJ).

3. Properties of the Model

Although, as far as the stationary state is concerned, the present network is mathematically equivalent to the forward-connection network (Fig.1), there exists several important distinctions between them from practical points of view.

Property 1 In order to obtain the connection weights of the network, no (pseudo)inverse nor correlation calculation is necessary. They can be determined directly from the prototype patterns without any calculation.

Property 2 As a result, addition or deletion of some prototype patterns is made by local modification of the connections. Suppose that the prototypes $(s^{(m)},y^{(m)})$ (m=1,..,M) have already been memorized and then another pair of prototype patterns $(s^{(M+1)},y^{(M+1)})$ is newly presented. In the present model the new network can be constructed simply by adding a new recognition neuron and connecting it to the other neurons with appropriate weights, requiring no modification for the old connections. In the model of Fig.1, on the other hand, the old weight of every connection needs to be altered completely again.

Property 3 The signs of the connections can be fixed if all components of the patterns to be memorized take nonnegative values. In (1) the components of YS^+ take positive as well as negative values in general. As for the present model, on the other hand, the connection weights from the supplementary layer to the recognition layer are all nonpositive, and the other connection weights are all nonnegative. This property is convenient when one tries to design the network in hardware.

Property 4 Suppose that it is known that some elements of the input, say the $j_1,..,j_N$-th elements, has an defect. Then, the optimal estimation of y based on the remaining data becomes

$$y = YS'^{+}s', \qquad (9)$$

where S' and s' are the matrix and the vector which are obtained by removing the $j_1,\ldots,j_N$-th rows from S and s, respectively. (9) can easily be realized by setting the values of the $j_1,\ldots,j_N$-th elements of the input and supplementary layers at null. In the network of Fig.1 the connection weights need to be altered to achieve this.

<u>Property 5</u> The orthogonal projection operator, SS^+, itself has no function to eliminate the noise component in s which is parallel to Π_s. However, if the patterns to be recalled are binary, i.e. $y_j^{(m)}=\pm1$, the performance of the network are considerably enhanced by adding binary threshold elements to the output layer (Fig.2, dotted lines, top). The characteristic of the threshold elements under consideration is

$$y'_i = g(y_i) = 1\ (y_i>0) \quad 0\ (y_i=0) \quad -1\ (y_i<0),$$

where y' is the output pattern of the threshold elements. Namely, the outputs of the linear elements are rectified to take +1 or −1 by the threshold elements. This network has an interesting property. Just after the input is presented to the input layer, the output becomes with x(0)=0

$$y' = g(y) \sim g(YS^Tst) = g(YS^Ts) \qquad (0<t\ll\tau), \qquad (10)$$

For $t\to\infty$, on the other hand, v approaches

$$y' \sim g(YS^+s). \qquad (11)$$

These relations imply that the single network works as a correlation type network as well as an orthogonal projection type network, depending on the time when the output is observed.

<u>Property 6</u> If the prototype inputs are binary, i.e. $s_i^{(m)}=\pm1$, the network can be modified to a feedback type of network, which is a variation of the model proposed by Personnaz et al. (1985) to hetero-associative memory. It is obtained by adding the threshold elements to the supplementary layer and connecting them back to the input terminal (Fig.2, dotted lines, bottom). Here, the nonlinear elements do not only function as threshold elements but also as sampling and holding elements, which hold the output of the supplementary layer at g(u(kT)) during time interval $kT\leqq t<(k+1)T$ (k=0,1,2,..). The network equations are

$$\tau\, dx/dt = S^Tv' - S^Tv \qquad (12)$$
$$v = Sx \qquad (13)$$
$$v'(t) = g(v(kT)) \qquad (kT\leqq t<(k+1)T) \qquad (14)$$
$$y = Yx. \qquad (15)$$

In this network, the key pattern is given as the initial values of the threshold elements; i.e., $v'(t)=s$ $(0\leqq t<T)$.

If T is large enough (or τ is small enough) and, thus, u(t) converges nearly to a stationary state for v=g(u(kT)) during each interval, v(kT) evolves approximately as

$$v((k+1)T) = g(SS^+v(kT)) . \qquad (16)$$

This is just the same as the model proposed by Personnaz et al. (1985). If v converges to some $s^{(m)}$, y converges to the corresponding prototype pattern $y^{(m)}$.

4. Conclusion

We have derived a novel associative memory that performs optimal associative mapping (orthogonal projection), and have showed that it has several advantageous properties from practical points of view.

References

Kohonen, T.: *Self-Organization and Associative Memory* (Second ed.)

Springer-Verlag (1987)

Matsuoka, K.: An associative network with cross inhibitory connections. (To appear in Biol. Cybern., 1989)

Matsuoka, K.: A Model of orthogonal auto-associative networks (To appear in Biol. Cybern.)

Personnaz, L., Guyon, I. Dreyfus, G.: Information storage and retrieval in spin glass like neural networks. J. Phys. Lett. (Paris), 46, L359-L365 (1985)

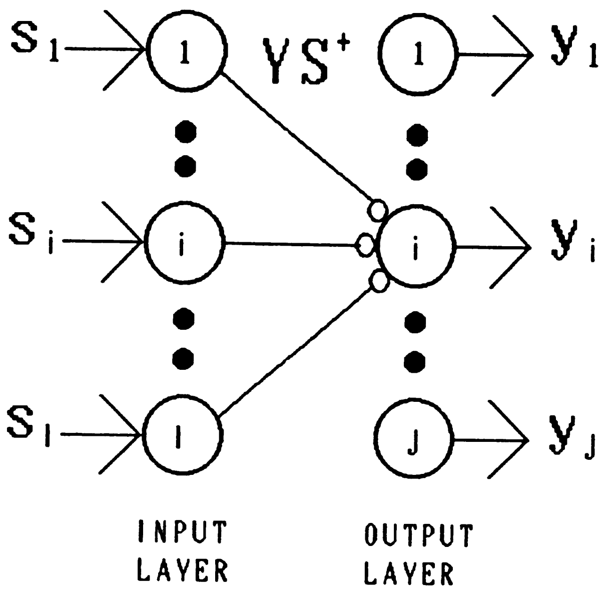


Fig. 1 Two-layer network.

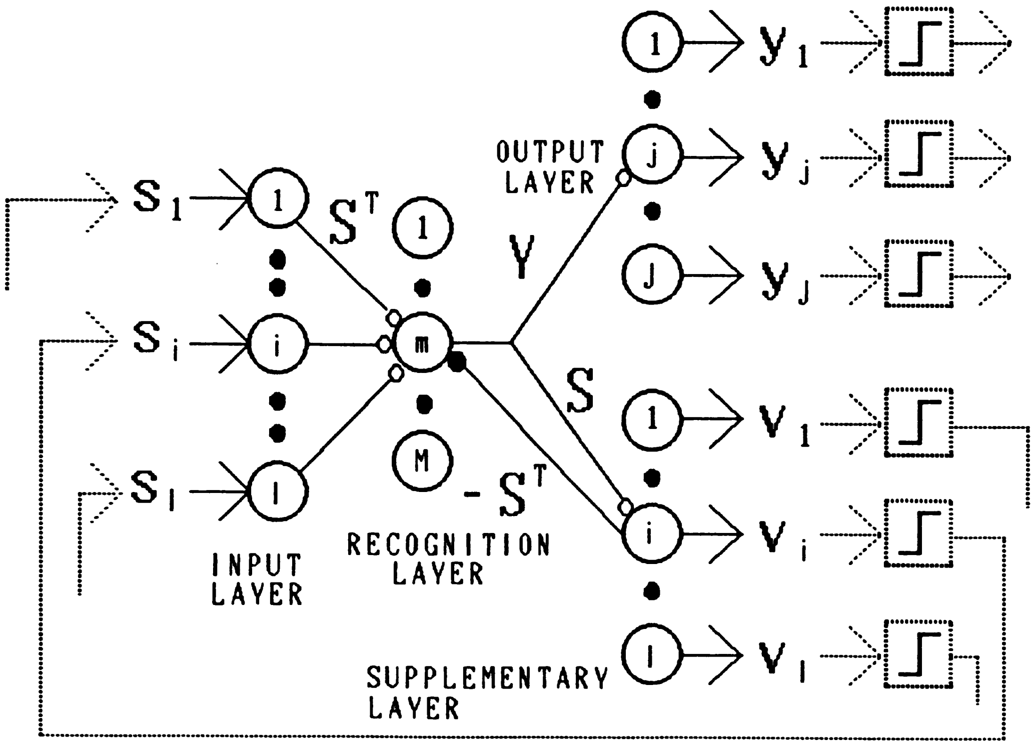


Fig. 2 Proposed model.

An Efficient Algorithm for Annealing Schedules in Boltzmann Machines

Robert Richards
Stanford University, Sweet Hall, 3rd Floor, Stanford, CA 94305

Abstract - An efficient algorithm for annealing schedules in Boltzmann machines is developed and demonstrated. A current deficiency of Boltzmann machines is the difficulty in determining good annealing schedules, consequently empirical determinations have been applied in practice. Theoretically no finite length annealing schedule guarantees that the optimum determined is the global. However, different schedules yielding the same probability of finding the global optimum can vary by orders of magnitude in computational effort. The algorithm developed here takes advantage of properties of Boltzmann machines known to be important to annealing schedules but often ignored.

1. Background

The Boltzmann machine (BM) offers a generalized computational approach that can be applied to the basic research issues of search, representation and learning and has a rigorous mathematical formalism. It can also be considered a model for a massively parallel implementation of the simulated annealing algorithm [Aarts & Korst, 1989a] which can be used to solve combinatorial optimization problems. Optimization is performed by the units which make up the BM attempting to reach a maximal consensus about their individual states, subject to the constraints set by the connection strengths, the connections having been learned for a particular problem. The units adjust their states (0 -> 1 or vice versa) to the states of their neighbors, i.e. the units to which they are directly connected [Aarts & Korst, 1989a].

To adjust the states of the individual units a probabilistic state transition mechanism, which is a function of the randomness, T, is employed. Randomness is necessary to avoid local optima. The randomness is described by the randomness parameter, which starts at high randomness (i.e. perturbations that cause a better solution are chosen with virtually equal chance as those that cause worse solutions) and is slowly decreased to zero randomness (i.e., only perturbations which cause better solutions will be chosen). When the randomness reaches zero the optimization is complete and a local optimum has been reached. The local optimum's degree of optimality is highly dependent on the path used from high randomness to zero randomness. This path is termed an annealing schedule.

A weakness of Boltzmann machines is that the proper path from high randomness to low randomness, has been difficult to determine. Many attempts have been forwarded both theoretical and empirical [Van Laarhoven & Aarts, 1987], however much room for improvement still exists. Theoretically no finite length path will guarantee that the optimum determined is the global optimum [Geman & Geman, 1984], however, annealing schedules having similar probabilities of discovering the global optimum may vary by orders of magnitude in computational effort required to complete the annealing process. To help ensure that a global optimum has been found, multiple runs are made to see if a better result is forthcoming.

Merit of an annealing schedule is determined by the probability of determining the global optimum given equivalent computations.

2. Annealing Schedule

In order to avoid local minimum or to escape from one, the Boltzmann machine allows for jumps to configurations of higher energy as long as the randomness parameter is greater than zero. An algorithm with this property was introduced by Metropolis [Metropolis, et al, 1953] to study average properties of thermodynamic systems and has since been applied to problems of constraint satisfaction [Kirkpatrick et al, 1983]. A form of the Metropolis algorithm that is suitable for parallel computation has been adopted for the Boltzmann machine [Hinton, Sejnowski & Ackley, 1984].

The transition rule is shown in Equation Set 1.

At high randomness, the BM performs a search of the coarse overall structure of the space of global states, and will find a good minimum at that coarse level. As the randomness is lowered, it respond to ever smaller energy differences and will find one of the better minima within the coarse-scale minimum it discovered at high randomness. Kirkpatrick [Kirkpatrick, et al, 1983] has shown that this way of searching the coarse structure before the fine is very

If the energy gap between the on and off states of the kth unit is ΔE_k then regardless of the previous state, set $s_k = 1$ with probability,

$$\text{probability}(s_k(t) = 1) = \frac{1}{1 + \text{Exp}\left(\frac{\Delta E_k}{T_{t-1}}\right)} \qquad (2.1)$$

where; T_{t-1} ≡ Randomness parameter at time t-1

s_k ≡ Activation of unit k - value is 1 or 0

The energy gap between the on and off states of the kth unit is ΔE_k, and is calculated by assuming the kth unit is on and then taking the negative sum of the weights of all connections to other units which are on, that is,

$$\Delta E_k = -\sum_{j \text{ connected to } k} w_{jk}\, s_j\, 1. \qquad (2.2)$$

At low randomness there is a strong bias in favor of states with low energy, while at high randomness the inclination toward making the move towards lower energy approaches that of a fair coin toss.

Equation Set 1

$$c_p(T) = \frac{\sum E^2 - \{\sum E\}^2}{T^2} \quad \text{[sum over all units]} \qquad (2.3)$$

where; $\sum E^2$ ≡ The average of the energy squared at randomness T

$\{\sum E\}^2$ ≡ The square of the average energy at randomness T

E ≡ The energy of unit k = $-\sum_{j \text{ connected to } k} w_{jk}\, s_j\, s_k$.

This is a global property of the Boltzmann machine which is proportional to the rate at which entropy - disorder - decreases as the randomness decreases [Smolensky, 1986].

$$Q = \int c_p \, \partial T \qquad (2.4).$$

Equation Set 2

effective for combinatorial problems.

The problem lies in finding efficient annealing schedules.

As has been noted by Kirkpatrick and others there are properties that can be exploited in the search for better annealing schedules. Most important of these is the specific heat, c_p, and the "heat output", Q, see Equation Set 2 for definitions.

3. Properties of an Illustrative Boltzmann Machine

To gain a understanding of the problem to be solved, quantitative values of the important properties of an illustrative Boltzmann machine are determined. To see how energy and specific heat of the Boltzmann machine change as a function of the randomness, measurements were made for the simple annealing schedule of $T(t+1) = 0.99T(t)$, see Figure 1. Note that the energy values have been scaled so the lowest possible energy is 1. It can be seen that at higher randomness values that the energy fluctuates greatly.

As mentioned, the specific heat gives an idea about how slow or fast the randomness should be decreased. The effect of the specific heat is not immediately clear from Figure 1. After an area of high specific heat, it is the peak of the energies that is changed, that is the peaks of the energies will never reach there previous peaks again. For example, at randomness values higher than 1.3 the peaks do not seem to be decreasing significantly from the starting randomness of 2.5, however after the high specific heat near 1.3 the peaks are never again as high as they were before the region of high specific heat.

4. The Algorithm

The foundational works of all Boltzmann machine annealing schedule research include; 1) the research by Metropolis [Metropolis, et al, 1953] where the stochastic relaxation technique was first developed. 2) The application of the stochastic relaxation technique developed by Metropolis to problems of combinatorial optimization by Kirkpatrick [Kirkpatrick et al, 1983], this optimization technique was termed "simulated annealing". 3) The two papers that introduced the Boltzmann machine neural network model, by Hinton, Sejnowski and Ackley [Hinton & Sejnowski, 1983],[Hinton, Sejnowski & Ackley, 1984].

These papers established the Boltzmann machine model as one which could implement simulated annealing opti mization in massive parallelism.

This development furthers the recent advances made by Aarts, Korst and Van Laarhoven [Aarts & Korst, 1989b], [Van Laarhoven & Aarts, 1987]. The algorithm is as follows with the development shown in Equation Set 3.

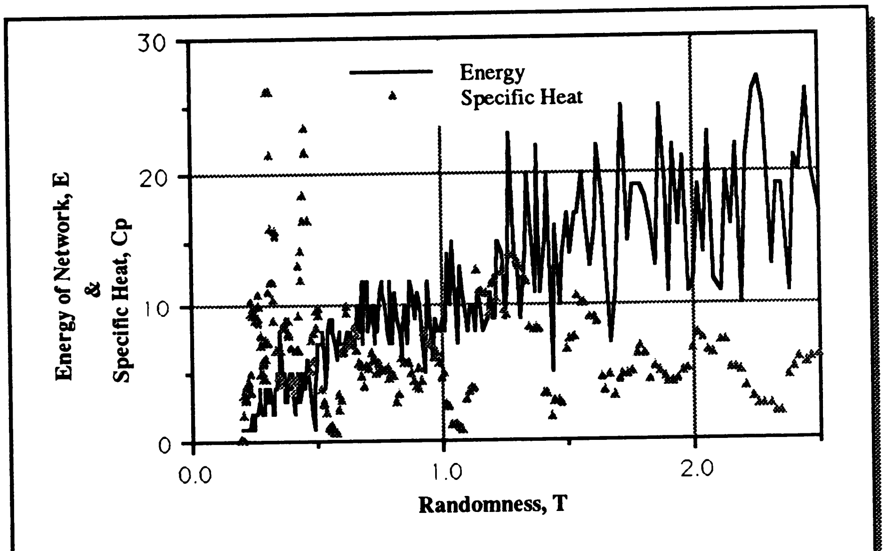


Figure 1: Specific Heat's Effect on Energy

The main assumption is that quasi-equilibrium holds. Quasi-equilibrium states that for a small enough change in randomness, if the original vector of configurations at randomness T_t, $q(T_t)$, is in equilibrium or quasi-equilibrium, then at randomness T_{t+1}, $q(T_{t+1})$, will be in quasi-equilibrium. That is $q(T_{t+1})$ is 'sufficiently close' to the true equilibrium at T_{t+1}.

$$\| q(T_t) - q(T_{t+1}) \| < \varepsilon \tag{4.1}$$

for some small positive value of ε and for all t greater than zero. With this assumption Aarts and Korst [Aarts & Korst, 1989b] have shown the following holds,

$$T_{t+1} = \frac{T_t}{1 + \frac{T_t * \beta}{E(t) - E_{opt}}} \tag{4.2}$$

Of course, the weakness of this equation is that the function's optimum needs to be known. To alleviate this problem a best estimate is made of $E(t) - E_{opt}$, which is bounded above by [Van Laarhoven, 1989],

$$E(t) - E_{opt} \le \langle E(t)\rangle - E_{opt} + 3T_t \sqrt{c_p} \tag{4.3}$$

Where $\langle E(t)\rangle$ is the expected energy, and is defined as,

$$E_{inf} - \int_0^{T(t)} c_p \,\partial T. \tag{4.4}$$

The best estimate for the E_{opt} term is then,

$$E_{opt} \approx E_{inf} - \int_0^{\infty} c_p \,\partial T. \tag{4.5}$$

Substituting (4.5) into (4.3) and taking the $\le$ as an equality yields,

$$E(t) - E_{opt} \approx \int_0^{T(t)} c_p \,\partial T + 3\, T_t \sqrt{c_p(t)}. \tag{4.6}$$

Substituting this into (4.1) yields the annealing schedule algorithm, shown in equation (4.7).

Equation Set 3

$$T_{t+1} = \frac{T_t}{1 + \frac{T_t * \beta}{\int_0^{T(t)} c_p \,\partial T + 3T_t\sqrt{c_p(t)}}} \tag{4.7}$$

The only parameter which can not be determined is the constant β. By increasing β a faster annealing schedule will result but with a lower probability of finding the global optimum.

5. Numerical Results

Annealing schedule algorithms that utilize specific heat have been demonstrated to be advantageous in the simulated annealing literature [Van Laarhoven & Aarts, 1987]. The benefit of considering c_p grows as the difficulty of the problem increases. Not surprisingly, past comparisons have used more difficult problems. The comparison here uses the relatively simple BM mentioned above, and the proposed algorithm is compared with the simple but highly efficient schedule, $T_{t+1} = \alpha\ T_t$. In this schedule α is a constant, having a similar effect as β in the proposed algorithm. The results of these two schedules is displayed in Figure 2, note that the logarithms used in the curve fit are to the base 10. Even with the minimal cues present for the proposed algorithm to exploit, a consistent improvement is found.

6. Conclusions

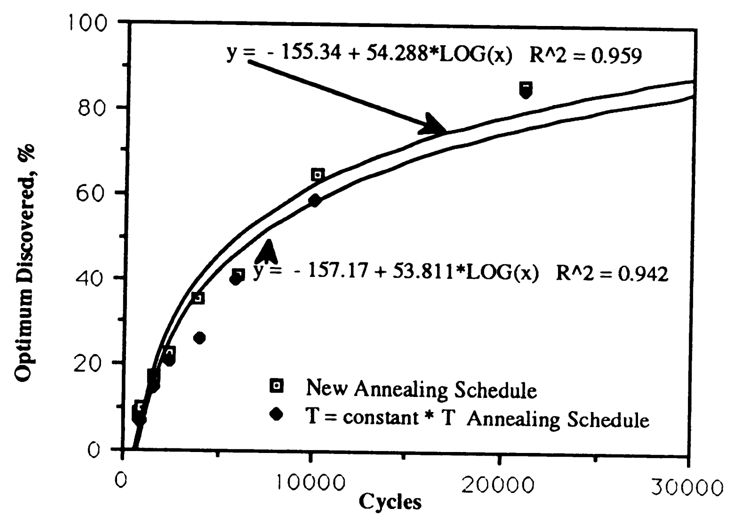


Figure 2: Comparison of Proposed Annealing Schedule and Good Empirical Schedule

The algorithm presented here is advantageous over empirical schedules and is shown to be better than a highly efficient algorithm which does not use the Boltzmann machine's global properties to benefit. Theory provided the guide in the proposed algorithm's development. From this has come the first algorithm to not only utilize specific heat, but also heat output. Of course, only further investigation will determine if this theoretical approach is the best in practice.

As with all annealing schedule algorithms the proposed algorithm leaves a parameter for the user to vary (β). Unfortunately as with previous algorithms the correlation between the values of the parameter and the probability of discovering the global optimum are not known a prior. However, for any probability of discovering the global optimum the proposed algorithm will use the least cycles.

The greatest virtue of this algorithm is its general applicability to all Boltzmann machines. While a particular empirical annealing schedule, such as the $T_{t+1} = \alpha T_t$ example used above, may be efficient for certain BMs, there is no guarantee that it will be efficient for others. Just as neural networks are adaptive, the proposed annealing schedule algorithm will tailor itself to maximize its efficiency for every particular Boltzmann machine.

References

Aarts, E.H.L., and Jan H.M. Korst, (1989a) Computations in Massively Parallel Networks Based on the Boltzmann Machine: A Review. *Parallel Computing*. Vol. 9. pp. 129-145.

Aarts, E.H.L., and Jan H.M. Korst, (1989b) Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing. New York, N.Y: John Wiley & Sons.

Geman, S. and D. Geman, (1984) Stochastic Relaxation, Gibbs Distributions and the Baysian Restoration of Images. *IEEE Transactions of Pattern Analysis and Machine Intelligence*. 6, 721-741.

Hinton, G. E., and T.J. Sejnowski, (1983) Optimal perceptual Inference. *Proc. IEEE Conference on Computer Vision and Pattern Recognition*. Washington, DC.

Hinton, G. E., Sejnowski T.J., and D.H. Ackley, (1984) Boltzmann Machines: Constraint Satisfaction Machines that Learn. *Technical Report CMU-CS-84-119*. Carnegie-Mellon University.

Kirkpatrick, S., C.D. Gelatt Jr., and M.P. Vechi, (1983) Optimization by Simulated Annealing. *Science*. 220, 671-680.

Metropolis, N., et al, (1953) Equation of State Calculations by Fast Computing Machines. *Journal of Chemical Physics* 21, 1087-1092.

Smolensky, P., (1986) "Information Processing in Dynamical Systems: Foundations of Harmony Theory," ch. 6 in Parallel Distributed Processing, Volume 1: Foundations. Rumelhart & McClelland (Ed), MIT Press, Cambridge, pp.194-281.

Van Laarhoven, P.J.M., and E.H.L. Aarts, (1987) Simulated Annealing: Theory and Applications. Boston, MA: D. Reidel Publishing Company.

Van Laarhoven, P.J.M., (1989) Theoretical and Computational Aspects of Simulated Annealing. Amsterdam, The Netherlands: CWI Tracts.

ON THE LEARNING POWER OF NETWORKS WITH A BOUNDED FAN-IN LAYER

(extended abstract)

Haim Shvaytser
David Sarnoff Research Center, Princeton, NJ 08543-5300.
Email: haim%sarnoff@Princeton.edu

Abstract

We analyze the learning power of networks with one hidden layer and the additional constraint that each node in the hidden layer can have at most a constant number of connections. We give conditions which guarantee with high confidence that the following training strategies are optimal: (a) Fixing the connections and weights of the hidden layer nodes and training the output node with the perceptron algorithm. (b) Choosing the connections of the hidden layer nodes at random and training the output node with the perceptron algorithm. We also show that networks of this type are capable of learning (in the sense of Valiant) nontrivial classes of boolean formulae in conjunctive and disjunctive normal forms.

1 Introduction

Models of neural nets can be characterized by the net topology, node characteristics, and learning algorithms. The simplest model of this type, the perceptron, has only one node with trainable weights. The perceptron power to express complicated functions of its input has been analyzed by Minskey and Papert in [8].

Multi-layer neural networks (see, e.g., [9]) were suggested as natural generalizations of perceptrons. Even though they are superior to perceptrons in computing complicated functions, they appear to be much harder to train. Recent results point out some inherent difficulties in both the loading problem [4,2] (determining weights for correct classification of the training examples) and the generalization problem [1,6]. This suggests that networks should be designed with some "hardware constraints" to enable efficient learning. The constraints that we consider here are:

> Each node in the hidden layer can have at most k connections with non-zero weights, where k is a constant independent of n, the number of input variables.

We call networks that meet these constraints *k-fan-in networks.* An example is shown in Figure 1. Notice that the constraints do not imply that the connections are hard-wired. Both the connections and the coefficients are to be determined during the training. The question that we consider here is: *can a k-fan-in network be trained efficiently and learn (in the sense of Valiant) non-trivial classes of boolean functions?*

We observe that potentially any network can be considered as a k-fan-in network by taking k to be the maximum fan-in of a hidden layer node. However, since the asymptotic complexity of our techniques is proportional to n^k, they are impractical *for large values of* n when k is not a constant. We show that if the network computes Boolean functions then:

- There is a k-fan-in network with a hard-wired hidden layer of $o(n^k)$ nodes and with fixed (non-trainable) weights that is as powerful as any k-fan-in network.
- With high confidence, the same computation power can be obtained with randomly chosen connections of $o(n^k \log n)$ hidden nodes with fixed (non-trainable) weights.
- These networks are capable of learning (in the sense of Valiant) k-CNF and k-DNF formulae by using the perceptron algorithm.

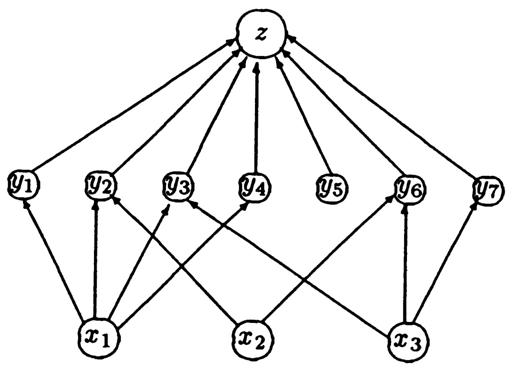


Figure 1: A 2-fan-in network. At most 2 incoming connections to nodes in the second layer

2 Representation power

We denote the input variables by $x_1, \cdots, x_n$, the values of the hidden layer nodes by $y_1, \cdots, y_m$, and the value at the output node by z. (See the example in Figure 1.) The values of these variables are either 0 or 1. As usual, each node computes a weighted linear summation and a threshold. The boolean functions that are computed by the network can be expressed in an algebraic form as multilinear functions.

Claim 1 *If $y = 1$ whenever $\sum_{i=1}^{k} a_i x_i > b$ and 0 otherwise then y can be expressed as a multilinear function of degree no greater than k of the variables $\{x_1, \cdots, x_n\}$.*

Proof: See [8]. Example: $x_1 + x_2 > 0 \Leftrightarrow y = x_1 + x_2 - x_1 x_2$ a multilinear function of degree 2, and $x_1 - 0.5x_2 + 0.5x_3 > 0.1 \Leftrightarrow y = x_1 + x_3 - x_1 x_3 - x_2 x_3 + x_1 x_2 x_3$, a multilinear function of degree 3.

Claim 2 *k-fan-in networks compute functions of the type $L_k(x_1, \cdots, x_n) > 0$, where $L_k(x_1, \cdots, x_n)$ is a multilinear function of degree bounded by k of the variables $x_1, \cdots, x_n$.*

Proof: (sketch) It follows from Claim 1 and the fact that the node z computes a linear function of its input before thresholding.

Remark: Therefore, k-fan-in networks are no more powerful than the order-restricted perceptrons of Minskey and Papert [8].

We conclude that no k-fan-in network can be more powerful than a network in which the set $y_1, \cdots, y_m$ spans the vector space of *all* multilinear functions of degree bounded by k. We denote by $\Phi(n,k) = \sum_{j=0}^{k} \binom{n}{j}$ the dimension of this vector space. (For a constant k, $\Phi(n,k) \approx n^k$.)

Claim 3 *Let N_H be a network with $m = \Phi(n,k)$ nodes in the hidden layer, such that (a) there are hard-wired connections from each node $\{y_i\}$ to a* different *subset of $x_1, \cdots, x_n$, of size bounded by k. (b) All the hidden layer connections have weights 1 and thresholds 0. Then N_H is as powerful as any k-fan-in network.*

Proof: (sketch) Since the dimension of the vector space of all multilinear functions of degree bounded by k is $\Phi(n,k)$, it is enough to show that the functions $y_1, \cdots, y_{\Phi(n,k)}$ of N_H are linearly independent multilinear functions of degree bounded by k. If y_i is connected to $\{x_{i1}, \cdots, x_{ir}\}$

($r \leq k$) then $y_i = 1$ if and only if $\sum_{j=1}^{r} x_{ij} > 0$. (It is easy to see that in fact, $y_i = x_{i1} \vee \cdots \vee x_{ir}$.) The proof follows from Claim 1 and the fact that the functions y_i defined as above are linearly independent.

Remark: Other choices of threshold values are also possible. The simplest choice is: $\sum_{j=1}^{r} x_{ij} > r - 1$, since then we have $y_i = x_{i1} \cdot \ldots \cdot x_{ir}$. However, Choosing 0 as threshold allows us to choose *the same* threshold for all nodes in the hidden layer.

The computation power of the network N_H in Claim 3 appears to be the result of its carefully chosen hidden layer connections. In choosing these connections each node has to know the connections of all other nodes. Therefore, choosing the connections for N_H cannot be done in parallel. In the following claim we describe a reliable distributed technique for choosing the connections.

Claim 4 *Let N_R be a network with $(1+\epsilon)k\Phi(n,k)\log n$ nodes in the hidden layer. The connections and weights of N_R are determines in the following way: each node chooses at random*[1] *a subset of at most k variables and forms connections with weights of* 1 *and threshold of* 0. *Then, with probability of at least $1 - O(1/\log^2 n)$ N_R is as powerful as any k-fan-in network.*

Proof: (sketch) It is enough to show that with probability of at least $1 - O(1/\log^2 n)$ each subset of $\{x_1, \cdots, x_n\}$ of size bounded by k is chosen at least once. This is an instance of the coupon collector problem (see, e.g., [3] page 225). The expected number for getting all subsets is $\Phi(n,k)(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{\Phi(n,k)}) \approx \Phi(n,k)\log\Phi(n,k)$, and the variance is $\frac{\pi^2}{6}\Phi(n,k)^2 - \Phi(n,k)\log\Phi(n,k) + O(\Phi(n,k))$. Claim 4 follows from the Chebyshev inequality, using the fact that $\Phi(n,k) \approx n^k$.

3 Learnability

We have shown in the previous section that when the number of hidden layer nodes is large enough, k-fan-in networks can be trained by (a) Fixing (or randomly choosing) the connections and weights of the hidden layer nodes. (b) Determining the weights of the connections of z, the output node, by some training procedure. Since there is only one node with trainable weights we can use the perceptron learning algorithm. This guarantees (see, e.g., [8]) that whenever a solution exists, the training converges after only a finite number of mistakes. Unfortunately, this does not guarantee *efficient* convergence.

From Theorem 11.1 in [8] it follows that if there are weights $w_1, \cdots, w_m$ such that $z = 1$ whenever $\sum_{i=1}^{m} w_i y_i > 0$ and 0 otherwise, and $\sum_{i=1}^{m} w_i y_i > \delta > 0$ for all positive examples, then the number of mistakes of the perceptron training procedure is bounded by:

$$\frac{\sum_{i=1}^{m} w_i^2 \cdot \sum_{i=1}^{m} y_i^2}{\delta^2}. \qquad (1)$$

Thus, the efficiency of the perceptron algorithm depends on the size of δ.

3.1 Learnability in the sense of Valiant

Valiant [10] suggested a complexity based definition of learnability. A class of concepts is learnable in the sense of Valiant only if there is a learning algorithm that runs in polynomial time[2] independent

[1] All subsets (of size bounded by k) have the same probability.

[2] The polynomial growth is with respect to some "natural" parameters of the concept class, such as n, the number of input variables and the concept size.

of the probability distribution of the examples. In this model, the perceptron does not always learn fast enough, because δ may be shrinking exponentially fast. However, as was observed by Littlestone [7], it is possible to prove learnability in Valiant's sense for specific concept classes.

By using the results of [1] one can compute a number $N(\epsilon)$ such that if a network produces correct classification for $N(\epsilon)$ examples chosen from an arbitrary probability distribution it will correctly classify at least a fraction of $1-\epsilon$ of the examples with confidence approaching certainty. This guarantees learnability in the sense of Valiant whenever $N(\epsilon)$ is polynomial. In the case of k-fan-in networks $N(\epsilon)$ is always polynomial.

Claim 5 *Let $N(\epsilon)$ be the number of examples that are required for valid generalization (see [1]) then the perceptron learning procedure requires no more than $N(\epsilon) \cdot \frac{\sum_{i=1}^{m} w_i^2 \cdot \sum_{i=1}^{m} y_i^2}{\delta^2}$ examples.*

Proof: (sketch) This number of examples guarantees that at least one run of $N(\epsilon)$ examples are correctly classified.

Corollary: *A concept class in learnable in Valiant's sense by a k-fan-in network whenever the separation δ in Equation 1 is inverse polynomially related to n, the number of input variables.*

3.2 Learnability of k-CNF and k-DNF

Boolean functions can be expressed in disjunctive normal form (DNF) and conjunctive normal form (CNF). DNF/CNF functions with bounded size of terms/clauses (k-DNF/k-CNF) are learnable in the sense of Valiant, while generalizations in certain directions lead to computational intractability [5]. Littlestone described in [7] a set of transformations that transform k-DNF and k-CNF into linearly separable functions. It is easy to see that the resulting functions can be computed by a k-fan-in network, and the separation δ is inverse polynomially related to n.

References

[1] E. B. Baum and D. Haussler. What size net gives valid generalization. *Neural Computation*, 1(1):151–160, 1989.

[2] A. Blum and R. L. Rivest. Training a 3-node neural network is NP-complete. In *Proceedings of the first workshop on computational learning theory*, pages 9–18. Morgan Kaufmann, 1988.

[3] W. Feller. *An introduction to probability theory and its applications*, volume I. WILEY, 3d edition, 1970.

[4] J. S. Judd. Complexity of connectionist learning with various node functions. Technical Report 87-60, Department of Computer Science, University of Massachusetts at Amherst, 1987.

[5] M. Kearns, M. Li, L. Pitt, and L. G. Valiant. On the learnability of boolean formulae. In *Proceedings of the nineteenth annual ACM symposium on theory of computing*, pages 285–295, May 1987.

[6] M. Kearns and L. Valiant. Learning Boolean formulae or finite automata is as hard as factoring. Technical Report TR14-88, Aiken Laboratory, Harvard University, 1988.

[7] N. Littlestone. Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[8] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. The MIT press, 1969.

[9] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing*. MIT press, 1986.

[10] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

Colored Noise Annealing Benchmark by Exhaustive Solutions of TSP

Harold Szu, Naval Research Laboratory, Code 5756, Washington, D.C. 20375-5000

Abstract: The shortest distances can be stochastically determined in a polynomial time for the Travelling Salesman Problems (TSP), employing the color noise generated by the fat-tail Cauchy probability density, $T/\pi (T^2 + X^2)$, which must be quenched with an inversely linear cooling schedule:$T= T_0/ (1+ t)$ [Phys. Lett. A122, 157, and Proc. IEEE 75, 1538]. The quenching of Cauchy colored noise must be consistently used in every iteration t-steps both in generating new states and in visiting some of the states only after passing a Cauchy acceptance criterion. Such an algorithm is known as Fast Simulated Annealing (FSA) [AIP Conf. Proc. V. 151, p.421 Snowbird 1986]. The FSA is t/log(t) faster than the Gaussian white noise simulated annealing algorithm being already better than the Monte Carol method held at a constant noise temperature. The performance of FSA is absolutely calibrated by comparing with the results obtained by a brute force search through all possible TSP solutions. The complete energy spectra that consist of all round-trip distances are computed for 4 cities up to 10 cities. While the FSA used about ten minutes or less to find global minima, the exhaustive search through hundred thousand possible cases had used several hours of CPU time on a Mac II (e.g. five hours for ten cities implies 50 hours for 11 cities). The shortest tours agreed with those found by FSA. FSA is superior because search for global minima required sampling less than 1%, with another 2% sampling to verify the stability. Thus, the traditional random sampling should be replaced with the better FSA algorithm, and with the best parallel Cauchy neural network Machine useful for Image Processing [Szu, Scheff,"Simulated Annealing Feature Extraction from Occluded and Cluttered Objects,"IJCNN-90, Wash. D.C., Jan. 15-18,1990].

KEYWORDS: Simulated Annealing, Cauchy Machines, Travelling Salesman Problem, Random Sampling

1. Cauchy Machine:

Neural networks for computing can mimic the liquid-solid phase transition which promises the minimum energy crystal state. Metropolis et al. [1] studied the phase transition by a numerical annealing algorithm in 1953, which had been recently adopted by Kirkpatrik et al. [2] for a VLSI circuit layout optimization in 1983. Then the neural network embodiment in Boltzmann Machine had been simulated by Hinton and Sejnowski[3]. A sufficient condition for the convergence was proven to be too slow to be useful; the temperature T(t) must be slowly cooled down to the zero degree according to the formula $T= To/ \log (1+t)$ of Geman and Geman[4]. Thus, in the interest of speeding up the annealing process and yet still finding the global minimum, Szu [5] applied the Cauchy colored noise to similarly derive the annealing schedule $T= To/ (1+t)$ to insure the complete search space available at all temperatures. This is known as the Fast Simulated Annealing (FSA), as opposed to the Gaussian white noise annealing known as the Classical Simulated Annealing (CSA). Since then the FSA has been studied for the termination formula: $\Delta E\, \Delta t = \text{Order}(1)$ in nonconvex optimizations [6], and applied to the N-dimensional bearing-fix problem [7,8]. Although the one-dimensional optical neural network embodiment of the FSA had been already referred to as a Cauchy Machine [9], the VLSI design was not completed. This shortfall is not because of any intrinsic property of the electronics versus the optics; but rather because a new Cauchy against-force acceptance criterion, which was not derived until recently [10], turns out to be local and thus can be distributed to each neurons taking the full advantage of parallel computing.

The total input firing rate u_i being summed at the ith tree of dendrites with thousands of concurrent channels with the dentrite weights W_{ij} has been propagated by ions and peptides through synaptic gaps to receivers, from thousands other neurons with output firing rates v_j:

$$u_i = \Sigma_j W_{ij} v_j + \theta_i, \qquad (1)$$

The total output v_i is transferred from the total input u_i Eq(2a) after passing the ith (axon hillock) threshold θ_i. Recently, to model interconnect growth or pruning, that is useful for dynamic reconfigurable layer nets, the author has extended the McCulloch-Pitts neuron to include the independent protein-actin-driven activity[14].

$$v_i = 1/ (1 + \exp(u_i/ T (t)); \qquad T_{ij} = 1/ (1 + \exp(W_{ij}/ T (t)) \qquad (2a,b)$$

To adopt Eq(2b)for a binary interconnect as the control parameter $T\to 0$, the author must furthermore generalize the Hopfield-like energy $E(v_i)$ to include $E(v_i,T_{ij})$, a second set independent variables: the axonic weights T_{ij}(e.g. in a top-down design fashion for the mini-max pattern clustering energy). In order to prove the convergence of the parallel asynchronous dynamics in the Lyaponov sense [14]:

$$dE/dt \le 0 \qquad (3a)$$

the following Hopfield hard-wired Eq(3b) coupled with Rumelhart soft-wired learning rule: $dW_{ij} /dt= -\partial E/\partial W_{ij}$ must be slightly generalized by a brittle-wired Eq(3c) to include the transfer from W_{ij} to T_{ij} Eq(2b)

$$du_i /dt= - \partial E/\partial v_i ; \qquad dW_{ij} /dt= - \partial E/\partial T_{ij} \qquad (3b,c)$$

Proof of Eq(3a): By the fact (energy gradient)^2 > 0, and the following chain rules:

$$dE/dt =(\partial E/\partial v_i)(dv_i/du_i)(du_i/dt) + (\partial E/\partial T_{ij})(dT_{ij} /dW_{ij})(dW_{ij} /dt)$$

we eliminate the above time derivatives in favor of the energy gradients using the righthand sides of Eq(3b,c). Only the positive nature of the transfer function slopes Eq(2,ab) is required for the absolute convergence. Q.E.D.

Using stochastic and cooling nature of neurons, neural networks have been referred to as Boltzmann Machines [3], demonstrated by the celebrated Net-Talk. Unfortunately, the learning is slow, and for the Gaussian white noise model, one can not integrate the Metropolis acceptance criterion to an elementary function, which gives, by using steepest descend approximation for the energy gain=(E_{new} - E_{old}) incured by neuronic decisions, the famous energy landscape up-hill acceptance criterion:

$$P_T = 1/[1+ \exp(gain/T)]$$

Such an energy landscape formula works all right on a conventional serial machine for one neuronic decision at one time. Efforts have been attempted by Hinton and Sejnowski to circumvent this shortfall, interpretating Eq(2a) as a local decision rule for individual Hopfield-like neurons in order to derive a specific hidden layer probabilistic learning in terms of weights(c.f Appendix of ref.[3]). Fortunately for Cauchy neurons, both the output v_i and the axonic weight T_{ij} are locally set to be one only if random numbers generated within [0,1] are less than acceptance functions computed also locally:

$$P_T(u_i) = =(1/\pi T) \int_0^\infty d\, X /[1+((X-u_j)/T)^2] = (1/2) + \arctan(u_j/T(t)) /\pi \qquad (4a)$$

$$P_T(W_{ij}) = (1/2) + \arctan(W_{ij}/T(t)) /\pi \qquad (4b)$$

Eq(4a) is integrated using the identical Metropolis formula[1] for the Cauchy state generating probability

$$G_T(x' \mid x'=x + X) = [T/\pi(T^2 + \mid X \mid^2)] \qquad (4c)$$

The random variable X is the distance between the old x and the new x', or for uniform angle between $\pm\pi/2$

$$X = T \tan(\theta) \qquad (4d)$$

Proof of Eqs(4c,d): Using $d\tan(\theta)/d\theta = 1/(1+\tan(\theta)^2)$, we replace $\tan(\theta)$ with X/T giving Eq(4)) Q.E.D. Moreover, the temperature is stepwise reduced [5-8]from an arbitrary initial temperature T_o:

$$T(t)= T_o/ (1+ t). \qquad (5)$$

Both random walks and random flights (long jumps) are responsible for the Cauchy variance divergence, which however turns out to be exactly integrable for the Metropolis new state acceptance criterion, Eq(4a,b).

2. Factorial Number Representation: A good mapping of a 2-D TSP on to a 1-D search space should preserve a neighborhood smoothness relationship. A smoothness spectrum would be desirable for a gradient descent method often used for a heuristic search technique. While a good mapping is desirable for a local gradient method, it is not needed for a semi-local FSA technique except a lower initial temperature T_o and thus earlier termination time in reaching the ground state within the spectrum resolution.

A good coding scheme must be 1-1 unique. Because the combinatorial nature of the TSP, a factorial number base system is adopted for the simplicity: (A) The real line x is sampled by the set of real integers x, using the function: Int( ); (B) Then, integers are made periodically in the module base set of (N-1)!, using the function: Mod(,); and (C) Such an integer number can represent a state of a valid tour since a factorial base set is related to the tour order permutations. Thus, one represents the integer in term of the factorial number base system by calculating the most significant numbers denoted by index(, , ,...).

$$x_{new} = \Sigma_n \ index_n \times n! \qquad (6)$$

sequentially for all n beginning with N-1 downto 0. For example, if five cities denoted by #1, #2, #3, #4, #5, and x_{old}= 0 = (#1,#2,#3,#4,#5) indicates the tour order that city #1 is visited first, etc., then one finds

x_{new}= 15 = 0x0! + 1x1! + 1x2! + 2 x 3! + 0x4! = (#1,#4,#3,#5,#2)

The representation index=(0, 1, 1, 2, 0) is obtained with respect to the base set(0! ,1! , 2! , 3!, 4!) sequentially decoded from the most significant bit first: (1) At step 1, the city at (1+0)-th position is the city #1 that is pick up to move 0 (meaning no) step to the left, and the rest is (in this case is not) push down, which is identical to: (#1,#2,#3,#4,#5); (2) At step 2, the city at the (2+2)-th position is read off by adding 2+2 to be the present city#4 that is pick up and move to the left by 2 positions, and the rest, city #2 and city#3, are push down, resulted in (#1,#4,#2,#3,#5); (3) At step 3, the city at the present (3+1)-th position is now the city#3 that is pick up and move 1 position to the left, and the rest, city #2, is push down, resulted in (#1,#4,#3,#2,#5); (4) At step 4, the city at the present (4+1)-th position is now the city #5 that is pick up and move 1 position to the left, and the rest, city#2, is now push down yielding finally (#1,#4,#3,#5,#2). [cf.Appendix A for 24 possible clockwise and counter-clockwise tours for 5 citis:5x4x3x2/5=24]

3. Applications: The FSA can reduce the running time from 5 hours to ten minutes for 10 cities on Mac II.

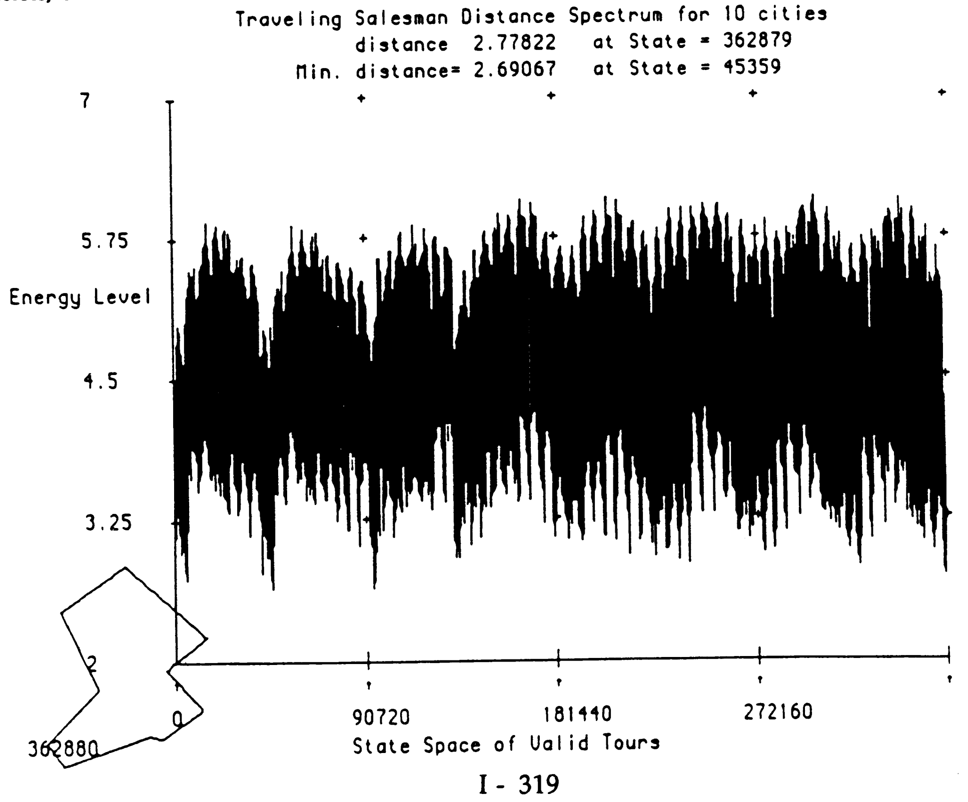
3.1 Exhaustive Search Technique: Exhaustive search techniques are useful, before the computational time limit, benchmarks calibrating the performances of other heuristic techniques. Results from 4 cities upto 10 cities are given in Figs.1. Divide n! by n possible home city, we take clockwise tours different from the counter clockwise tours for the sake of easy book keeping in the FSA.

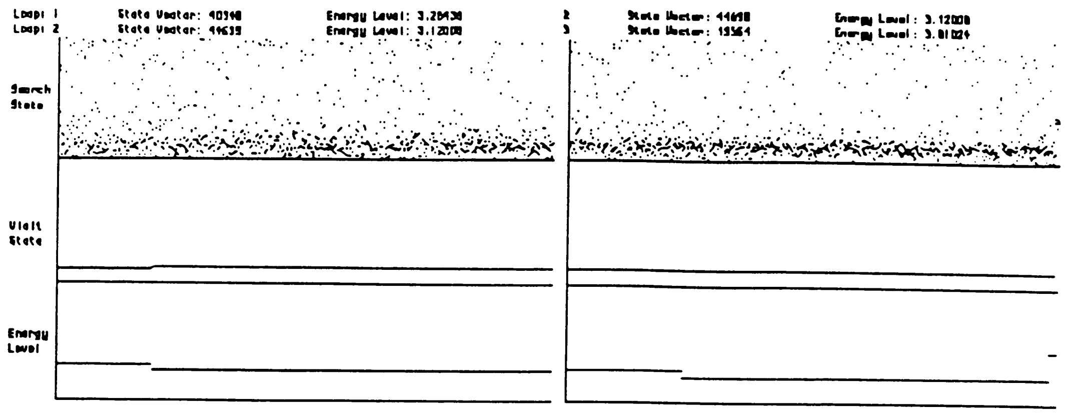
3.2 Fast Simulated Annealing Search Technique: Since Cauchy Machine is based on the local force field,u_i rather the total energy, then the decoupled processors can be much faster than the serial simulation. Fig.2 has the search, the accept, and the energy plotted against the time steps giving correct answers.

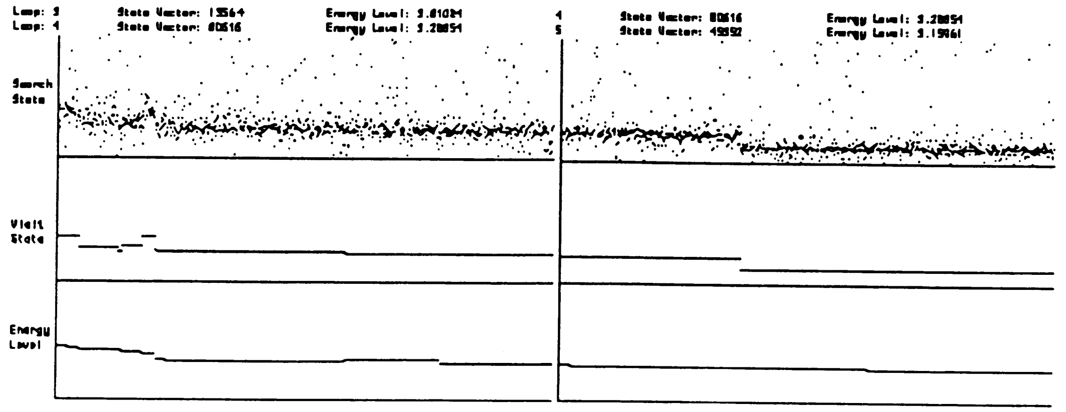
Appendix A Table for all 5 city 24 tours obtained by the coding scheme Eq(6)

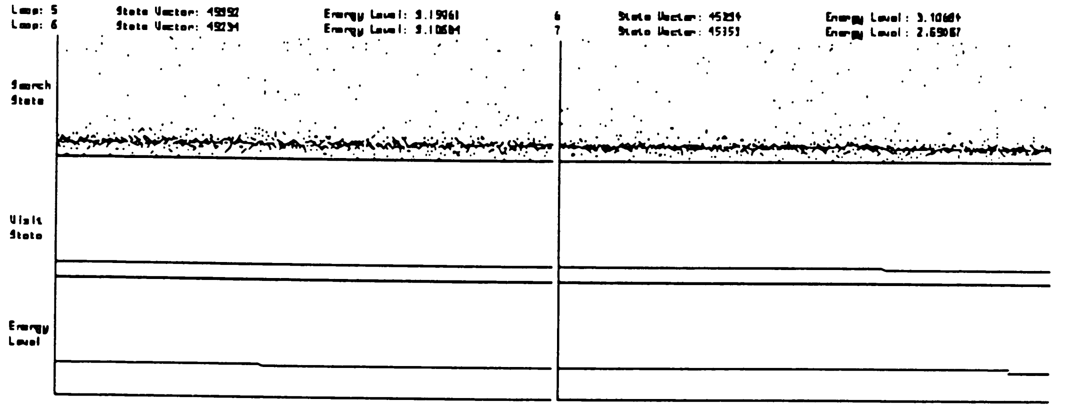
State (x)	#city tour (orders)	factorial-base (representation)	State (x)	#city tour (orders)	factorial-base (representation)
0	(#1,#2,#3,#4,#5),	(0,0,0,0,0)	12	(#1,#4,#2,#3,#5),	(0,0,0,2,0)
1	(#1,#2,#3,#5,#4),	(0,1,0,0,0)	13	(#1,#4,#2,#5,#3),	(0,1,0,2,0)
2	(#1,#2,#4,#3,#5),	(0,0,1,0,0)	14	(#1,#4,#3,#2,#5),	(0,0,1,2,0)
3	(#1,#2,#4,#5,#3),	(0,1,1,0,0)	15	(#1,#4,#3,#5,#2),	(0,1,1,2,0)
4	(#1,#2,#5,#3,#4),	(0,0,2,0,0)	16	(#1,#4,#5,#2,#3),	(0,0,2,2,0)
5	(#1,#2,#5,#4,#3),	(0,1,2,0,0)	17	(#1,#4,#5,#3,#2),	(0,1,2,2,0)
6	(#1,#3,#2,#4,#5),	(0,0,0,1,0)	18	(#1,#5,#2,#3,#4),	(0,0,0,3,0)
7	(#1,#3,#2,#5,#4),	(0,1,0,1,0)	19	(#1,#5,#2,#4,#3),	(0,1,0,3,0)
8	(#1,#3,#4,#2,#5),	(0,0,1,1,0)	20	(#1,#5,#3,#2,#4),	(0,0,1,3,0)
9	(#1,#3,#4,#5,#2)	(0,1,1,1,0)	21	(#1,#5,#3,#4,#2),	(0,1,1,3,0)
10	(#1,#3,#5,#2,#4),	(0,0,2,1,0)	22	(#1,#5,#4,#2,#3),	(0,0,2,3,0)
11	(#1,#3,#5,#4,#2),	(0,1,2,1,0)	23	(#1,#5,#4,#3,#2),	(0,1,2,3,0)

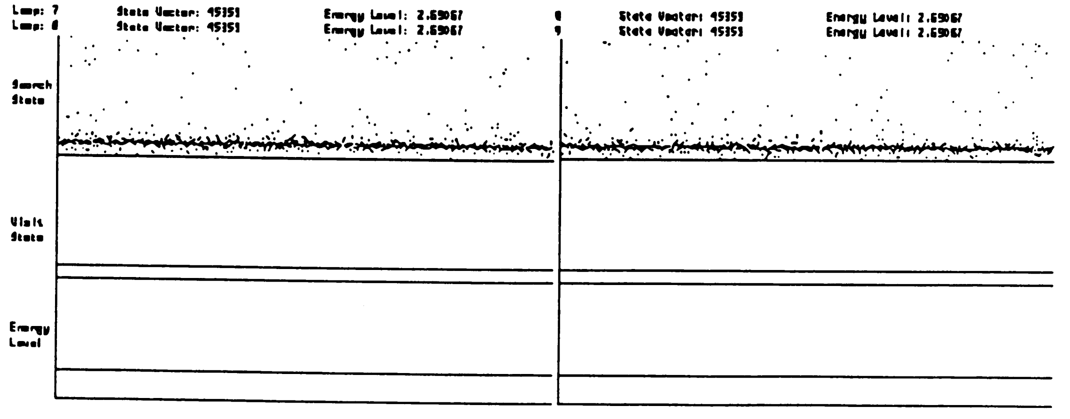
[1] N. Metropolis, A.W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, J. Chem. Phys. Vol.21, pp.1087-1092, June 1953.
[2] S. Kirkpatrick, C. Gelatt Jr, M.P. Vecchi, "Optimization by Simulated Annealing," Sci., 220, 4598, May 1983.
[[3] G. E. Hinton, T. J. Sejnowski, In:PDP Ed by Rummelhart,McClelland MIT Press, 1986, Ch. 7,pp.282-317
[4] S. Geman, D. Geman,, IEEE Trans. on Patt. Anal. Mach. Int.,vol. PAMI-6, pp. 721-741,Nov. 1984.
[5] H. Szu, "Fast Simulated Annealing," Neural network Conf. Proc.AIP, Vol. 151, pp.420-425,, Snowbird UT, Ed.Denker, 1986.
[6] H.Szu, "Nonconvex Optimization," SPIE Vol. 698, pp.59-65, 1987
[7] H. Szu,R. Hartley, "Fast Simulated Annealing," Phys. lett. A, Vol.122, pp.157-162, June 1987.
[8] H. Szu, R. Hartley, "Nonconvex Optimization by Fast Simulated Annealing," Proc. IEEE, Vol. 75,, pp.1538-1540, Nov. 1987.
[9] K. Scheff, H. Szu, "1-D Optical Cauchy Machine Infinite Film Spectrum Search," ICNN-87 , p. III-673,San Diego 1987
[10] Y. Takefuji, H. Szu, " Parallel Distributed Cauchy Machine," IJCNN-89, p.I-529, D.C. , June 18-22, 1989
[11] J.J. Hopfield, D. W. Tank, "Neural Computation of Decisions in Optimization Problems," Bio.Cyb., 52, pp. 141-152, 1985.
[12] D. W. Tank, J.J. Hopfield, IEEE Trans.Circ. Sys., Vol. CAS-33, No.5, pp.533-541, May 1986.
[13] S. Foo, H. Szu, " Solving Large Scale TSP by divide-and-conquer, " IJCNN-89, p. I-507, D.C., June 18-22, 1989
[14] H. Szu, "Reconfigurable Neural Nets ...," IJCNN-89, pp. I-485-496, Washington D.C. , June 18-22, 1989
[15] H. Szu, S. Foo,"Space-Scanning Curves for Spatiotemporal Representations, ..." IJCNN-90, Wash. D.C., Jan. 15-18,1990.
[16] H. Szu, K.Scheff,"Simulated Annealing Feature Extraction from Occluded and Cluttered Objects,"IJCNN-90, Wash.




Loop: 1 State Vector: 40348 Energy Level: 3.20430
Loop: 2 State Vector: 44635 Energy Level: 3.12000
2 State Vector: 44698 Energy Level: 3.12000
3 State Vector: 15564 Energy Level: 3.01024
Search State
Visit State
Energy Level


Loop: 3 State Vector: 15564 Energy Level: 3.81024
Loop: 4 State Vector: 80616 Energy Level: 3.28854
4 State Vector: 80616 Energy Level: 3.28854
5 State Vector: 49892 Energy Level: 3.19861
Search State
Visit State
Energy Level


Loop: 5 State Vector: 49892 Energy Level: 3.19861
Loop: 6 State Vector: 49234 Energy Level: 3.10684
6 State Vector: 45234 Energy Level: 3.10684
7 State Vector: 45351 Energy Level: 2.69067
Search State
Visit State
Energy Level


Loop: 7 State Vector: 45351 Energy Level: 2.69067
Loop: 8 State Vector: 45351 Energy Level: 2.69067
8 State Vector: 45351 Energy Level: 2.69067
9 State Vector: 45351 Energy Level: 2.69067
Search State
Visit State
Energy Level

Nonlinear dynamics of analog associative memory neural networks

F. R. Waugh, C. M. Marcus, and R. M. Westervelt
Division of Applied Sciences and Department of Physics
Harvard University, Cambridge, Massachusetts 02138

Parallel dynamics is essential for fast computation in large artificial neural networks. Networks of two-state neurons, often studied because of their formal resemblance to Ising systems, must be updated serially to prevent oscillation and as a result perform computational tasks slowly. In contrast, networks of analog neurons with a smooth graded response can be updated in parallel with guaranteed convergence to a fixed point.

In this paper, we analyze the dynamics of discrete-time analog associative memory neural networks updated in parallel. Using global stability analysis, we derive phase diagrams describing the network dynamics as a function of the fraction α of stored memories and of the gain or maximum slope β of the neuron transfer function. We show that, for a range of values of α and β, such networks can function as reliable and stable associative memories. The results also suggest the possibility of "annealing" a network deterministically by varying the gain so as to decrease the likelihood that the system becomes trapped in undesirable local minima. Numerical investigations confirm both the phase diagrams and the analog annealing concept. Details of the results presented here will appear in a subsequent publication.[1]

The network we study is the iterated map described by the N coupled nonlinear equations[2]

$$x_i(t+1) = F_i\left(\sum_{j=1}^{N} T_{ij} x_j(t)\right), \quad i = 1,\ldots,N \tag{1}$$

where $x_i(t)$ denotes the output of neuron i at time step t and F_i is the neuron transfer function, which may be different for each i. In Ref. 2 we proved two general properties of the dynamical system (1). First, if T_{ij} is symmetric and if all F_i are single-valued, monotonically increasing functions rising less rapidly than linearly for large arguments, then all attractors of (1) are either fixed points or period-two limit cycles. Second, if the system obeys the condition $\beta_i < |1/\lambda_{\min}|$ for all i, where $\beta_i > 0$ is the maximum slope of F_i and $\lambda_{\min}$ the most negative eigenvalue of T_{ij}, then all period-two limit cycles are eliminated and convergence to a fixed point attractor is guaranteed.

In this paper, we apply these results to two well-studied models of associative memory, the

Hebb rule[3] and the pseudoinverse rule.[4, 5] For the Hebb rule, the connection matrix is given by

$$T_{ij} = \frac{1}{N}\sum_{\mu=1}^{\alpha N} \xi_i^\mu \xi_j^\mu \ , \ i \neq j; \ T_{ii} = 0, \tag{2}$$

while for the pseudoinverse rule, it is given by

$$T_{ij} = \frac{1}{N}\sum_{\mu,\nu=1}^{\alpha N} \xi_i^\mu \left(C^{-1}\right)_{\mu\nu} \xi_j^\nu \ , \ i \neq j; \ T_{ii} = 0 \tag{3a}$$

$$C_{\mu\nu} = \frac{1}{N}\sum_{i=1}^{N} \xi_i^\mu \xi_i^\nu \ . \tag{3b}$$

In Eqs. (2) and (3), the ξ_i^μ are random, unbiased binary variables ($\xi_i^\mu \in \{-1,+1\}$ for all i and μ). We choose $F_i(x) = \tanh(\beta x)$ for all i and consider the dynamics of (1) for large N. Using the results of Ref. 2 and the eigenvalue spectra of the matrices (2) and (3a),[5, 6] we find that period-two limit cycles are eliminated from the system (1) for both learning rules when

$$\beta < 1/\alpha \tag{4}$$

and that the origin is the only attractor of the system for the Hebb rule when

$$\beta < 1/\left(1 + 2\sqrt{\alpha}\right) \tag{5}$$

and for the pseudoinverse rule when

$$\beta < 1/(1-\alpha). \tag{6}$$

We also derive the maximum storage capacity of the system as a function of β for both learning rules by examining the dynamical stability of the memory recall states. For the particular choice of neuron transfer function given above, the maximum storage capacity agrees with the results from the thermodynamical treatment at temperature $1/\beta$.[7]

We use these results to compute the phase diagrams shown in Figs. 1(a) and 1(b). Figure 1(a) shows the phase diagram for the system (1) in the limit of large N when $F_i(x) = \tanh(\beta x)$ and T_{ij} is given by the Hebb rule (2). In the region marked "origin," the system possesses a single fixed point attractor at its origin ($x_i = 0$ for all i). The boundary delineating this region arises from the condition (5). In the region marked "sg," the system converges to fixed points with negligible overlap with the stored memories ("spin glass states"). In the region marked "recall," fixed points appear which have macroscopic overlap with a single stored memory ("recall states"). Because the basins of attraction of the recall states are large, the system works reliably as an associative memory in this region. The boundary separating the spin glass and recall regions is the maximum storage capacity (see above). Finally, in the region marked "osc," period-two limit cycles ("oscillatory states") appear along with recall and spin glass states. The boundary delineating this region arises from the condition (4).

Figure 1(b) shows the analogous phase diagram for the system (1) with the pseudoinverse matrix (3a). The regions marked "origin," "recall," and "osc" have the same meaning as in Fig. 1(a). Their boundaries are given by conditions (4) and (6) and by the maximum storage capacity; the boundaries arising from condition (6) and from the maximum storage capacity are coincident. The system does not exhibit a spin glass region.

Figures 3(a) and 3(b) show typical results of numerical investigations of networks of 100 neurons. Each figure represents an average over 20 matrices T_{ij}. For each matrix, β was varied from $\beta \cong 0.3$ to $\beta \cong 90.0$. For each value of β, the system (1) was started at 50 random points ($x_i(0) \in \{-1,+1\}$ for all i) and iterated until convergence to a fixed point or period-two limit cycle. The vertical axes of Figs. 3(a) and 3(b) represent the fraction of iterations which converged to the origin, recall states, spin glass states, and oscillatory states; the bars across the top of the figures indicate the corresponding regions of the phase diagrams. For both learning rules, the data agree well with the phase diagrams. The relatively low number of spin glass states in the recall region at low values of β, furthermore, suggests the possibility of annealing the network deterministically by starting β at the lower boundary of the recall region and increasing it to the upper boundary while the network iterates.[8]

One of us (F.R.W.) acknowledges support by the U. S. Army Research Office as a JSEP Graduate Fellow. This work was supported by the Joint Services Electronics Program, contract N00014-89-J-1023, and by ONR contract N00014-89-J-1592.

References

1. C. M. Marcus, F. R. Waugh, and R. M. Westervelt, to be published.
2. C. M. Marcus and R. M. Westervelt, Phys. Rev. A **40**, 501 (1989).
3. D. O. Hebb, *The Organization of Human Behavior* (Wiley, New York, 1949).
4. L. Personnaz, I. Guyon, and G. Dreyfus, J. Physique Lett. **46**, L359 (1985).
5. I. Kanter and H. Sompolinsky, Phys. Rev. A **35**, 380 (1987).
6. A. Crisanti and H. Sompolinsky, Phys. Rev. A **36**, 4922 (1987).
7. D. J. Amit, H. Gutfreund, and H. Sompolinsky, Ann. Phys. **173**, 30 (1987).
8. F. R. Waugh, C. M. Marcus, and R. M. Westervelt, to be published.

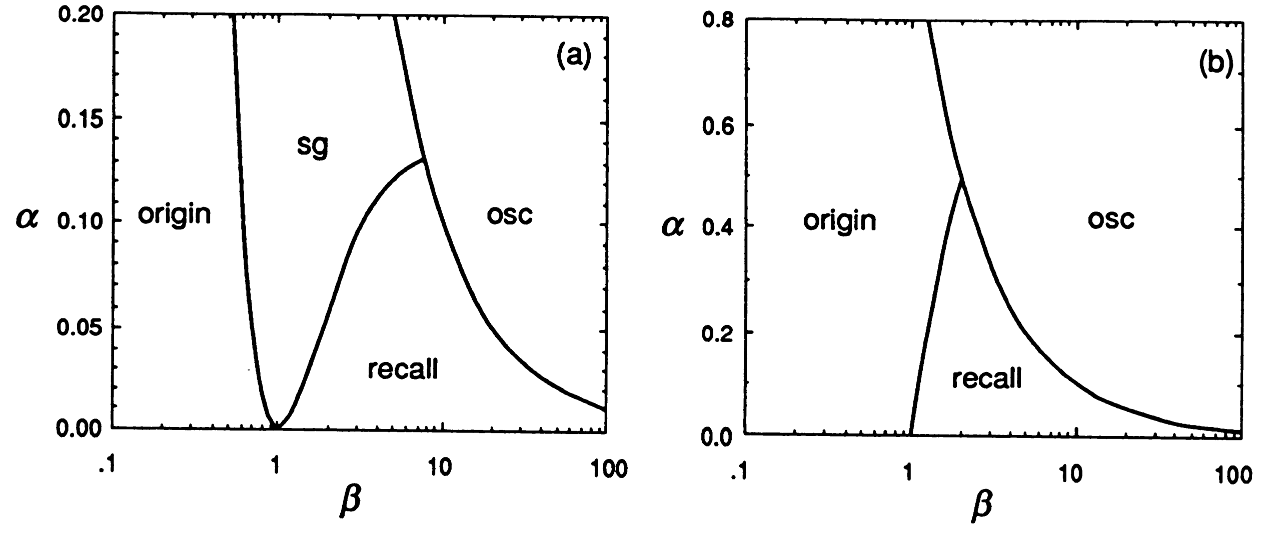


Figure 1. Phase diagrams (a) for the Hebb rule and (b) for the pseudoinverse rule. See text for details.

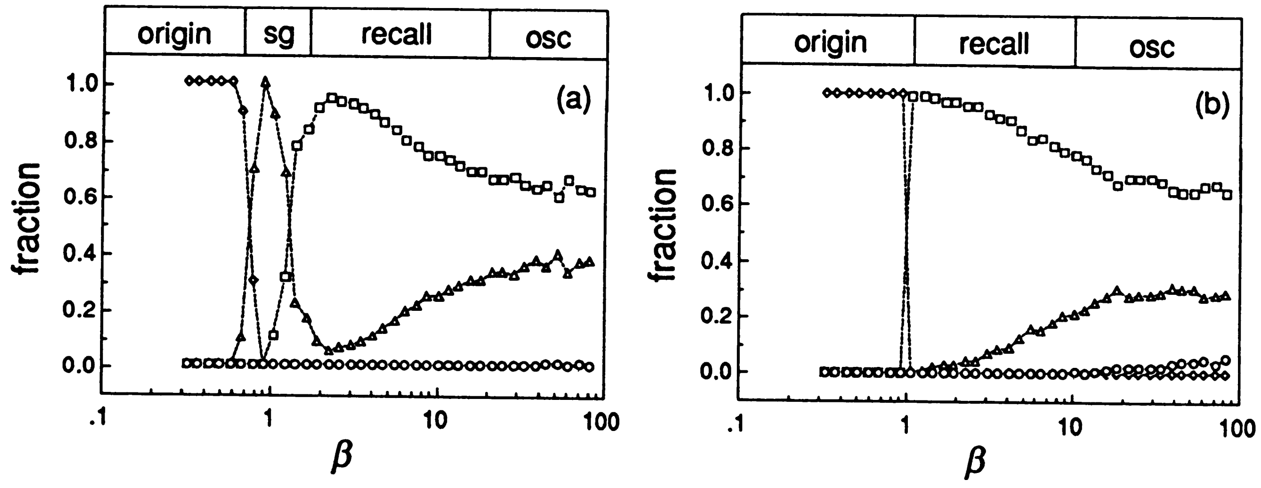


Figure 2. Results of numerical investigations of the system (1) (a) for the Hebb rule with α=0.05, and (b) for the pseudoinverse rule with α=0.10. Diamonds indicate the origin, squares the recall states, triangles the spin glass states, and circles the oscillatory states. See text for details.

MODELING OF FAULT-TOLERANCE IN NEURAL NETWORKS

Lee A. Belfore, II Barry W. Johnson James H. Aylor

University of Virginia
Center for Semicustom Integrated Systems
Department of Electrical Engineering
Charlottesville, Virginia 22901
Phone: (804) 924-7623

Abstract -- Neural networks are finding a wide range of engineering applications in areas such as adaptive control, machine vision, and pattern recognition. In order to use neural networks in applications requiring high reliability, a method for fault tolerance analysis must be developed to allow the prediction of a network's performance in the presense of faulty elements. This paper presents an overview of an analytic technique for assessing the fault tolerance of neural networks. The basis of the technique is developed through an analogy with magnetic spin systems using statistical mechanics. A Markov model is created using the statistical mechanics analogy, and the results are compared with simulations. The primary example presented in this paper is an autoassociative memory.

Key Words -- fault tolerant computing, fault tolerance analysis, neural networks, redundancy, VLSI

This work was supported by the Virginia Center for Innovative Technology Development Center grant to the Center for Semicustom Integrated Systems.

1. Introduction

As neural networks emerge as viable solutions to many engineering problems, some shortcomings exist in their engineering analysis. Among the many engineering design goals is fault tolerance. Without some analytical technique for accurately assessing fault tolerance, the usefulness of the neural network in certain high reliability applications is questionable, even though the neural network may actually have the appropriate fault tolerance properties. This paper provides an overview of results using an analytic technique for the assessment of the performance of faulty neural networks which was developed in [Belf89].

Many researchers address the fault tolerance of neural networks in a qualitative fashion, however little work has been done to analytically assess the performance of neural network in the presence of faults [Jack86, Fuku75, Litt75, Pere86a, Hopf82, Cart88]. The networks appear to be fault tolerant, however previous research either assumed system learning will provide fault tolerance or simply stated the network to be fault tolerant without any further analysis. The research presented in this paper has taken the position that it is important to be able to assess the performance of the neural network in the absence of learning since it may not be possible for relearning to occur in the network during the time following a fault.

2. Overview of the Analytical Technique

A statistical mechanics analogy can be made between the Ising magnetic spin system and a model for a neural network [Litt74, Shaw74, Litt75, Litt78, Pere84, Pere86a, Pere86b]. The magnetic spins are analogous to the state of the neuron, either firing or not firing, and the magnetic interactions are analogous to the interconnections between neurons. Each neuron will have a firing transition probability defined by

$$p^i(\boxed{\blacksquare} \mid \mathrm{V}^I) = \frac{1}{1+\exp\left[-\frac{(\sum_{j=0}^{N} w_{ij} V_j^I) - \theta_i}{T}\right]} \tag{1}$$

where V^I is the initial state from which transition probabilities are computed, V_j^I is the initial state of neuron j, w_{ij} is the interconnection weight from neuron j to neuron i, $\boxed{\blacksquare}$ indicates the neuron is to fire, θ_i is the threshold for neuron i, and T is the parameter adjusting the randomness of the transition.

The fault polynomial representation developed in [Belf89] and summarized here allows one to describe the behavior of the neural network in terms of state transition probabilities in the presense of faults. The fault polynomial representation includes all outcomes of possible faulty behavior and averages their effect with respect to a degree of faultiness. Faults in this model are represented by the absence of weight connections between neurons.

The smallest building block is each neuron in the network. A fault polynomial representation for each neuron is generated by considering the probability of the neuron transitioning to either firing or nonfiring states. This representation is generated by looking at each possible degree of faultiness for the neuron. Each neuron has k^i inputs and can have $1, \dots, f, \dots, k^i$ inputs faulted, and the coefficients for the fault polynomial are generated by taking a cumulative sum of the probabilities of the neuron transitioning to a firing. The divisor polynomial coefficients are, therefore, $D^f = \begin{bmatrix} k^i \\ f \end{bmatrix}$, where $\begin{bmatrix} k^i \\ f \end{bmatrix} = \frac{k^i!}{f!(k^i-f)!}$ which is simply, the number of possible ways that neuron i can have f faulted weight connections. The nonfiring transition probabilities for a neuron are computed similarly.

The degree of each term in the fault polynomial representation corresponds to the number of weight faults inserted at the input to each neuron. Suppose there are f faults inserted, and we want to know the *average* behavior of the neuron in the presence of f faults. We need to average together all possible cases of there being f faults. If the neuron has k^i inputs, there are $\begin{bmatrix} k^i \\ f \end{bmatrix}$ cases of the neuron having f faults, where $\begin{bmatrix} k^i \\ f \end{bmatrix}$ is the number of combinations of f inputs chosen faulty out of k^i possible inputs. Therefore, taking an average of $\begin{bmatrix} k^i \\ f \end{bmatrix}$ neuron transition probabilities is necessary. Since we are taking the cumulative total of the neuron transition probabilities for there being f faults, we can extract the neuron transition probability if we also keep track of the number of outcomes of there being f faults. The coefficients of $\mathrm{C}(r)$ are computed by summing over all outcomes of neuron i having f faults. Specifically,

$$C^{i\,f}_{s\blacksquare} = \sum_{all\,\zeta} p^i(\blacksquare \mid f, \mathrm{V}^I, \zeta) \quad \zeta \in \{neuron\ i\ with\ f\ faults\} \tag{2}$$

where $p^i(\blacksquare \mid f, \mathrm{V}^I, \zeta)$ is the probability of neuron i firing given a particular trial ζ of a level of faultiness f from starting state V^I. The cumulative polynomial for this case would take the form

$$\mathrm{C}^i_{s\blacksquare}(r) = \sum_{f=0}^{k^i} C^{i\,f}_{s\blacksquare} r^f . \tag{3}$$

The coefficients of the divisor polynomial can be computed, knowing the number of inputs to each neuron i, as

$$D^{i\,f} = \begin{bmatrix} k^i \\ f \end{bmatrix}, \tag{4}$$

Thus, the divisor polynomial is simply

$$\mathrm{D}(r) = D^{i\,f} r^f . \tag{5a}$$

$$= \sum_{f=0}^{k^i} \begin{bmatrix} k^i \\ f \end{bmatrix} r^f . \tag{5b}$$

The average probability of neuron i firing given a starting state V^I is given by

$$p^i(\blacksquare \mid f, \mathrm{V}^I) = \frac{C^{i\,f}_{s\blacksquare}}{D^f} . \tag{6}$$

Thus, to get the neuron firing probability for all degrees of faultiness f, the cumulative polynomial is divided, term by term, by the divisor polynomial. The fault polynomial representation for a neuron not firing is developed similarly by considering nonfiring probabilities instead of firing probabilities in the above development.

If the individual neuron transition probabilities are considered independent, the transition probabilities to the neural system states are composed of the product of transition probabilities over all neurons to their respective states. It can be shown that the fault polynomial representation for the network is determined by a product of the fault polynomial representations of the neurons [Belf89]. The assumption of independence for the neuron transition probabilities is justified by considering that the neuron makes its transition based only on the input at a particular time and the statistics of that particular neuron. In order to generate the transition probabilities from one state to another state and more importantly the Markov transition matricies, the fault polynomial representations for each state transition are computed. Thus, the average faulty behavior for the system is captured for specific state transition probabilities.

3. Analysis of the Fault Tolerance of a Neural Network Implementation

An autoassociative memory was chosen for analysis because of the simple storage algorithm and regular structure that makes it straight forward to analyze. The storage algorithm used in this analysis is a modified form of [Hopf82] and is as follows

$$w_{ij} = \sum_{s=1}^{M} S_i^s S_j^s \quad \textit{for } i,j=1 ,\dots, N \tag{7}$$

where S_i^s is the i^{th} bit of the s^{th} pattern being stored in the memory and w_{ij} is the interconnection weight from the output of neuron i to the input of neuron j. The patterns stored in this network are S^1=⊡⊡⊡⊡□□□□ and S^2=⊡⊡□□⊡⊡□□. The fault polynomial representation was applied to the autoassociative memory with these patterns stored, and analytical results are compared with simulated results.

Applying the fault polynomial representation and approach to the autoassociative memory is a straight forward process. First, the performance has to be defined. For the results presented here, the performance is measured by

$$\Gamma(f,h) = p(S^s \mid V^I \in \{\textit{states Hamming distance } h \textit{ from } S^s\}, f), \tag{8}$$

which is the probability of resulting in one of the stored patterns S^s given an initial state V^I that is a Hamming distance h from the stored pattern S^s and f faults. For the 8 neuron neural system, Figure 1 shows the performance plotted, as the solid curves, for different Hamming distances from either S^1 or S^2. The horizontal axis represents the level of faultiness in the neural system measured by the number of weight connections removed. The vertical axis represents the performance of the neural system measured by the probability of resulting in S^s. Figure 1 has four plots corresponding to Hamming distances of 0, 1, 2, and 3 from one of the stored patterns. A temperature of T=1.0 was chosen to show the behavior at a temperature that is large enough to capture the random behavior yet small enough so that the system functions intuitively. Figure 1 has a number of interesting features, foremost is the smooth degradation of the performance as the number of faults is increased.

The simulations performed used neurons whose form is identical to that used in the analysis performed. This approach has two motivations. The first is an informal verification of the analytical approach. If all the neuron parameters are identical, the simulation results should agree favorably. The second is a demonstration of the fault tolerance of the autoassociative memory.

The parameters used in the simulations are summarized in Table 1.

parameter	value
temperature	T=1.0
transition relation	Boltzmann pdf with neurons having outputs of 1 and -1
iterations	5
initial states	for each hamming relation, bits chosen in a uniform random fashion
faults	for fault level, faults chosen in a uniform random fashion
simulations per data point	512

TABLE 1. Boltzmann Simulation Parameters

Figure 1 shows the results of the Boltzmann simulations as the data points superimposed on the analytical results, and as expected, the analytical method accurately predicts the simulated results.

4. Summary

In summary, this paper presents the analysis of the fault tolerance of neural networks using a statistical mechanics model. Simulated results are compared with analytical results showing that the analytical model does indeed conform to the simulation model.

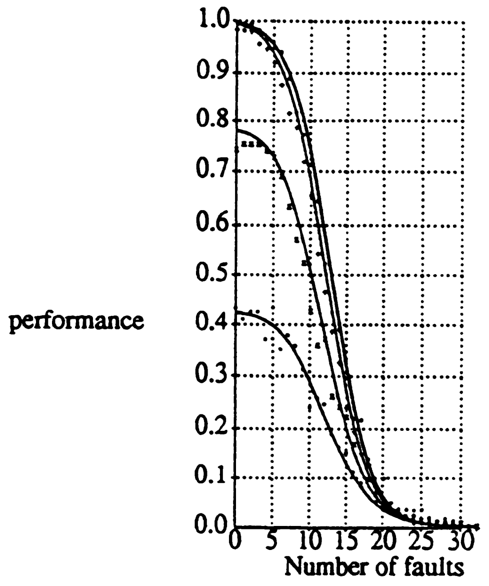


Figure 1. Analytical and Simulated Results using Boltzmann Neurons

5. References

[Belf89] Lee A. Belfore, II, *Modeling of Fault Tolerance in Neural Networks*, Ph. D. Dissertation, August, 1989.

[Fuku75] Kunihiko Fukushima, "Cognitron: A Self-organizing Multilayered Neural Network," *Biological Cybernetics*, vol. 20, 1975, pp. 121-136.

[Hopf82] J.J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities", *Proceedings of the National Academy of Sciences USA*, Vol. 79, April 1982, pp. 2554-2558.

[Jack86] L. D. Jackel, R. E. Howard, H. P. Graf, B. Straughn, J. S. Denker, "Artificial neural networks for computing," *J. Vac. Sci. Technol. B.*, vol. 4, no. 1, Jan/Feb 1986, pp. 61-63.

[Litt74] W. A. Little, "The Existence of Persistent States in the Brain," *Mathematical Biosciences*, vol. 19, 1974, pp. 101-120.

[Litt75] W. A. Little and Gordon L. Shaw, "A Statistical Theory of Short and Long Term Memory," *Behavioral Biology*, vol. 14, 1975, pp. 115-133.

[Litt78] W. A. Little and Gordon L. Shaw, "Analytic Study of the Memory Storage Capacity of a Neural Network," *Mathematical Biosciences*, vol. 39, 1978, pp. 281-190.

[Pere84] P. Peretto, "Collective Properties of Neural Networks: A Statistical Physicas Approach," *Biological Cybernetics*, vol. 50, 1984, pp. 51-62.

[Pere86a] P. Peretto and J. J. Niez, "Long Term Memory Storage Capacity of Multiconnected Neural Networks," *Biological Cybernetics*, vol. 54, 1986, pp. 53-63.

[Pere86b] P. Peretto and J. J. Niez, "Stochastic Dynamics of Neural Networks," *IEEE Transactions on Systems, Man And Cybernetics*, vol. SMC-16, no. 5, September/October 1986, pp. 73-83.

[Shaw74] Gordon L. Shaw and R. Vasudevan, "Persistent States of Neural Networks and the Random Nature of Synaptic Transmission," *Mathematical Biosciences*, vol. 21, 1974, pp. 207-218

NEURAL NETWORKS WITH PERIODIC OUTPUTS :APPLICATION TO THE RECOGNITION OF TEMPORAL SEQUENCES OF PATTERN

P.Bourret *+

*Department of Computer sciences
UNIVERSITY OF MARYLAND
COLLEGE PARK
MD 20742
USA

+ ONERA-CERT/DERI
2 Avenue E.Belin
BP 4025
31055 Toulouse CEDEX FRANCE

I Introduction

In the past few last years the interest in learning and recognition of temporal sequences of patterns has increased.[PINE87],[DEHA87],[DOYA89],[UCHI89].Hopf bifurcations have been use to achieve limit cycle attractors [BAIR89];some other have tried to process temporal patterns by means of "leaky integrator", which are neurons which have continuous outputs.But, in each case the exhibited results are not very efficient.The aim of our paper is to give a framework for temporal sequences recognition by examining several interesting properties that we have experimentally shown on a very small network which consists of only five nodes.After the presentation of this network and its properties we show that using an assembly of "five nodes networks" (5NN) and given a set of possible inputs, we are able to recognize each of these inputs.Moreover we also show that another set of 5NN enables us to determine what the previous input was using a recursive method.

II The "5NN" architecture and its activation rule

The architecture is very simple and is shown in figure I

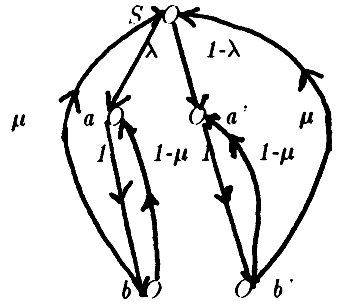


FIGURE I

The activation rule is a slight generalization of the competitive activation mechanism [REGG87].
Let $a_i(t)$ be the activity level of node U_i at time t and w_{ij} the strength of the link between U_i and U_j .($a_i \in [0,1]$).The activation rule is based on the following differential equation:

$$\frac{da_i}{dt} = ((\sum_j \frac{w_{ij}\, a_j(t)}{\sum_k w_{kj}\, a_j(t)} - 1)a_i(t) + \theta_i \sin \omega t)(1 - a_i(t))[1]$$

With the constraint: $\sum_i \theta_i = 0$ which implies that $\sum_i Log(1 - a_i) = C'_0$ [2] for some constant C'_0

Let $\nu = \frac{(1-\mu)}{\mu}$ and $\rho = \frac{(1-\lambda)}{\lambda}$

Because of constraint [2] b and b' must not equal 1.0.The equations, of which the solutions are the coordinates of the equilibrium points of a 5NN, can be written as follows

$$\frac{db}{dt} = a - b + \theta_b = 0[3]$$

$$\frac{db'}{dt} = a' - b' + \theta_{b'} = 0[4]$$

$$\frac{ds}{dt}=(\frac{a}{s+\nu a}+\frac{a'}{s+\nu a'}-1)s(1-s)+(\frac{\theta_b s}{s+\nu a}+\frac{\theta_{b'} s}{s+\nu a'}+\theta_e)(1-s)[5]$$

$$\frac{da}{dt}=(\frac{\nu a}{s+\nu a}+\frac{s}{a+\rho a'}-1)a(1-a)+(\frac{\nu a\,\theta_b}{s+\nu a}+\theta_a)(1-a)[6]$$

$$\frac{da'}{dt}=(\frac{\nu a'}{s+\nu a'}+\frac{\rho s}{a+\rho a'}-1)a'(1-a')+(\frac{\nu a'\ \theta_{b'}}{s+\nu a'}+\theta_{a'})(1-a')[7]$$

Given $\theta_b, \theta_{b'}$ we can easily compute $\theta_a, \theta_{a'}, \theta_e$ in such a way that there is an equilibrium point out of the planes $\theta_a=1. \theta_b=1, \theta_{a'}=1, \theta_{b'}=1, \theta_e=1$.For this purpose we can solve the two folowing systems,in order to get a particular solution.

System I

$$\frac{a}{s+\nu a}+\frac{a'}{s+\nu a'}-1=0[8]$$

$$\frac{\nu a}{s+\nu a}+\frac{s}{a+\rho a'}-1=0[9]$$

$$\frac{\nu a'}{s+\nu a'}+\frac{\rho s}{a+\rho a'}-1=0[10]$$

which gives the values of s,a,a' which are used to solve the system II.

System II

$$\frac{\theta_b s}{s+\nu a}+\frac{\theta_{b'} s}{s+\nu a'}+\theta_e=0[11]$$

$$\frac{\nu a\,\theta_b}{s+\nu a}+\theta_a=0[12]$$

$$\frac{\nu a'\ \theta_{b'}}{s+\nu a}+\theta_{a'}=0[13]$$

which gives $\theta_e, \theta_a, \theta_{a'}$.

When $\theta_e, \theta_a, \theta_{a'}$ are computed the differential equations which are defined in equations [3] through [7] are used to compute the next state of the "5NN".

IIIProperties of the "5NN"

An extensive experimental study of the "5NN" behaviour enables us to claim the three following properties:

1) Given the parameters (λ ,μ)of a "5NN" and its starting point s(0).a(0).a'(0).b(0).b'(0) , three of its nodes oscillate between two extreme values which are different for each node whereas the two other nodes remain constant.The sets of oscillating nodes are either s.a.b or s.a'.b'

2)the set of oscillating nodes is determined by the value of $r(0)=\frac{(\frac{da(0)}{dt})^2+(\frac{db(0)}{dt})^2}{(\frac{da'(0)}{dt})^2+(\frac{db'(0)}{dt})^2}$

If r(0) 1 then s.a.b oscillates. whereas s.a .b oscillates when r(0) 1.Moreover r(t)-1 has the same sign as r(0)-1 for the same inputs $\theta_b . \theta_{b'}$.

3)r(0)=1 is a manifold which split the hypercube of dimension 5 into two parts.The hypercube is defined by the activity value of each node.If the starting node is on one side of the manifold the points which represent the states of the "5NN" stay on this side until the inputs are changed.A more detailled report of the experimentally observed properties of the "5NN" can be found in [BOUR89]

IV Inputs sequence determination by an assembly of "5NN"

Given a finite set Θ of possible inputs $\{(\theta_b^1\theta_{b'}^1),....(\theta_b^k\theta_{b'}^k)\}$ it is possible to define p "5NN" such that if 2^p k there is only one possible input remaining such that the trajectory projection on the plane (a.b) of the "5NN" states is not reduced to a single point.Thus, with enough well chosen "5NN" it is possible to identify without a doubt what the input is.(see Figure II).

The search of the previous input is not so simple.It is based on the definition of a "5NN" which allows oscillations of a set of nodes for two and only two pairs of inputs among those which are allowed.First we can remark that the constraint [2] implies that s,a,b,a',b' cannot be too close to 1.0 because Log(1-s),Log(1-a),..would overflow.So,there is an ϵ ,computer dependant, such that s,a,b,a',b'$<$1-ϵ [14].Thus we can deduce that s,a,b,a',b'>l [15], where l stands for $1-e^{K_0}\eta^2$.With η such that a',b'>1-η
$K_0 = Log(1-a_0)+Log(1-b_0)+Log(1-a'_0)+Log(1-b'_0)+Log(1-s_0)$.

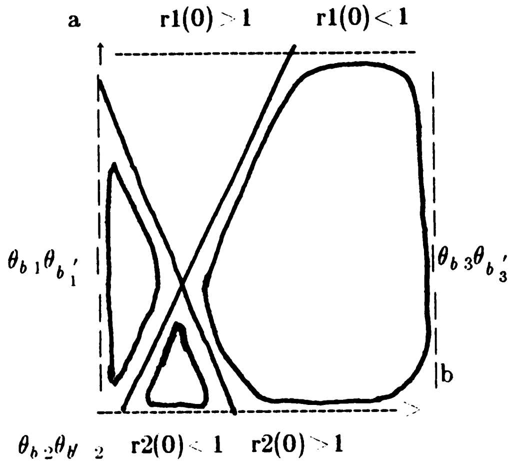

a
r1(0) > 1
r1(0) < 1
$\theta_{b1}\theta_{b'1}$
$\theta_{b3}\theta_{b'3}$
b
$\theta_{b2}\theta_{b'2}$
r2(0) < 1
r2(0) > 1

FIGURE II

Let $r_k^i(0)$ be the value of r(0) for a given "5NN" defined by $\nu_k\,\rho_k$ and an input $\theta_{b_i}\theta_{b'_i}$ .If we can prove that either $r_k^i(0) > 1$;$r_k^j(0) > 1$ and $r_k^l(0) < 1$ for all $l \neq$ i,j[16];or that $r_k^i(0)$ 1 and $r_k^j(0) < 1$ and $r_k^l(0) > 1$ for all $l \neq$ i,j[17] ;then the only the only possible input previously sent before the present one $\theta_{b_i}\theta_{b'_i}$ is $\theta_{b_j}\theta_{b'_j}$ .The search of solution space of inequalities [16],[17] is done in the following manner.

$$[16] \equiv \text{for all } l \neq i.j \; r_k^l(0) < 1 \equiv \frac{(\frac{da_k^l(0)}{dt})^2 + (\frac{db_k^l(0)}{dt})^2}{(\frac{da'^l_k(0)}{dt})^2 + (\frac{db'^l_k(0)}{dt})^2} < 1$$

$$\equiv \text{for all } l \neq i,j \; (\frac{da_k^l(t)}{dt})^2 + (\frac{db_k^l(t)}{dt})^2 < (\frac{da'^l_k(t)}{dt})^2 + (\frac{db'^l_k(t)}{dt})^2$$

Let U(f),L(f) be an upper (lower) bound of f.Then we have:

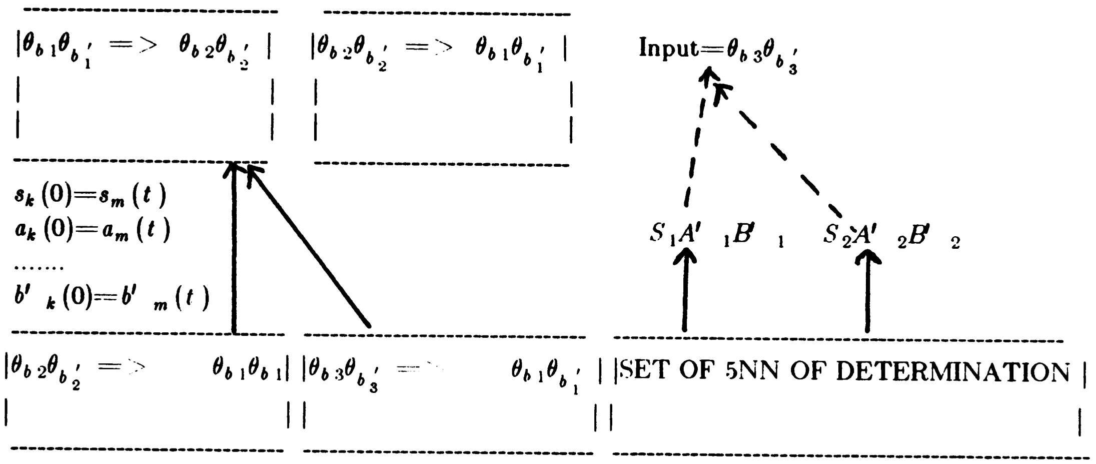
$$\text{for all } l \neq i,j \; L((\frac{da^l(t)}{dt})^2) + L((\frac{db^l(t)}{dt})^2) \;\; U((\frac{da'^l(t)}{dt})^2) + U((\frac{db'^l(t)}{dt})^2) - \quad [16]$$

$$\text{for all } l \neq i,j \; U((\frac{da^l(t)}{dt})^2) + U((\frac{db^l(t)}{dt})^2) < L((\frac{da'^l(t)}{dt})^2) + L((\frac{db'^l(t)}{dt})^2) = \quad [17]$$

The computation of $L(\frac{da^l}{dt})$or $U(\frac{da^l}{dt})$is calculated by determining the upper and lower bounds of s,a,a',b,b' given in [14] and[15].Then the derivatives with respect to ν and ρ of $L((\frac{da^l}{dt})^2) + L((\frac{db^l}{dt})^2) \cdot U((\frac{da'^l}{dt})^2) \; U((\frac{db'^l}{dt})^2)$are linear in $\frac{\nu}{\nu+1}$and $\frac{\rho}{\rho+1}$.Thus we can maximize (minimize) $r^l(0)$.In fact we only prove that if there has been a previous input then it could not be different from $(\theta_{b_j}, \theta_{b'_j})$.Therefore our assembly of 5NN can only be used when we know how many inputs have already be applied to the network.(see Figure III)

V Conclusion

It is not obvious how one can find a 5NN which, given a set of inputs will have its ratio r(0) 1 for only two given inputs.Actually what we have done is to define a set of 5NN and look for a set of inputs such that any pair of these inputs may only be recognized by one of the 5NN.A lot of others problems have arisen which cannot be detailed here,especially the determination of ϵ and l for a given computer and its software.At last the properties which have been presented in section III should be mathematically proved.We hope that some readers will be interested by the amazing properties of our 5NN and will help us to solve some of the remaining open problems, the most important of which is:Can a 5NN learn to have three oscillating nodes for two given inputs and a different set of oscillating nodes for the n-2 other inputs?



$\theta_{b1}\theta'_{b_1} => \theta_{b2}\theta'_{b_2}$
$\theta_{b2}\theta'_{b_2} => \theta_{b1}\theta'_{b_1}$
Input=$\theta_{b3}\theta'_{b_3}$
$s_k(0)=s_m(t)$
$a_k(0)=a_m(t)$
....
$b'_k(0)=b'_m(t)$
$S_1A'_1B'_1$
$S_2A'_2B'_2$
$\theta_{b2}\theta'_{b_2} => \theta_{b1}\theta_{b1}$
$\theta_{b3}\theta'_{b_3} => \theta_{b1}\theta'_{b_1}$
SET OF 5NN OF DETERMINATION

FIGURE III

References

[Bair89] B.Baird A Bifurcation Theory Approach to Vector Field Programming for Periodic Attractors.Proc IJCNN Conf.Washington June 89.

[Bour89] P.Bourret Neural Network with Periodic Output .Technical Report .CERT/DERI (to appear September 89)

[Doya89] K.Doya.S.Yoshizawa Memorizing Oscillatory Patterns in the Analog Neuron Network.Proc IJCNN Conf.Washington June 89

[Deha89] S.Dehaenne.J.P. Changeux.J.P. Nadal:Neural Networks that learn temporal squences by selection.Proc. Nat. Acad. of Sciences.USA Vol 84 pp2727-2731

[Pine87] F.Pineda : Generalization of backpropagation to recurrent Neural Networks.Phys. Review Letter n 59 1987

[Regg87] J.Reggia:Properties of a competition based activation in neuromimetic network.Proc 1st Int Conf on Neural Network 1987

[Uchi89] T.Uchiyama.K.Shimohara.Y.Tokunaga A Modified Leaky Integrator Network for Temporal Pattern Processing.Proc IJCNN Conf. Washington June 89

AN ASYMMETRIC SPIN-GLASS MODEL OF LONG-TERM MEMORY IN A DYNAMIC NETWORK ARCHITECTURE

Valerio Cimagalli, Massimiliano Giona
Faculty of Engineering, University of Rome «La Sapienza», Via Eudossiana 18, 00184 Rome, Italy
Gianfranco Basti, Antonio Perrone
Pontifical Gregorian University, Piazza della Pilotta 4, 00187 Rome, Italy
Eros Pasero
Department of Electr. Engineering, University of Rome «Tor Vergata», Via O. Raimondo, 00173 Rome, Italy.

ABSTRACT

We are studying a particular dynamic architecture of neural network founded on the neurophysiological evidence and on the «spin-glass oriented» modelling. The key-concept of our model is to distinguish among the *coding function* of the input at the level of short-term memory (by a small and fast variance on the fixed weights extracting dynamic invariants), the *long-term storage* of the codes (by a learning on the weights) and the *recognition function* (by a matching process between new codes and learned codes). In this way, we can make the net capable of reckoning with non-steady inputs without falling in the «oscillatory catastrophe» or in the «noise catastrophe» during the learning phase on the weights (LTM). In this paper, after a general presentation of the architecture, we discuss a 2D *asymmetrical* spin-glass model of LTM. The main property of this model is its capability of *making stable*, under given conditions, *unstable equilibrium points* of a chaotic dynamics (*chaotic filtering*).

1. FAST CODING IN SHORT-TERM MEMORIES

In the classic paradigm of *static* neural networks, the coding function of the input is generally devoted to the LTM connection weights, acting as a sort of «gate» on the STM activations.

Generally, the learning of the LTM traces and/or the self-organization capacity of the net follow an Hebbian rule. Many problems arise when such an architecture must reckon with *non-steady* inputs. The radical solution would be the attribution not only of a *buffering* function to the STM as in Grossberg's ART architecture [1], but the attribution also of a *coding* function to it, so that the LTM has to learn and/or to recognize only pre-processed codes. But if we want to ascribe a pre-processing function to a layered STM extracting in real time, step by step, even more general dynamical invariants from the continuous statistical input, we need another variable in addition to the activations and the weights of the classical models.

The neurophysiological evidence suggests the hypothesis that each neuron could be considered as a *coincidence detector* in the time domain [2]. A network of coincidence detectors could thus exhibit another form of *fast* cell assembling, different from the Hebbian one, called *synaptic patterns*. Different synaptic mechanisms, related to the releasing of the mediator at the dendritic spine [3-4], could grant the the capacity of the synapse of defining and continuously modifying a «temporal window» within which all the impinging stimuli can be considered as simultaneous. This further control on the *strength* of the transmitted impulse for every cycle could thus allow the fast establishment of good timing relationships among cells. In the meanwhile, it could furnish us with the supplementary variable that we are searching for. Finally, the metastable character of the net could grant the immediate resetting of the memory for the next detection. To sum up, if we want only a *dynamic STM storage* in the net to perform input driven coding operations, we must suppose a net with fixed synaptic weights s and fast variable synaptic strengths w. On the contrary, if we want a *static LTM storage*, we must also modify the permanent synaptic weights s.

At present, different examples are already available of this dynamic preprocessing of the input at the STM level in the mammalian brain. For instance, the multilayered structure of the mammalian visual cortex grants very naturally the extraction of *position invariants* of the input features by the simple loss of retinotopy of the stimulus among successive layers of the visual cortex [2]. Moreover, the evidence of a *synchronization in phase* (after a *chaotic* state during the resting phase for the resetting of the STM) of the oscillatory behavior of the cells within the same column of feature detectors or among homologous columns in the primary visual cortex of the cat, might be a way to establish relations between features in different parts of the visual field [4-7].

2. ASYMMETRIC SPIN-GLASS CONTINUOUS MODELS OF DYNAMIC STM AND LTM

Our STM model requires that the synaptic weights are *fixed*, and only their strengths are *fast variable* within a finite range. Essentially this architecture can be modelled in a continuous form by considering that each weight J_{ij} in the network is a random (Gaussian) variable with a given mean (e.g. $\bar{J} = 0$) and a specified *variance* $[J^2{}_{ij}]_j$. H. Sompolinsky and his colleagues [8] recently proposed a modified Hopfield model incorporating such a condition and constituted by N non-linear oscillators interacting via *random asymmetric couplings*. The dynamics of the net consists of N coupled first order differential equations:

$$h_i = -h_i + \sum_{j=1}^{N} J_{ij} S_j = -h_i + \sum_{j=1}^{N} J_{ij}\, \Phi\,(h_i) \tag{1}$$

where J_{ij} is the synaptic efficacy (i.e., Von der Malsburg's synaptic weights s), h_i is the local field associated to each neuron: $-\infty < h_i < +\infty$; $\phi(x)$ is a non-linear *gain* function defining the input (h_i) – output (S_i) characteristic of the neuron: $\phi(x) = \tanh(gx)$. Such a dynamics, governed by the dimensionless control parameter gJ, shows a chaotic behavior for $gJ > 1$. Particularly, with $N \geq 100$, as gJ is increased above unity, the system pass through different limit cycle attractors, till for $gJ > 2$, the system falls in a chaotic attractor. Such an architecture shows a (metastable) temporal coding of the input (limit cycle) made progressively invariant by a sequence of layers obeying to Eq.(1) (see above, Sect.1). In this way, the periodic final output of this «global STM» is a compressed code ready to be learned (if it is stable for a sufficient long time) and/or recognized by a «global LTM» [9,10]. This global LTM [11] is another Hopfield asymmetric net having this time a modified Hebbian rule:

$$\frac{d J_{ij}(t)}{dt} = [\, -\lambda J_{ij}(t) + \bar{S}_i(t)\, \bar{S}_j(t)\,]\, f(\, \bar{S}_i(t)\, \bar{S}_j(t)\,)$$

$$\bar{S}_i(t) = \frac{1}{T} \int_0^T S_i(t-\tau)\, d\tau \tag{2}$$

The correlation function f in (2) makes the net capable of switching off the learning for unstable inputs (transition to chaos). The net is thus able to *distinguish dynamically* between successful (recognition or learning) states (=limit cycle, time *independent* states) and unsuccessful states (=chaotic, time *dependent* states). This use of the chaos as a «safety device» preserves very naturally the LTM traces from the «oscillatory catastrophe».

3. A STUDY OF THE FILTERING PROPERTIES OF THE TEMPORAL CORRELATION IN AN ASYMMETRICAL 2D SPIN-GLASS MODEL OF LTM

We have studied a bidimensional asymmetrical spin-glass model of LTM that, under given conditions, *can make stable unstable equilibrium points of a chaotic dynamics.* We defined this property as *chaotic filtering* [12-13]. The dependence of the stable output of the net on the correlation with a series of precedent states is the main characteristic of our net. We define this characteristic as the *inner dynamic memory of the net.*

Let be $S_{ij}^{(n)} \in M_{NxN}(\Sigma)$ the state of the spin at the time step (n), where $M(\Sigma)$ is the bidimensional matrix of the states and $\Sigma = \{-1, 1\}$.

If we indicate with $J_{ijml}^{(n)}$ the connection weights of our 2D net, then the state dynamics is given by:

$$S_{ij}^{(n+1)} = T_{ij}\,[\, H^{Nmem}(\, S^{(n)})\,] \tag{3}$$

where

$$T_{ij}[\,X\,] = \begin{cases} +1 & \text{if } \sum_{ml} J_{ijml}^{(n)} X_{ml} \geq 0 \\ -1 & \text{if } \sum_{ml} J_{ijml}^{(n)} X_{ml} < 0 \end{cases} \tag{4}$$

with

$$H_{ij}^{Nmem}(\, S^{(n)}) = \sum_{K=0}^{Nmem-1} S_{ij}^{(n-k)} \tag{5}$$

H is the filter operating the time correlation with dimension *Nmem*. The dynamics of the weights is given by the following asymmetrical evolution including a modified Hebbian rule:

$$J_{ijml}^{(n+1)} = \begin{cases} J_{ijml}^{(n)} & \text{if} \quad f[\,C_{ijml}^{Nmem}(S^{(n)})\,] < V \\ (1-\lambda)J_{ijml}^{(n)} + \lambda H_{ij}^{Nmem}(S^{(n)})\,C_{ijml}^{Nmem} & \text{if} \quad f[\,C_{ijml}^{Nmem}(S^{(n)})\,] \geq V \end{cases} \qquad (6)$$

where $C_{ijml}{}^{Nmem}$ is the following correlation:

$$C_{ijml}^{Nmem} = \sum_{k=0}^{Nmem-1} S_{ij}^{(n-k)}\,S_{ml}^{(n-k)} \qquad (7)$$

V is the correlation threshold and *f* is the sigmoidal function :

$$f(x) = \frac{1}{1+e^{-x/T}} \qquad (8)$$

where *T* is the temperature of the system. The presence of the term $H_{ij}{}^{Nmem}(S(n))$ makes asymmetrical the learning rule.

The properties of the time-correlation typical of this kind of dynamic filtering (i.e.,«chaotic filtering») [12-13] depend on the dynamics of the state evolution as well as on the dynamics of the weights. The filtering grants the stabilization (i.e., a learning) for a sufficient extension of the dynamic memory (*Nmem*). On the contrary, with an insufficient memory, the asymmetrical term predominates, so that the net exhibits a chaotic behavior. The metrical parameter $d^{(n+1)}$:

$$d^{(n+1)} = \sum_{ij} |\,S_{ij}^{(n+1)} - S_{ij}^{(n)}\,| \qquad (9)$$

represents a measurement of the difference between successive outputs of the net. In the *Figures 1* and 2, the temporal evolution of this parameter is represented for different memory dimensions (*Nmem*) and *N*=7. For a sufficiently high dimension, the output is stable, for a low dimension, the output is chaotic. The chaotic behavior is enhanced in the

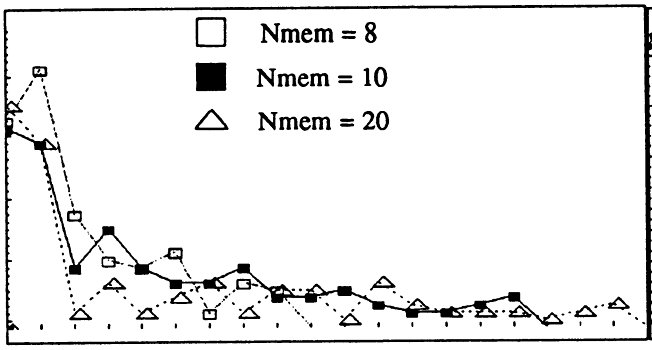


Fig.1. *d* vs. time for different *Nmem*

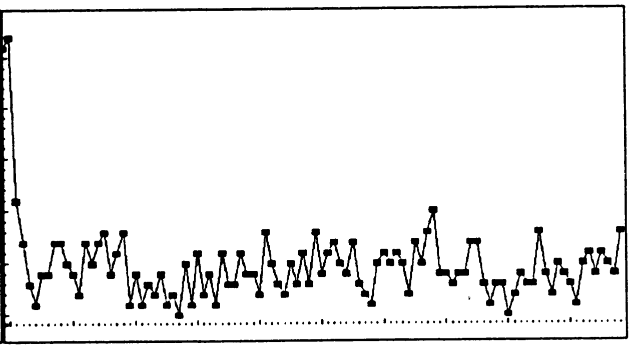


Fig.2. As Fig.1. Irregular case for *Nmem* = 4

Figure 3 describing the evolution, for a memory of dimension 1, of a typical parameter of this net: the spatio-temporal correlation parameter $z^{(n+1)}$:

$$z^{(n+1)} = \frac{1}{N^2} \sum_{ij} |\,S_{ij}^{(n+1)} - S_{ij}^{(n)}\,| * i * j \qquad (10)$$

In this net, the correlation is indeed a function not only of the time, but also of the position of the interacting spins.

To sum up, with respect to the other classical learning rules, the essential feature of our model is the presence of an inner memory granting a dynamic processing of the information. Moreover, it is this same characteristic that grants the self-stabilizing property of the net. In particular, it can avoid the oscillatory catastrophe in presence of too oscillating inputs, by the presence of the term $C_{ijml}{}^{Nmem}$ of *Eq.6*. Its function in our model is very similar to that of the correlation integral of Parisi's model in *Eq.2*. Nevertheless, there are two main differences: *1)* the chaotic filtering is something more than a simple correlation; *2)* the net is always asymmetrical also after the learning.

4. DISCUSSION AND PERSPECTIVES

Generally, the comprehension of the cognitive function of a chaotic dynamics in a neural net is today blocked within an insuperable dichotomy. *1)* If we suppose that the incoming stimulus is inserted into the initial conditions of the dynamics, we obtain an high selective power of the net, but we loss any categorization property of it, owing to its strong dependence on the initial conditions. *2)* On the other hand, if we insert extrinsically the input in the control parameter switching the dynamics from a chaotic to a stable state, we obtain a sort of categorization property (similar inputs are mapped into close limit cycles), but in such a case we do not obtain any appreciable advantage with respect to more classical models such as a static Hopfield net. The necessity of an external teacher is requested in both the cases.

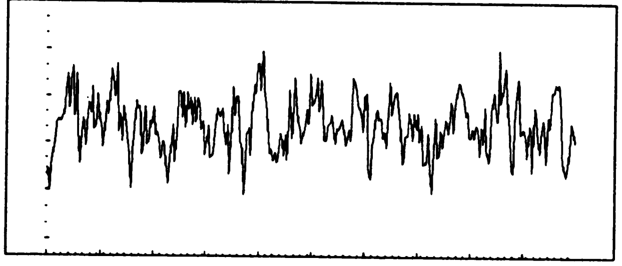


Fig.3. Chaotic behavior of z vs. n($Nmem$=1, N=7)

On the contrary, in our maps, owing to their self-filtering capacity, it is granted that similar inputs are mapped in very similar ways, though they are inserted into the initial conditions of the dynamics. Indeed, the mapping or the self-stabilizing function of the map corresponds here to an extraction of some essential ***invariant*** of the ongoing dynamics. In this sense, it is obvious to expect that two correlated maps of this type receiving a similar input will oscillate in phase only when they detect the same invariant. The synchronization of the oscillating behavior of the neural feature detectors of the mammalian brain (see above, Sect.1) could have thus in the chaotic filtering properties its more natural explication.

At present, we are studying a more sophisticated version of the model presented here, that is, a net exhibiting a *non-autonomous evolution* depending on the oscillation of its capacity of dynamic memory (*Nmem*).

REFERENCES

[1] S.Grossberg, "Nonlinear neural networks: principles, mechanisms and architectures", *Neural Networks*, vol. 1, pp.17-61, 1988.

[2] C.Von der Malsburg & E.Bienenstock, "Statistical coding and short-term synaptic plasticity: a scheme for knowledge representation in the brain". In E.Bienenstock (Ed.), *Disordered System and Biological Organization*, NATO ASI Series, Vol. F20, Berlin-Heidelberg-New York, 1986, pp.247-271.

[3] W.Singer, "The role of acetylcholine in use-dependent plasticity of the visual cortex". In M.Steriade & D.Biesold (Eds.), *Brain Cholinergic Systems*, Oxford-New York, 1989. In Press.

[4] W.Singer, "Self-organization in cognitive systems". In Eccles J.C. & Creutzfeldt O. (Eds.). *The Principles of Design and Operation of the Brain. Proceedings of the Study Week Organized by the Pontifical Academy of Sciences, Vatican City, October 19-24, 1988*, Vatican City and Berlin-Heidelberg, New York, 1989, In Press.

[5] C.M.Gray, P.Koenig, A.K.Engel & W.Singer, "Oscillatory responses in cat visual cortex exhibit intercolumnar synchronization which reflects global stimulus propereties", *Nature*, Vol. 338-6213, pp.334-337, 1989.

[6] C.M.Gray & W.Singer, "Stimulus-specific neuronal oscillations in orientation columns of cat visual cortex". In *Proceedings of the National Accademy of Sciences USA*, vol. 86, pp. 1698-1702, 1989.

[7] M.Livingstone & D.Hubel, "Segregation of form, color, movement and depth: anatomy, physiology and perception", *Science*, Vol. 240, pp. 740-749, 1988.

[8] H.Sompolinsky, A.Crisanti & H.J.Sommers, "Chaos in random neural networks", *Physical Review Letters*, Vol.61, 259-262, 1988.

[9] G.Basti & A.Perrone, "On the cognitive function of deterministic chaos in neural networks". In *Proceedings of IEEE/INNS IJCNN-89. International Joint Conference on Neural Networks*, Washington D.C., June 18-22, 1989, Vol.1, Washington, pp.657-663, 1989.

[10] G. Basti & A.Perrone, "Time-dependent short-term memories in neural networks". In *Proceedings of Second Italian Workshop on Parallel Architectures and Neural Networks, Vietri sul Mare, April 26-28*, 1989, London, 1989. In Press.

[11] G.Parisi, "Asymmetric neural networks and the process of learning", *Journal of Physics A: Mathematics General*, Vol.19, pp.L675-L680, 1986.

[12] M. Giona, "Recursive filtering of chaotic maps". To be published.

[13] G.Basti, V.Cimagalli, M.Giona, E.Pasero & A.Perrone, "Fast coding in short-term memories of a dynamic network architecture", submitted to *IEEE NIPS-89*, November 27-30, 1989, Denver, Colorado.

Sensitivity of Layered Neural Networks to Errors in the Weights

Maryhelen Stevenson Rodney Winter Bernard Widrow
Stanford University Department of Electrical Engineering, Stanford, CA 94305-4055

Abstract

An important consideration when implementing neural networks with either limited precision digital or analog hardware is the sensitivity of neural networks to weight errors. In this paper, we derive an approximation for the sensitivity of layered feed-forward networks of Adaline elements (threshold logic units) to weight errors. It is shown that for large networks with small weight errors, the probability that an output neuron makes a decision error can be approximated by

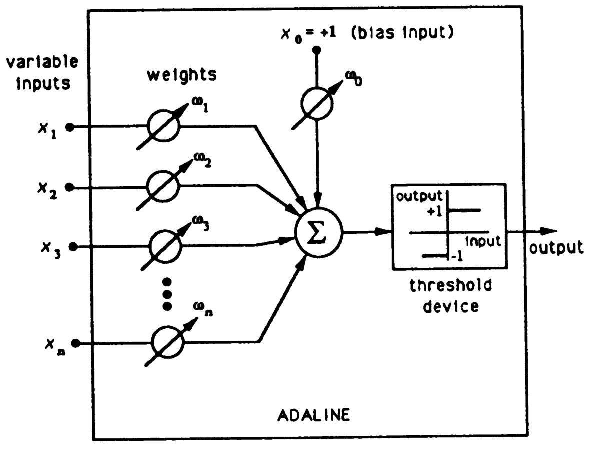
$$\mathrm{PE}_L \approx \frac{4}{\pi^2}\left[\frac{\pi}{4}\frac{|\Delta\vec{W}|}{|\vec{W}|}\right]^{2^{1-L}}$$

where L is the number of layers of the network, and $\frac{|\Delta\vec{W}|}{|\vec{W}|}$ is the normalized weight error or the "weight perturbation ratio." The probability of error increases with L and with $\frac{|\Delta\vec{W}|}{|\vec{W}|}$, but is essentially independent of the number of weights per neuron and of the number of neurons per layer, as long as these numbers are large, on the order of 100 or more.

1 Introduction

The input-output function realized by a neural network is determined by the values of its weights. When using analog hardware to store the desired weights, an important issue is that of weight sensitivity; how sensitive is the input-output mapping of the neural network to weight drift? In this paper, we investigate this question for neural networks with Madaline structures [5].

The *Adaline* (*ad*aptive *lin*ear *e*lement) [5] (also known as a linear threshold unit) is the basic building block of the *Madaline* (*m*any *Adalines*) network. Figure 1a shows an Adaline with n variable inputs: $x_1, x_2, \ldots, x_n$. The inputs take on binary values of either $+1$ or -1. The bias input, x_0, is fixed at a value of $+1$. Associated with the Adaline are $n+1$ adjustable analog weights: $w_0, w_1, \ldots, w_n$. The weights of the Adaline scale the corresponding inputs, the scaled inputs are summed, and the weighted sum is input to a threshold device. The threshold device outputs a -1 for negative inputs and a $+1$ for positive inputs. The output of the threshold device is the Adaline output. In geometric terms, the Adaline output is $+1$ if the angle between the input vector $X \triangleq [x_0 \ldots x_n]$ and the weight vector $W \triangleq [w_0 \ldots w_n]$ is less than $90°$ and -1 if the angle between these two vectors is greater than $90°$.



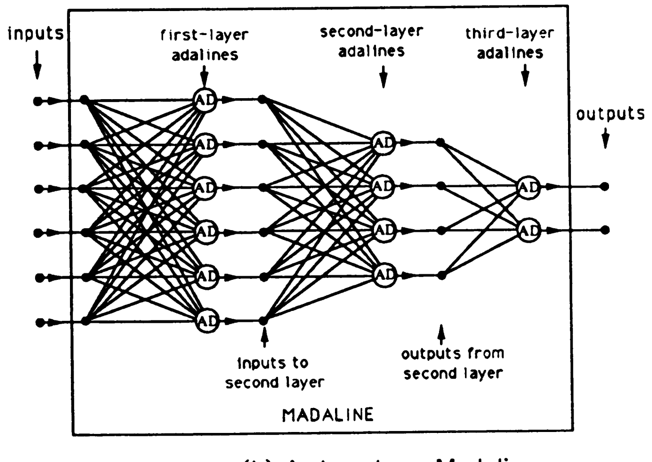


Figure 1: (a) The Adaline (b) A three-layer Madaline

A layered network of Adaline elements (a Madaline) is shown in Figure 1b. The inputs to the network are presented to each of the Adalines in the first layer. The outputs from the first-layer Adalines then serve as inputs to the second-layer Adalines, and so on. The Adalines of the final layer (in this case, the third layer) are called the output Adalines. Their outputs are the outputs of the network.

2 Effects of Weight and Input Errors on the Adaline Output

As stated in the previous section, the angle between the input vector and the weight vector determines the output of the Adaline. Changes in either the input vector or the weight vector can cause the angle between these two vectors to change. In this section we report some recent findings regarding the probability that the Adaline output changes state as a result of errors in either the weight vector, the input vector, or both.

Consider an Adaline with weight vector W and input vector X. We first discuss the effects of errors in the weight vector on the Adaline output response. For this purpose, we consider a perturbed weight vector W_p which can be expressed as the sum of the original weight vector W and a randomly oriented perturbation vector ΔW: $W_p = W + \Delta W$. Using the concepts of higher dimensional geometry [3, 6, 1, 2], it can be shown that the probability of change in the Adaline output state as a result of the weight vector perturbation ΔW is given by $\frac{1}{\pi}\theta_{WW_p}$ where θ_{WW_p} is the angle between W and W_p. Given the length of the perturbation vector $|\Delta W|$, the expected value of this angle is approximated by the weight perturbation ratio, $\frac{|\Delta W|}{|W|}$. Hence, the probability of change in the Adaline output state due to a randomly oriented weight vector perturbation ΔW is approximated as:

$$\text{P(Adaline Error)} \approx \frac{1}{\pi}\frac{|\Delta W|}{|W|} \tag{1}$$

As intuition would suggest, errors in the input vector have the same effect as errors in the weight vector[4]; intuitively, this is because it is the angle between these two vectors that determines the Adaline output response. Thus, the probability that the Adaline output changes state due to a randomly oriented input vector perturbation ΔX is given by:

$$\text{P(Adaline Error)} \approx \frac{1}{\pi}\frac{|\Delta X|}{|X|} \tag{2}$$

When both input errors and weight errors are present, we can define an equivalent net perturbation ratio. This equivalent net perturbation ratio is approximated by the square root of the sum of the squares of the input perturbation ratio and the weight peturbation ratio[4]. The probability that the Adaline output changes state due to an input vector perturbation ΔX and to a weight vector perturbation ΔW is given by:

$$\text{P(Adaline Error)} \approx \frac{1}{\pi}(\text{net perturbation ratio}) \approx \frac{1}{\pi}\sqrt{\left(\frac{|\Delta X|}{|X|}\right)^2 + \left(\frac{|\Delta W|}{|W|}\right)^2} \tag{3}$$

The approximations made in this section are valid for large numbers of Adaline inputs and small weight and input perturbation ratios.

3 Effects of Weight Errors on a Madaline Output

We now investigate the sensitivity of a network of Adalines to changes in the weights. Consider a Madaline network with n_1 first-layer Adalines, n_2 second-layer Adalines, ..., and n_L L^{th}-layer Adalines, where L is the number of layers in the network. Suppose the weight vector associated with each Adaline of a multilayer network is perturbed in a random direction by an amount which results in a weight perturbation ratio, $\frac{|\Delta W|}{|W|}$. Then the probability that a first-layer Adaline makes an error, PE_1, is given by: $\text{PE}_1 \approx \frac{1}{\pi}\frac{|\Delta W|}{|W|}$. Assuming the weight vectors of the first-layer Adalines to be independent, the expected number of first-layer Adalines which make decision errors is $n_1\text{PE}_1$. The input perturbation ratio corresponding to k out of $n+1$ binary-valued (± 1) inputs being in error is $\sqrt{\frac{4k}{n+1}}$. Substituting $n_1\text{PE}_1$ in place of k results in an input perturbation ratio for the second-layer Adalines of:

$$\left(\frac{|\Delta X|}{|X|}\right)_{2^{nd}\text{-layer}} \approx \sqrt{\frac{4n_1\text{PE}_1}{n_1+1}} \approx \sqrt{4\text{PE}_1} \tag{4}$$

The probability of error for a second-layer Adaline, PE_2, is found by substituting this expression for the input perturbation ratio in Equation 3.

$$\mathrm{PE}_2 \approx \frac{1}{\pi}\sqrt{\left(\frac{|\Delta W|}{|W|}\right)^2 + 4\mathrm{PE}_1} = \frac{1}{\pi}\frac{|\Delta W|}{|W|}\sqrt{1 + \frac{4}{\pi}\frac{|W|}{|\Delta W|}} \tag{5}$$

Continuing to propagate the probability of error from one layer to the next in this fashion, it is found that the probability of error for an output of a Madaline with L layers of Adalines is approximately:

$$\mathrm{PE}_L \approx \frac{1}{\pi}\frac{|\Delta W|}{|W|}\sqrt{1 + \frac{4}{\pi}\frac{|W|}{|\Delta W|}\sqrt{1 + \frac{4}{\pi}\frac{|W|}{|\Delta W|}\sqrt{\cdots\sqrt{1 + \frac{4}{\pi}\frac{|W|}{|\Delta W|}}}}} \tag{6}$$

where the number of square roots in this approximation for PE_L is $L-1$. For small weight perturbation ratios, $\frac{4}{\pi}\frac{|W|}{|\Delta W|}$ is much bigger than 1 and Equation 6 can be approximated as:

$$\mathrm{PE}_L \approx \frac{1}{\pi}\frac{|\Delta W|}{|W|}\left(\frac{4}{\pi}\frac{|W|}{|\Delta W|}\right)^{\frac{2^{L-1}-1}{2^{L-1}}} = \frac{4}{\pi^2}\left[\frac{\pi}{4}\frac{|\Delta W|}{|W|}\right]^{2^{1-L}} \tag{7}$$

This last simplification has the advantage of resulting in a closed-form expression and is a good approximation to Equation 6 for values of $\frac{|\Delta W|}{|W|} < 10\%$.

4 Simulation Results

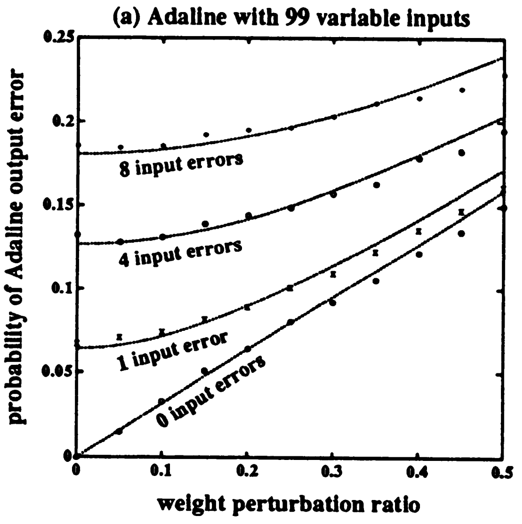
Two computer simulations were written to obtain experimental results for comparison with the theoretical results of the previous sections.

The purpose of the first simulation was to experimentally determine the relative frequency of Adaline error as a function of various combinations of weight and input perturbations. To do so, a reference weight and input vector were randomly generated. The reference vectors were then perturbed by randomly generated vectors of the desired length. The outputs of the reference and perturbed Adalines were compared to determine whether or not the perturbations resulted in an Adaline decision error. The experimental frequency of Adaline error is computed based on data from 18000 such trials.

The experimental results from the first simulation are compared with the theoretical results (Equation 3) in Figure 2a. The results shown are for an Adaline with 99 variable inputs (100 total inputs). The continuous curves illustrate the theoretical results whereas the data points depict the experimental results. Each curve shows the probability of an Adaline decision error as a function of weight perturbation ratio for a specific input perturbation ratio (given in terms of the number of input errors). The comparison indicates that agreement between the theoretical and experimental results is good. From results not shown, we see that good agreement holds for Adalines with as few as 9 inputs and for weight and input perturbation ratios as big as 50%. The derivation of Equation 3 assumes a large number of Adaline inputs as well as small weight and input perturbation ratios.

The second simulation found the experimental frequency of error for a Madaline output as a function of the weight perturbation ratio. A randomly generated weight vector was assigned to each Adaline of a reference network. A perturbed network was then generated from this reference network by adding a randomly oriented perturbation vector of desired magnitude to each of the Madaline's weight vectors. A randomly selected input vector was then presented to both networks and their outputs were compared. The experimental frequency of error is based on over 4000 such comparisons.

The results of the second simulation (for a Madaline network with 99 Adalines per layer) are compared with the theoretical results of Equation 6 in Figure 2b. Again the continuous curves represent the theoretical predictions and the data points depict the experimental results. The four curves shown correspond to networks with one, two, three, and four layers of Adalines. The experimental and theoretical results are in close agreement for weight perturbations as big as 50%. For networks with fewer numbers of Adalines per



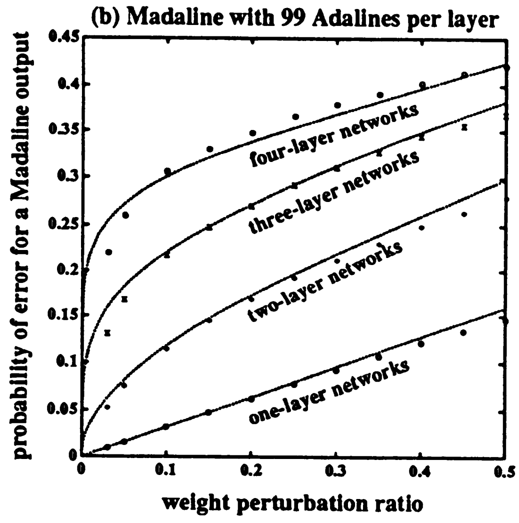


Figure 2: Comparison of theoretical and experimental results. Continuous curves illustrate theoretical results, data points depict experimental results. (a) Probability of Adaline decision error vs. weight perturbation ratio for various input perturbation ratios. (b) Probability of Madaline output error vs. weight perturbation ratio for networks with various numbers of layers.

layer, the experimental results for small weight perturbation ratios (less than 5%) start to drop below the theoretical results predicted by Equation 6. If more accuracy is desired for the smaller weight perturbation ratios, a more complicated approximation[4] can be used.

5 Conclusion

In this paper, we have presented the results of our analysis on the sensitivity of a Madaline's input-ouput mapping to errors in the weights. We have presented a simple formula which approximates the probability of a Madaline output error as a fucntion of the weight perturbation ratio and number of layers in the network. This approximation is independent of the number of Adalines per layer; simulation results showed that for networks with 99 Adalines per layer, the approximation accurately predicts the performance of the networks for weight perturbation ratios between 5% and 50%. As the number of Adalines per layer increases, the approximation improves for the smaller weight perturbation ratios and maintains its good performance for weight perturbation ratios as high as 50%. In a forthcoming paper, we present our theoretical results in more detail and present a more precise approximation for the probability of a Madaline output error as a function of the weight perturbation ratio.

References

[1] F.H. Glanz, *Statistical Extrapolation in Certain Adaptive Pattern-Recognition Systems*, Ph.D. dissertation, Dept. Electrical Engineering, Stanford Univ., May 1965.

[2] M.E. Hoff, Jr., *Learning Phenomena in Networks of Adaptive Switching Circuits*, Ph.D. dissertation, Dept. Electrical Engineering, Stanford Univ., June 1962.

[3] D.M.Y. Sommerville, *An Introduction to the Geometry of N Dimensions* London, England: Methuen & Co., 1929.

[4] M. Stevenson, R. G. Winter, and B. Widrow, *Weight Errors and Output Errors in Layered Neural Networks*, to be published.

[5] B. Widrow and R. G. Winter "Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition," *IEEE Computer*, pp. 25-39, March, 1988.

[6] R. G. Winter, *Madaline Rule II: A New Method for Training Networks of Adalines*, Ph.D. dissertation, Dept. Electrical Engineering, Stanford Univ., Jan. 1989.

This research was sponsored by SDIO Innovative Science and Technology Office and managed by ONR under contract #N00014-86-K-0718, by the Department of the Army Belvoir RD&E Center under contract #DAAK 70-89-K-0001, and by a Hughes Aircraft Co. fellowship.

AN IMPROVEMENT ON SIMULATED ANNEALING AND BOLTZMANN MACHINE

Lei Xu
Lappeenranta University of Technology, Department of Information Technology
BOX 20, 53851 Lappeenranta, Finland
Premanent address: Dept. of Mathematics, Peking University, P.R.China

Abstract. A method is proposed for improving the performance of the commonly used simulated annealing(or SA for short) techniques. It produces the better solutions and can reduce the computer time. The similar improvement is also made on Boltzmann machine. The advantages of such an improvement are shown by computer simulation on attributed graph matching problem.

1. Introduction. SA not only has many applications[1][2], but also has many variants and extensions [3]. SA and its variants have also been used to simulate the networks' dynamics of the symmetrically interconnected neural networks. Boltzmann machine and its variant are such examples [4]. In this paper, a method is proposed which can improve the performance of SA and its variants. It can guarantee to always yield a better solution than that yielded by those original SA methods. It is useful especially in the following cases (which are often encounterd in actual applications) :
(1). The time spent on each temperature is not long enough to let Metropolis Sampling(or MS for short) process reach its equilibrium state. (case(1) is only suitable for the stationary annealing way as used in [1][4], but not for the nonstationary annealing way as used in [3] where temperature decreases at each step).
(2). the speed of the annealing is too fast.
(3). The temperature specified for stopping the annealing process is not low enough.

In these cases, the original methods will usually find a bad solution, while the improved methods can still obtain a quite better one. In addition, The improvment also supplied a simple but effective way to decide when a MS process can be finished to start another MS process(only for the sationary annealing way) and when the whole annealing process can stop such that time cost is reduced but the solution is still satisfactory.

Furthermore, the method is introduced into Boltzmann machine for improving its performance for combinatorial optimization use. The advantages similar to the ones given above are again obtained. In fact, the method probably gives a general strategy which is also suitable for improving some other neural network methods which relate to SA or even some stochastic combinatorial optimization methods. Finally, computer simulation on an attributed graph matching problem is briefly introduced to show the advantages of our method in comparision with the original ones.

2. An Improvement On Simulated Annealing Methods.

Where It Needs to Be Improved. It is intereseting to relate SA methods to the classical iterative improvment algorithm (or IIA for short) which iteratively chooses a potential state s' and compares its cost $E(s')$ with that $E(s)$ of the current state s, and then replaces s by s' according a accepting rule of $\Delta E = E(s') - E(s) < 0$. With

This work was supported by Tekes Grant 4196/1988 under Finsoft project

IIA, the current state will never reach a state which is worse than those reached before, and the current soultion updating(or CSOU for short) sequence $[E(s)]$ is a monotonically unincreasing sequence, as a result, the current state is easy to be stack at a local minimum.

However, with SA methods, s' is permit to replace s with probability $P(s')$ even when $\Delta E \geq 0$ (this could be considered as a generalized accepting rule of the above mentioned), this $P(s')$ deponds on ΔE and a parameter T (called temperature). The smaller ΔE and the higher T gives the larger $P(s')$. In order to reach a global optimal state, T should start at a high value and gradually decreases to a very low value in a way which imitates the annealing process of a physical system. As a result, the current state has a chance to escape from the local minimum, but in the same time, $[E(s)]$ is no long a monotonically unincreasing sequence. Theoretically speaking, when T is slowly reduced by a strict cooling schedule and $T \to 0$, the final current solultion will be the global optimal one with probability one.

But, practically, it is difficult to decide how high T should start and how low T sould stop. Furthermore, for the stationary annealing way, it is also difficlut to decide when a MS process at each T reaches its equilibrium and how slowly T should decreases from one to the other; or even for the nonstationary annealing way, although there is a theoretical formula (e.g., (3) in [3]) which indicates how to reduce T at each step, it is too slow to be practical. Instead, some fast cooling schedule are usually used. Because of these reasons, actually, the final current solution is probably worse than some solutions it ever encoutered before. This explains why the solution by SA methods sometime is even worse than that by some heuristic methods, and it also interprets the curve phenomenon in Fig.7d of ref. [2].

How It Can Be Improved. With both IIA and SA methods, the current state updating (or CSTU for short) track has two functions simultaneously. one is as the searching control track to indicates which states are searched, the other is as the CSOU track to indicates how the current solution is renewed. With IIA, the CSTU track acts as a bad searching control track since it is easy to be stack at a local minimum, but it acts as a good CSOU track since its current solution is the best one of those ever met before. Oppositely, with SA methods, the CSTU track acts as a good searching control track since it can escape from a local minimum, but it acts as a bad CSOU track since its current solution can be worse than those ever met before.

However, for SA methods, it is not neccessary to bundle up the two tracks. We can let the CSTU track be only as the searching control track so that all its good characteristics are retained, and construct a good CSOU $[E(\hat{s})]$ on which the current solution $E(\hat{s})$ is updated(i.e., $\hat{s}$ is replaced by s') only when $E(s') < E(\hat{s})$ so that the current solution is always the best one of those ever met before. This change only increases very few additional computer cost , but brings the following advantages:

(1). As stated above, not only all the good features of the original SA methods can be retained, but also the better final solution can be obtained.

(2). The track $[E(\hat{s})]$ supplies a way to stop the annealing process through checking if $E(\hat{s})$ remains unchanged for the p_0 successively reduced T's.(if so, stop)

(3). For stationary annealing way, $[E(\hat{s})]$ also supplies, at each T, a way to stop the MS process to start a new one through checking if $E(\hat{s})$ remains unchanged for the q_0

succesive steps. (if so, start a new MS)

An algorithm for improving the conventional SA [1] is given as follows:

(Initially, randomly choose a state s as the current state, set T, T_{min}, j_m. Let $p = 0, q = 0, j = 0$, $\hat{s} = s, E = E(s)$, $E_{\hat{s}} = E$, $E_t = E$.)

```
step 1 : If $j > j_m$, goto step 6. otherwise, $j = j + 1$, randomly make a small perturbation $\Delta s$ which results in a new state $s + \Delta s$ with $\Delta E = E(s + \Delta s) - E(s)$
step 2 : If $\Delta E > 0$, generate a random number $\xi$ which is a sample of the uniform distribution over [0,1]. If $e^{(-\Delta E/T)} < \xi$, goto step 1.
step 3 : $s + \Delta s$ replaces $s$ as the new current solution, $E \leftarrow E + \Delta E$.
step 4 : If $E < E_{\hat{s}}$ , then $E_{\hat{s}} \leftarrow E$, $\hat{s} \leftarrow s + \Delta s$, $q = 0$. Otherwise $q \leftarrow q + 1$.
step 5 : If $q < q_0$ , goto step 1.
step 6 : If $E_t < E_{\hat{s}}$ , then $E_t \leftarrow E_{\hat{s}}$ , $p = 0$, if not, $p \leftarrow p + 1$.
step 7 : If $p < p_0$ and $T > T_{min}$, reduce $T$ by some means, let $q = 0$, $j = 0$. goto step 1. otherwise, the current $E(\hat{s})$ is taken as the final solution, stop.
```

Where T_{min} is a given threshold for the minimum T and j_m is also a given threshold for the maximum time of a MS at each T. The two thresholds togather with q_0 and p_0 control when a MS at each T finishes and when the whole annealing stops.

3. The Improvement On Boltzmann Machine. The symmetrical interconnected neural networks has been applied to many combinatorial optimization problems [5]. In such networks, a global state is considered as a set of binary neurons c_i's with its enery E as given in formula(1) of Fig.2. The E is governed by an additive STM equation which relaxes the network Usually, it hopes that the networks could be relaxed to the state of its global minimum energy, for this end, Hinton [4] developed Boltzmann machine in which SA technique is used to avoid the local minimum of E in the following way:

Select a neuron c_i, calculate its energy gap ΔE_i (i.e., the difference between the energy of the two global states, with c_i off and on repectively) by formula(2) in Fig.2. and let the neuron ci take value $c_i = 1$ with probabilty $p_i = 1/(1 + e^{-\Delta E_i/T})$ Then, select another neuron and repeat the same process untill the equilibrium is reached. This process starts at a high temperature T, and with T gradually decreasing to a low enough value, the process stops with the current networks' global state being taken as the state of the global minimum energy.

In Boltzmann machine, its CSOU track is also identical to the CSTU track. So the improvement similar to that given in **sect.2** could be also obtained by constructing a new CSOT track as follows : **In addition to a binary matrix A which indicates the current global state of the network, another binary matrix B is used to record the global state of the current minimal energy. Initially, $A = B$ both record a randomly chosen global state, and then A will be updated once each neuron ci changes its value according to probability p_i, but B is updated by $B = A$ only when the energy of the current global state is lower than the energy of the state recorded in B.** In this way, a montonically unincreasing energy sequence of the current solution is obtained. When the whole process is stoped, the current state recorded in B could be taken as the final solution. Similarly, the three advantages mentioned in **sect.2** are still true here.

3. Simulation Examples. Attributed Graph (AG) have been widely used in computer vision. In [6], we used the methods of SA and Boltzmann machine to solve the AG matching problem. Due to the limited space, here, we only briefly mention an example which is done by the improved Boltzmann machine in comparision with the original one(both in the stationary annealing way). The details and more examples are refered to [6].

As shown in Fig.1, two AGs, G and G', each has 8 nodes v_i's with attributes av'_is and 10 edges $v_i v'_j$s with attributes ae_{ij}'s. The end is to set up an one-to-one correspondence between the nodes of G and G' so that the total cost E, as given in formula(3) of Fig.2, is minimized. In the formula, u_{ij}'s are 0-1 variables, $u_{ij} = 1$ denotes that v_i of G corresponds to v'_j of G' and its cost is $(av_i - av'_j)^2$. $u_{ij}u_{lk} = 1$ denotes that edge $v_i v_l$ of G corresponds to edge $v'_j v'_k$ and its cost is $(ae_{il} - ae'_{jk})^2$. In effect, the formula is just a version of formula(1) of Fig.2 and each U_{ij} is just a neuron.

By the original Boltzmann machine with the starting $T = 2000$, the final $T_{min} = 0.01$, and T is reduced by $T \leftarrow 0.96T$, when the annealing process finaly stops after 119194 steps, a bad and unfeasible (there are one-to-many correspondences) solution with cost $E = 1125$ is obtained. However, by the improved Boltzmann machine with the same parameter set, the annealing process finaly stops after 11253 steps at approximately $T = 800$ (due to the new stoping way, see **sect.2**), the global optimal solution is obtained with cost $E = 0.5$, so the example shows that the improved Boltzmann machine can get a much better solution with much less cost.

REFERENCES
[1] S.Kirkpatrick et al, Science, Vol.220, No.4598, 1983, pp671-680.
[2] E.H.L.Aarts et al, Philips J Res. Vol.40, No.4, 1985, pp193-226.
[3] M.Goldstein, et al, Proc. of IEEE ICNN88, ppII267-II273.
[4] G.E.Hinton et al, Tech. Rep. No.CMU-CS-84-119, may, 1984.
[5] J.Hopfield et al, Biological Cybernetics, Vol.52, 1985, pp141-152.
[6] L.Xu and E.Oja, "Improved Simulated Annealing, Boltzmann Machine and Attributed Graph matching", Tech. Rep., Lappeenranta Univ. Tech., May, 1989.

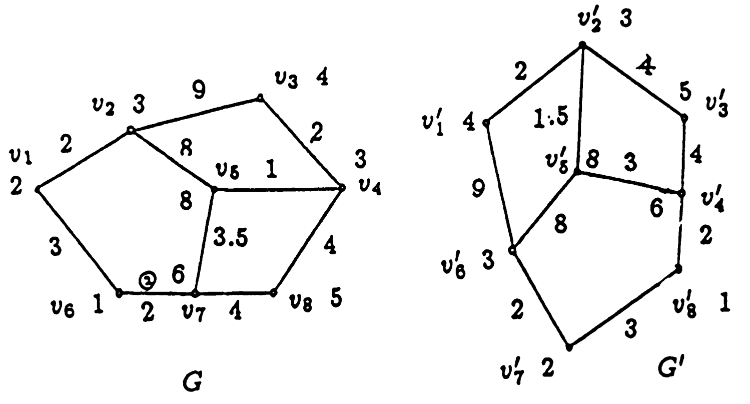


Fig.1

$$E = -0.5 \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} c_i c_j + \sum_{i=1}^{N} \theta_i c_i \qquad (1)$$

where θ_i is a threshold associated to neuron c_i.

$$\Delta E_i = \sum_{i=1, j \neq i}^{N} w_{ij} c_j - \theta_i \qquad (2)$$

$$E = a \sum_{i=1}^{N} (\sum_{j=1}^{N} (u_{ij} - 1)^2 + a \sum_{j=1}^{N} (\sum_{i=1}^{N} (u_{ij} - 1)^2 c_i c_j$$

$$+ b \sum_{i=1}^{N} \sum_{j=1}^{N} (av_i - av'_j)^2 u_{ij} \qquad (3)$$

$$+ c \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{l=1, l \neq i}^{N} \sum_{k=1, k \neq j}^{N} (ae_{ij} - ae'_{lk})^2 u_{ij} u_{lk}$$

Fig.2

PROGRAMMING NEURAL NETWORKS: A Dynamic-Static Model

Yong Yao, and Qing Yang
Department of Physiology, University of California, Berkeley, CA. 94720

We propose a dynamic-static model of neural networks (DS_net). The circuitry of the model is presented. Theoretical analysis shows that the set of the equilibria of the DS_net is associated with the set of the minima of a certain energy function. The energy function may be more general than that of the Hopfield-Tank network. Applications of the DS_net have been made to solve linear programming as well as quadratic programming problems.

1. Introduction

Using neural networks to solve optimization problems has been pioneered by Hopfield and Tank [1],[2]. In an appropriate neural network, the dynamic processing from its initial state to its final state may be considered as a computation for a certain optimization problem. The final state is one of the solutions of the problem. From this concept, we are going to develop a more general model which includes Hopfield-Tank model as a special case. The main property of our model is to associate the set of the minima of a certain optimization problem with the set of the equilibria of the model.

In a earlier work [3], a fast-slow neural network model (FS_net) was discussed. The idea of the FS_net is to introduce different time scales, i.e. the network is composed of several subnetworks, some of them have faster dynamics than others. This idea was suggested by analyzing the dynamics of a real brain. For instance, the process of recognition is usually much faster than the process of learning in the brain. The time scales of the changing of synapse weights and the changing of axon hairs might not be identical, e.g. those changings are treated by using different differential equations in [4]. Although in the brain it is not necessary that physically those subnetworks with different time scales are separated since the brain is of three-dimension. For example, in an olfactory bulb, Stellate cells, which are considered as supporting neurons, are embedded into the network composed of Mitral cells and Granule cells [5]. It is plausible to consider that the time scale of the Mitral-Granule cell network is different from that of the Stellate cell network. In this paper, we discuss a particular case of the FS_net. In this case the time scales of the fast subnetworks are considered as infinite. That is, the dynamic processing of the fast subnetworks is negligible. This kind of FS_net is called a dynamic-static neural network (DS_net).

2. A Dynamic-Static Model

Fig.1 shows a DS_net with one dynamic subnetwork and one static subnetwork. The left subnetwork is dynamic with a bypass RC circuit, and the subnetwork on right is static. R_i simulates the transmembrane conductance. C_i simulates the transmembrane capacitance. $\mathbf{I}^{(D)}$ is the external input to the dynamic subnetwork, while $\mathbf{I}^{(S)}$ is the external input to the static subnetwork.

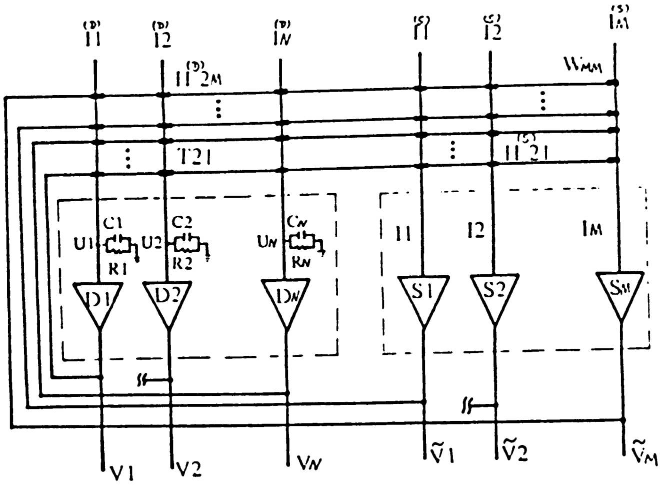


Fig.1 The circuitry of a dynamic-static neural network (see the context for the details).

In Fig.1, $\mathbf{D}_i$, $\mathbf{S}_k$ are nonlinear elements of the DS_net. They may be referred to "neurons". The neuron $\mathbf{D}_i$ and the neuron $\mathbf{S}_k$ may be of different functions. But $\mathbf{D}_1,\mathbf{D}_2,\ldots\mathbf{D}_N$ are identical. $\mathbf{S}_1,\mathbf{S}_2,\ldots\mathbf{S}_M$ are identical. The functions f, and g are used to denote the input/output relationships of the D-type neuron and the S-type neuron, respectively. That is $V_i = f(u_i)$, and $\tilde{V}_k = g(I_k)$. T_{ij} represents the synaptic strength from $\mathbf{D}_j$ to $\mathbf{D}_i$. W_{kl}, $H^{(D)}{}_{ik}$, $H^{(S)}{}_{ki}$ have the similar meanings to T_{ij}. Thus the matrices $\mathbf{T}$, $\mathbf{W}$, $\mathbf{H}^{(D)}$, and $\mathbf{H}^{(S)}$ are the connection matrices from D to D, S to S, S to D, and D to S, respectively. Mathematically, the DS_net can be described by the following differential-algebraic equations

$$C_i \frac{du_i}{dt} = -\frac{u_i}{R_i} + \sum_{j=1}^{N} T_{ij} V_j + I_i^{(D)} + \sum_{l=1}^{M} H^{(D)}{}_{il} \tilde{V}_l \quad (1)$$

$$I_k = \sum_{j=1}^{N} H^{(S)}{}_{kj} V_j + I_k^{(S)} + \sum_{l=1}^{M} W_{kl} \tilde{V}_l. \quad (2)$$

Although generally the collective behavior of the DS_net is unknown, the network can be proved to be of Lyapunov-type * if it satisfies the following conditions: **T** is symmetric, $\mathbf{H}^{(D)}$ equals to the transposition of $\mathbf{H}^{(S)}$, **W** is a zero matrix (i.e. there is no feedback from the static subnetwork to itself), the I/O relationship of the D-type neuron, i.e. the function f, is a strictly monotonically increasing function, and f and g are integrable.

To see this, let us consider the following function

$$E = -1/2 \sum_{i=1}^{N} \sum_{j=1}^{N} T_{ij} V_i V_j - \sum_{i=1}^{N} I_i^{(D)} V_i$$

$$+ \sum_{i=1}^{N} 1/R_i \int_0^{V_i} f^{-1}(v)dv - \sum_{l=1}^{M} G(I_l) \quad (3)$$

where the function $G(z)$ is defined by $G(z) = \int_0^z g(x)dx$. Since $V_i = f(u_i)$,

$$\frac{dE}{dt} = \sum_{i=1}^{N} \frac{\partial E}{\partial V_i} \frac{dV_i}{dt}$$

$$= \sum_{i=1}^{N} \frac{dV_i}{dt} [\frac{u_i}{R_i} - \sum_{j=1}^{N} T_{ij} V_j - I_i^{(D)} - \sum_{l=1}^{M} H^{(S)}{}_{li} g(I_l)] (4)$$

Notice $\tilde{V}_l = g(I_l)$, $\mathbf{H}^{(S)} = (\mathbf{H}^{(D)})'$, where $'$ denotes the transposition of a matrix, $\frac{dV_i}{dt} = \frac{df}{du_i}\frac{du_i}{dt}$, we have

$$\frac{dE}{dt} = \sum_{i=1}^{N} \frac{dV_i}{dt} (-C_i \frac{du_i}{dt})$$

$$= -\sum_{i=1}^{N} C_i \frac{df}{du_i} (\frac{du_i}{dt})^2. \quad (5)$$

Furthermore f is supposed to be monotonically increasing, i.e. $\frac{df(z)}{dz}$ is positive. This yields

*A dynamic system is called Lyapunov-type if there exists a function so that the value of the function is strictly decreasing along with every dynamic flow of the system except at the equilibria of the system. The function is called the energy (or Lyapunov) function of the system. In terms of the definition, there is not any kind of oscillation in a Lyapunov system.

$$\frac{dE}{dt} < 0, \ \frac{dE}{dt} = 0 \text{ if and } only \text{ if } \frac{du_i}{dt} = 0. \quad (6)$$

This shows the function **E** in (3) is an energy function of the DS_net under the above conditions. The equilibria of the DS_net and the minima of **E** are equivalent. In other words, the neural network in Fig.1 may be used to seek the minima of the function **E**.

The similar proving procedure may be applied to show the DS_net in Fig.1 is of Lyapunov-type under the conditions: $\mathbf{T} = \mathbf{T}'$, $\mathbf{H}^{(D)} = \mathbf{0}$, i.e. the dynamic subnetwork is symmetry and independent of the static subnetwork, f is a strictly monotonically increasing function, and f and g are integrable. In this case the energy function is chosen to be

$$E = -1/2 \sum_{i=1}^{N} \sum_{j=1}^{N} T_{ij} V_i V_j - \sum_{i=1}^{N} I_i^{(D)} V_i$$

$$+ \sum_{i=1}^{N} 1/R_i \int_0^{V_i} f^{-1}(v)dv \quad (7)$$

Since under the conditions the equation (1) is reduced to be

$$C_i \frac{du_i}{dt} = -\frac{u_i}{R_i} + \sum_{j=1}^{N} T_{ij} V_j + I_i^{(D)} \quad (8)$$

and

$$\frac{dE}{dt} = \sum_{i=1}^{N} \frac{dV_i}{dt} [\frac{u_i}{R_i} - \sum_{j=1}^{N} T_{ij} V_j - I_i^{(D)}]$$

$$= -\sum_{i=1}^{N} C_i \frac{df}{du_i} (\frac{du_i}{dt})^2. \quad (9)$$

This yields the conclusion (6). For summarying, we have the following theorem about the DS_net

Theorem: Let the functions f and g be integrable, and f is strictly monotonically increasing. The DS_net in Fig.1 is of Lyapunov-type if either of the following two sets of conditions is satisfied: (i) $\mathbf{T} = \mathbf{T}'$, $\mathbf{H}^{(D)} = (\mathbf{H}^{(S)})'$, and $\mathbf{W} = \mathbf{0}$; (ii) $\mathbf{T} = \mathbf{T}'$, and $\mathbf{H}^{(D)} = \mathbf{0}$.

Remarks: (i) There is no restriction to be added to the function g to guarantee the DS_net to be of Lyapunov-type provided g is integrable. g is mainly related to the static properties of the network, while the Lyapunov-type property is related to the dynamic processing. But in the next sections we will find that an appropriate choice of the function g plays an important role to implement constraints in a constrained programming problem.

(ii) The DS_net includes the Hopfield-Tank network as a special case, i.e. when $M = 0$. In addition as it was pointed out in [2] that the Hopfield-Tank's linear programming network does not belong to their general model, this network can be deduced from the DS_net model (see the next section).

3. Linear Programming

A linear programming problem can be described to attempt to minimize a cost function

$$\phi(\mathbf{V}) = \mathbf{A}^T\mathbf{V} \qquad (10)$$

subject to

$$\mathbf{E}_j^T\mathbf{V} \geq B_j, \qquad j=1,2,\ldots,M \quad or$$
$$\mathbf{E}^T\mathbf{V} - \mathbf{B} \geq 0. \qquad (11)$$

where $\mathbf{V}$ is a N-vector variable, $\mathbf{A}$ is a N-vector, $\mathbf{E}$ is a $N\times M$ matrix, and $\mathbf{B}$ is a M-vector. In Fig.1, let $T_{ij}=0$, $W_{kl}=0$, $I_k^{(S)}=-B_k$, $I_i^{(D)}=-A_i$, $H_{il}^{(D)}=H_{li}^{(S)}=E_{il}$, f be an appropriate sigmoid curve, g be the following function:

$$g(z) = \begin{cases} 0 & \text{if} z\geq 0 \\ -\gamma z & \text{if} z<0 \end{cases} \qquad (12)$$

where γ is positive and large enough (see [6] for the detail discussion about the choice of the parameter γ. In this case, S is a nonlinear negative resistor).

Under the setting of the parameters, the equations (1)-(3) are reduced to be

$$C_i\frac{du_i}{dt} = -\frac{u_i}{R_i} - A_i + \sum_{l=1}^{M} E_{il}\tilde{V}_l. \qquad (13)$$

$$I_k = \sum_{j=1}^{N} E_{jk}V_j - B_k = \mathbf{E}_k^T\mathbf{V} - B_k. \qquad (14)$$

$$E = \sum_{i=1}^{N} A_iV_i + \sum_{i=1}^{N} 1/R_i\int_0^{V_i} f^{-1}(v)dv - \sum_{l=1}^{M} G(I_l)$$
$$= \mathbf{A}^T\mathbf{V} + \sum_{i=1}^{N} 1/R_i\int_0^{V_i} f^{-1}(v)dv - \sum_{l=1}^{M} G(I_l) \quad (15)$$

Substituting (14) into (13), we have

$$C_i\frac{du_i}{dt} = -\frac{u_i}{R_i} - A_i + \sum_{l=1}^{M} E_{il}g(E_lV-B_l) \qquad (16)$$

Since the condition (i) of the theorem are held under the setting of the parameters, the equilibria of (16) are equivalent to the minima of the function (15). Then as long as we can show the equivalence between (15) and the linear programming (10) and (11), we also show the DS_net under these conditions can be used as a linear programming network. That is, a stable output of the network is a solution of the linear programming. Notice $G(I_l) = \int_0^{I_l} g(z)dz$. According to the definition of the function g, we have

$$G(I_l) = \begin{cases} 0 & \text{if} I_l\geq 0 \\ -\frac{\gamma}{2}I_l^2 & \text{if} I_l<0 \end{cases} \qquad (17)$$

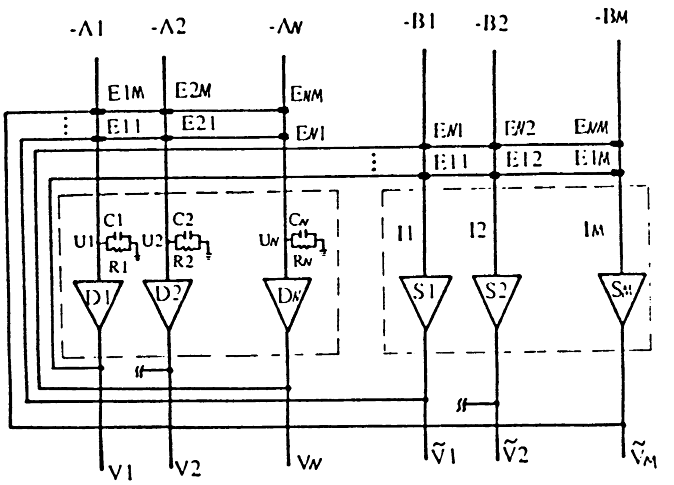


Fig.2 The circuitry of a linear programming neural network. Here **D** is a near-ideal amplifier, **S** is a nonlinear negative resistor defined by (12), which is different from that in [2].

Since γ is assumed to be large enough, I_l (l=1,2, ...,M) must not be less than zero in order to minimize the function E. This implies that the constraint (11) is satisfied. On the other hand, if the gain of the sigmoid curve f is high enough, the middle term on the right side of (15) can be neglected [1]. That is

$$\min \mathbf{E} = \min \mathbf{A}^T\mathbf{V}, \qquad (18)$$

subject to the constraint (11). The linear programming neural network is sketched in Fig.2 which is similar to that in [2]. The differences between the network presented here and that in [2] are the nonlinearity (12) and the matrices $\mathbf{H}^{(D)}$ and $\mathbf{H}^{(S)}$. In [2], $\gamma = 1$ and $\mathbf{H}^{(D)} = (\mathbf{H}^{(S)})^T = -\mathbf{E}$. Mathematically, $\gamma = 1$ is not sufficient.

4. Quadratic Programming

A quadratic programming problem can be described to attempt to minimize the following cost function

$$\phi(V) = A^T V + 1/2 V^T G V \qquad (19)$$

subject to

$$E^T V - B \geq 0. \qquad (20)$$

where G is a $N \times N$ symmetric matrix, and V, A, E, B have the similar meanings to those in the linear programming. In Fig.1, let

$$I_1^{(D)} = -A_1, \ldots, I_N^{(D)} = -A_N,\ I_1^{(S)} = -B_1, \ldots, I_M^{(S)} = -B_M$$

$$W_{ki} = 0,\quad (H_{ik}^{(D)})_{N\times M} = -(E_{ik})_{N\times M},$$

$$(T_{ij})_{N\times N} = -(G_{ij})_{N\times N}\quad (H_{ik}^{(S)})_{M\times N} = -(E_{ik})_{N\times M}^T.$$

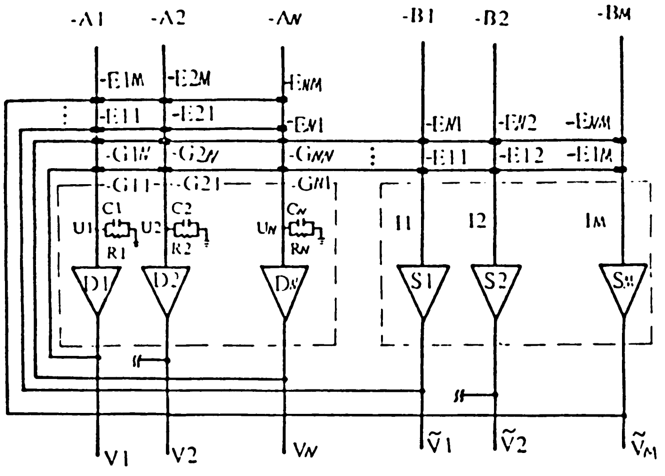


Fig.3 The circuitry of a quadratic programming neural network. The neurons D and S are as the same as those in Fig.2.

Further if the neuron D_i (i=1,2,...,N) is chosen to be an appropriate sigmoid curve, and the neuron S_k (k=1,2,...,M) is defined by (12). Then the equation (1)-(3) is reduced to be

$$C_i \frac{du_i}{dt} = -\frac{u_i}{R_i} - \sum_{j=1}^{N} G_{ij} V_j - A_i - \sum_{l=1}^{M} E_{il} \tilde{V}_l \qquad (21)$$

$$I_k = \sum_{j=1}^{N} E_{jk} V_j - B_k = E_k^T V - B_k. \qquad (22)$$

$$E = 1/2 \sum_{i=1}^{N}\sum_{j=1}^{N} G_{ij} V_i V_j + \sum_{i=1}^{N} A_i V_i + \sum_{i=1}^{N} 1/R_i \int_0^{V_i} f^{-1}(v)dv - \sum_{l=1}^{M} G(I_l)$$

$$= A^T V + 1/2 V^T G V + \sum_{i=1}^{N} 1/R_i \int_0^{V_i} f^{-1}(v)dv - \sum_{l=1}^{M} G(I_l) \qquad (23)$$

Since the connection matrix of T is symmetric, the matrix $H^{(D)}$ equals the transposition of the matrix $H^{(S)}$, and no connection from the dynamic subnetwork to the static subnetwork, the network will converge to one of the minima of the function (23) in terms of the theorem. On the other hand, with the same reason as in the linear programming case, the minimization of (23) is equivalent to the quadratic programming problem (19) and (20). Fig.3 shows the circuit implementation of a quadratic programming neural network.

Notice: (i) Because of the limitation of the space the simulation results are not presented here. (ii) The concept of FS_net can be applied to somewhere else. For instance, the fast subnetworks may be used to store different characters with the same codes so that there are not two different characters sharing a same code in the main subnetwork [3]; In the implementation of the dynamic tunneling algorithm [6], the minimization phase can be accomplished by the fast subnetwork, while the slow subnetwork can be used to assign new starting points to the fast subnetwork.

Acknowledgement: Support from grants MH06686 and AFOSR-87-0317 is gratefully acknowledged.

REFERENCES

[1] J.J. Hopfield "Neural networks and physical systems with emergent computational abilities" *Proc. Natl. Acad. Sci. USA Vol. 79 1982, pp 2554-2558.*

[2] D.W. Tank and J.J. Hopfield "Simple 'neural' optimization networks: an A/D converter, signal decision circuit, and a linear programming circuit" *IEEE Trans. CAS-33, No.5, 1986 pp.533-541.*

[3] Y. Yao "A neural network model of CAAM and its application to handprinted Chinese character recognition" *Oral presentation at the First IEEE Int. Conf. on Neural Networks, San Diego, 1987 (see the Proceedings, Vol.III-309-316).*

[4] H. Szu "Reconfigurable neural nets by energy convergence learning principle based on extended McCulloch-Pitts neurons and synapses" *Proc. of the First Int. Joint Conf. on Neural Networks, Washington D.C. 1989 Vol-I pp.485-496.*

[5] W.J. Freeman "Mass action in the nervous systems" *Academic Press, New York 1975.*

[6] Y. Yao "Dynamic tunneling algorithm for global optimization" *IEEE Trans. CMS, No.5, 1989 in press.*

Theories on the Hopfield Neural Networks with Inequality Constraints

Shigeo Abe   Junzo Kawakami

Hitachi Research Laboratory, Hitachi, Ltd.
4026 Kuji, Hitachi, Ibaraki, Japan

Introduction

The Hopfield neural networks [1], whose convergence characteristics have been clarified [2], are well suited to solving a combinatorial optimization problem, provided it has no inequality constraints.

In this paper, inequality constraints are introduced into the Hopfield model, and the convergence characteristics for the extended model are clarified. Then the algorithm to determine the weights in the energy function is discussed. Finally, convergence of an ill-conditioned problem such as a transportation problem is shown to be improved by the introduction of the inequality constraints. The results are also verified by computer simulations.

Problem Formulation

Let the inequality constraints be expressed by

$$d \geq \sum_{i=1}^{n} w_i x_i \quad \text{or} \quad \sum_{i=1}^{n} w_i x_i \geq d \qquad \text{where } d > 0,\ x_i = 0, 1. \tag{1}$$

By introducing variable y, the two equations in (1) become, respectively,

$$dy - \sum_{i=1}^{n} w_i x_i = 0,\quad 1 \geq y \geq 0 \quad \text{or}\ dy - \sum_{i=1}^{n} w_i x_i = 0,\quad y \geq 1. \tag{2}$$

Now a combinatorial optimization problem with inequality constraints can be formulated as minimizing the energy E' expressed by

$$E' = 1/2x^t T' x + b^t x \tag{3}$$

with the constraints:

$$d_i y_i - \sum_{i=1}^{n} w_{ij} x_j = 0,\quad 1 \geq y_i \geq 0,\ \text{or}\ y_i \geq 1 \qquad \text{for } i = 1, \ldots, k \tag{4}$$

where x, b are n-th variable and input vectors, respectively, and T' is an nxn symmetric coefficient matrix.

Adding the square of (4) divided by 2 to (3) gives

$$E = 1/2(x, y)^t \begin{pmatrix} T & V \\ V^t & D \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + b^t x \tag{5}$$

where y is a k-th vector, $T = T' + W$, and

$$W = \begin{pmatrix} \Sigma w_{i1}^2 & \cdots & \Sigma w_{in} w_{i1} \\ \vdots & & \vdots \\ \Sigma w_{i1} w_{in} & \cdots & \Sigma w_{in}^2 \end{pmatrix} \quad D = \begin{pmatrix} d_1^2 & & 0 \\ & \ddots & \\ 0 & & d_k^2 \end{pmatrix} \quad V = -\begin{pmatrix} d_1 w_{11} & \cdots & d_k w_{k1} \\ \vdots & & \vdots \\ d_1 w_{1n} & \cdots & d_k w_{kn} \end{pmatrix}$$

The Hopfield model for (5) is given by

$$x_i = 1/2(1 + \tanh u_i),\quad y_i = 1/2(1 + \tanh v_i) \text{ for } 1 \geq y_i \geq 0,\ 1 + \exp(v_i) \text{ for } y_i \geq 1 \tag{6}$$

$$\begin{pmatrix} du/dt \\ dv/dt \end{pmatrix} = -\begin{pmatrix} T & V \\ V^t & D \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} b \\ 0 \end{pmatrix} \tag{7}$$

Eigenvalue Analysis

Eliminating u and v from (6) and (7) gives

$$\begin{pmatrix} dx/dt \\ dy/dt \end{pmatrix} = -\begin{bmatrix} 2(1-x_1) & & & & 0 \\ & \ddots & & & \\ & & 2(1-x_n)x_n & & \\ & & & f(y_1) & \\ & & & & \ddots \\ 0 & & & & f(y_k) \end{bmatrix}\left\{\begin{bmatrix} T & V \\ V^t & D \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b \\ 0 \end{bmatrix}\right\} \tag{8}$$

where $f(y_i) = 2(1 - y_i)y_i$ for $1 \geq y_i \geq 0$, $(y_i - 1)$ for $y_i \geq 1$.

The solutions obtained by integrating (8) are singular points of (8) given by

(i) $x_i = 1, 0$, and $y_j = 1, 0$, or 1;

(ii) the solution of $Tx + Vy + b = 0$, and $V^tx + Dy = 0$; and

(iii) combination of (i) and (ii).

The singular point (ii) is given by

$y = -D^{-1}V^tx$, and $(T - VD^{-1}V^t)\, x + b = T'\, x + b = 0$.

Thus the following theorem holds:

Theorem 1 If, for singular point (ii), y satisfies $1 \geq y_i \geq 0$ or $y_i \geq 1$ for $1 \leq i \leq k$, x corresponds to the singular point without inequality constraints.

The desired solution is

$x_i = 1, 0 \quad i = 1,\dots,n, \quad 1 \geq y_j \geq 0$, or $y_j \geq 1$, $j = 1,\dots,k$.

To derive a linearized equation of (8) around the above solution, substitution of

$x = x' + c$, $c_i = 1, 0$, and $y = y' + e$, $1 \geq e_j \geq 0$ or $e_j \geq 1$

into (8) gives

$$dx'/dt = \begin{bmatrix} 2(2c_1 - 1)(T_1c + V_1\, e + b_1) & & 0 \\ & \ddots & \\ 0 & & 2(2c_n - 1)(T_n\, c + V_n\, e\, b_n) \end{bmatrix} x' \tag{9}$$

$$dy'/dt = -\begin{bmatrix} f(e_1)d_1^2 & & 0 \\ & \ddots & \\ 0 & & f(e_k)d_k^2 \end{bmatrix} y' - \begin{bmatrix} f(e_1) & & 0 \\ & \ddots & \\ 0 & & f(e_k) \end{bmatrix} V^tx' \tag{10}$$

Thus the eigenvalues concerning x are given by

$$\lambda_{c,i} = 2(2c_i - 1)(T_i\, c + V_i\, e + b_i) \qquad i = 1,\dots,n \tag{11}$$

Since the coefficients of y' are negative, the solution (c, e) is stable if

$$\lambda_{c,i} < 0 \qquad i = 1,\dots,n \tag{12}$$

holds. This means that y is a dependent variable vector and does not influence convergence of the solution. The following theorem can be easily derived:

Theorem 2 If vertex $c = (c_1,\dots,c_n)$ satisfies inequality constraints (2), the eigenvalues given by (11) are the same as those without inequality constraints. Namely:

$$\lambda_{c,i} = 2(2c_i - 1)(T'_i c + b_i) \quad i = 1,\dots,n \tag{13}$$

Let the solution that satisfies equality constraints be c and the solution $c(i)$ be the one with the i-th element of c converted from c_i to $1-c_i$. Then if all vertexes c, and $c(i)$ satisfy constraints all the theorems shown in [2] hold. Further the following theorem can be derived:

Theorem 3 Let $T'_{ii} = 0$, for $i = 1,\dots,n$ and vertex c satisfy inequality constraints. Then if, for energy E' given by (3),

$$E'_c < E'_{c(i)} \qquad \text{for } i = 1,\dots,n \tag{14}$$

holds, vertex c is stable.

Determination of Weights

Let the energy be given by

$$E = A\,f(x) + B\;g(x)^2 + C\;h(x,y)^2 \tag{15}$$

where $f(x)$ is an objective function, $g(x)$ are equality constraints, $h(x,y)$ are inequality constraints, and A, B, C are weights.

So long as the solution satisfies the inequality constraints, they do not influence the stability of the solution as seen from theorem 2. Thus, the method to determine weights A and B is the same as that without inequality constraints. Namely

(1) Convert x_i^2 in $f(x)$ and $g(x)^2$ into x_i. Let $f(x)$ and $g(x)^2$ be thus converted.

(2) Determine B so that

$$A\,f(c(i)) + B\;g(c(i))^2 > A\,f(c)$$

holds for arbitrary vertexes c and $c(i)$.

Weight C is determined so that the solutions that do not satisfy the inequality constraints are unstable.

Convergence Improvement of Transportation Problem

Introduction of inequality constraints can improve convergence for a problem which does not require inequality constraints. One example is a transportation problem in which a commodity must be transmitted from one point to another in a network with minimum distance.

The energy function of the transportation problem is given by

$$E = A/2 \sum_{i=1}^{n}\Big(I_i + \sum_{j\in N(i)}(b_{ji} - b_{ij})\Big)^2 + B/2\sum_{i=1}^{n}\;\sum_{j\in N(i)} b_{ij}b_{ji} + D/2 \sum_{i=1}^{n}\;\sum_{j\in N(i)} (b_{ij} + b_{ji})\,C_{ij} \tag{16}$$

where the first term specifies that the summation of the flows into and out of a node is zero, the second term inhibits simultaneous flows from i to j and j to i, the third term is the objective function and

I_i : injection to i-th node (1: start point, -1: end point , 0: all others), b_{ij}: flow from node i to j(1: with flow, 0: without flow), C_{ij}: cost from i to j, n: the number of nodes, and A, B, D: weights.

Since no inequality constraints exist, the problem can be solved by the Hopfield neural networks. Weights A, B, and D can be determined as follows: When the output of any one neuron is changed frcm 1 to 0 or 0 to 1 for any feasible solution, the energy for the first and the second term in (16) increases, respectively, by A and B. Thus let $A = B$. Then the minimum increment of the energy, undeı the above condition, for the first and second terms in (16) is A, and the maximum decrement of the third term is D max C_{ij}. Therefore, weights A and D be chosen so that

$$A > D \max C_{ij} \tag{17}$$

holds. For this problem, all the elements of input vector b in (3) are positive, if the adjacent nodes are not selected as the start and end nodes. This means that the origin 0 becomes a stable point. This can be avoided if the inequality constraint is introduced:

$$\sum_{i=1}^{n} \sum_{j\in N(i)} b_{ij} \geq 1 \tag{18}$$

thus the energy is $E' = E + F/2\,(y - \sum_{i=1}^{n} \sum_{j\in N(i)} b_{ij})^2, \; y \geq 1.$ (19)

According to the eigenvalue analysis the origin becomes unstable if

$$F > D \max C_{ij}/2 \tag{20}$$

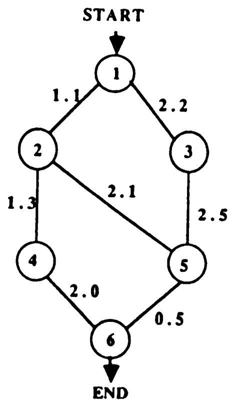


Fig. 1 A NETWORK MODEL

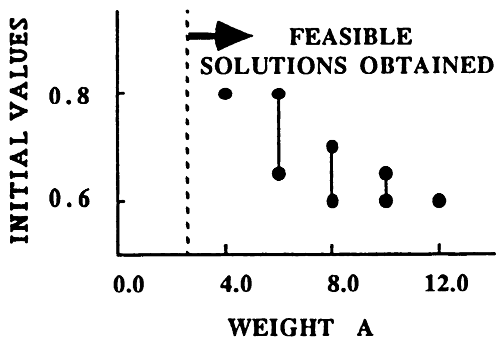


FIG. 2 CONVERGENCE TO OPTIMAL SOLUTION

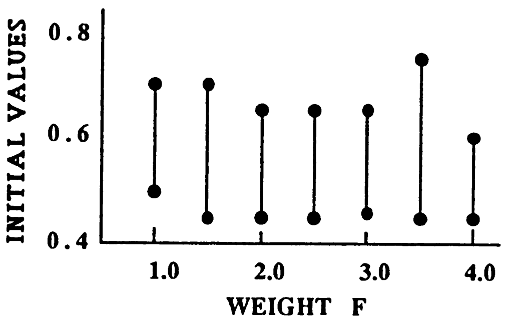


FIG. 3 CONVERGENCE IMPROVEMENT BY INEQUALITY CONSTRAINT(A = 3.0)

Consider the network model shown in Fig. 1. Assuming $D = 1$, weights A and F should be, respectively, larger than 2.5 and 1.25 from (17) and (20). Figure 2 shows the convergence to the optimal solution without the inequality constraint by varying all the initial values from 0.05 to 0.95 with the increment of 0.05. The convergence region is very small. When the initial values are selected as smaller than those for the optimal solution, the solution converges to the origin. Figure 3 shows the convergence to the optimal solution with the inequality constraint and $A = 3.0$. The convergence is improved drastically. Without the inequality constraint the optimal solution for $A = 3.0$ cannot be obtained.

Conclusion

The convergence characteristics of the Hopfield neural networks with inequality constraints were clarified. Determination of the weights in the energy function was derived. The result was tested for the transportation problem and significant convergence improvement was obtained.

References

[1] J. J. Hopfield and D. W. Tank, "Neural Computation of Decision in Optimizing Problems," Biological Cybernetics, 52, pp 141-152, 1985.

[2] S. Abe, "Theories on the Hopfield Neural Networks," Proc. IJCNN-89, pp I-557 - I-564, June 1989.

Adaptive Junction: A Spatio-Temporal Neuron Model

Yoshiaki Ajioka, Yuichiro Anzai and Hideo Aiso
Department of Electrical Engineering, Keio University
3-14-1 Hiyoshi, Kohokuku, Yokohama 223 JAPAN

ABSTRACT Adaptive Junction is a neural network that can discriminate simple spatio-temporal patterns. In this paper, we show how Adaptive Junction can discriminate signals moving along either vertical or horizontal direction in the two dimensional plane.

1. Introduction

In this paper, we describe a neural model called Adaptive Junction for discriminating simple spatio-temporal patterns moving along either vertical or horizontal direction in the two dimensional plane. We believe that construction of such models are among important steps towards realizing mental actions such as thinking, reasoning and association by neural networks.

2. Definition of spatio-temporal patterns

Spatio-temporal patterns are defined in the following way. Consider two lamps, lamp-A and lamp-B. Either lamp may be completely off (with an activation value of 0), maximally bright (an activation of 1), or at some intermediate brightness. The brightness of a lamp may be changed exponentially with a time constant τ.

Now, consider the two lamps to be at an intermediate brightness level with the value of 0.5. Next only lamp-A is maximally bright and lamp-B is turned off. This state (lamp-A at 1 and lamp-B at 0) remains for T seconds. Then, lamp-A is turned off and lamp-B is maximally bright, remaining so for T more seconds. This pattern of lights (a gradual change from both neutral, to only lamp-A bright, to only lamp-B bright) will be called the spatio-temporal pattern AB (or pattern-AB).

3. Network

In the above example, consider the case where we must distinguish four patterns (as defined above): pattern-AA, pattern-AB, pattern-BA and pattern-BB.

The network of Adaptive Junction has three layers: (1) the sensory layer (S-layer), (2) the response layer (R-layer), and (3) the teaching layer (T-layer). The S-layer consists of sensor neurons (S-neurons), which correspond to the lamps discussed above. In the current situation, the S-layer has two neurons, S_A and S_B. The R-layer consists of response neurons (R-neurons), which respond what the input pattern is. Thus, the R-layer needs response neurons R_{AA}, R_{AB}, R_{BA} and R_{BB}. For example, if the pattern in the S-layer is pattern-AB, only R_{AB} is excited while the other R-neurons are inhibited. Finally, the T-layer consists of teacher neurons (T-neurons), which supply teacher signals used for supervised learning.

The connections of the network are set up as follows: (see Figure 1)

- The output of each S-neuron is connected to any R-neurons.
- The output of each R-neuron is connected to any R-neurons, but there are no self-loops.
- Each R-neuron has a connection from its corresponding T-neuron.

Overall, the network does not have any hidden unit.

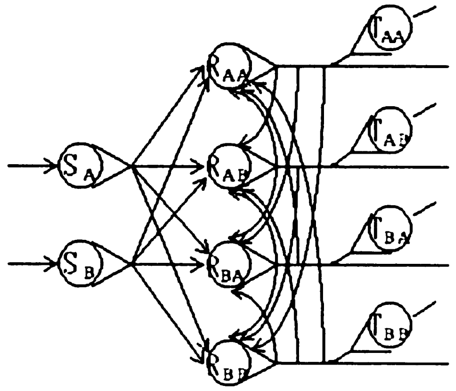


Fig.1 A network of Adaptive Junction.

4. Neuron model

The input-output relation of an R-neuron in our model is formulated as follows. For the R-neuron j, net_j and O_j are defined by the equations:

$$net_j(t) = \sum_i w_{ji} O_i(t) \tag{1}$$

$$\tau \frac{\partial O_j(t)}{\partial t} = -O_j(t) + f(net_j(t)) \tag{2}$$

where, w_{ji} is the weight from a neuron i to j, O_j represents the output of j, and f(·) is the sigmoid function.

In addition, S-neurons and T-neurons output the time-lag signal of the input.

5. Learning rule

The rule for updating the weight w_{ji} per unit time of the simulation ($t << \tau$) is defined by the following function:

$$\Delta w_{ji}(t) = \eta_w \varepsilon_j(t) f'(net_j(t)) O_i(t) \Delta t \tag{3}$$

where η_w is the learning rate, which is a small, positive constant, and ε_j is the estimation value, derived using the following equations:

$$\varepsilon_j(t) = \frac{T}{t}(T_j(t) - O_j(t)) + C_j(t) - f(net_j(t)) \tag{4}$$

$$C_j(t) = T_j(t) + \tau \frac{dT_j(t)}{dt} \tag{5}$$

where C_j is called the criterion value and is equal to the input of the T-neuron j. Finally, T_j represents the output of the T-neuron j.[1][4][5][6][8]

6. Simulation

Here, let us describe examples of simulations for the model, particularly which show how Adaptive Junction can discriminate signals moving along either vertical or horizontal direction of the two dimensional plane.

Suppose that nine S-neurons, named A to I, are laid out along two dimensions as shown in Figure 2. A signal is made to appear, and move, by firing only one S-neuron at a time, as shown in Figure 3. In all, there were twelve patterns used. A signal might move to left along one of three rows (e.g., pattern-IHG), from top to bottom along one of three columns (e.g., pattern-BEH), or from bottom to top along one of three columns (e.g., pattern-GDA). The task for Adaptive Junction is to discriminate four directions that the signal moved to. The simulation results for this problemare shown in Figure 4.

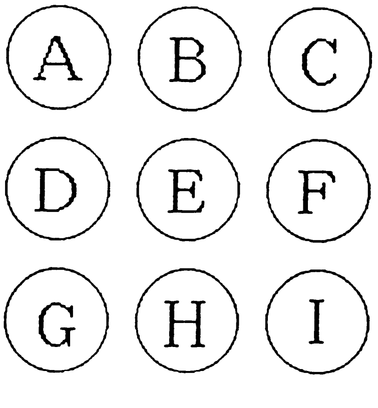


Fig. 2 S-neurons laid out along two dimensions.

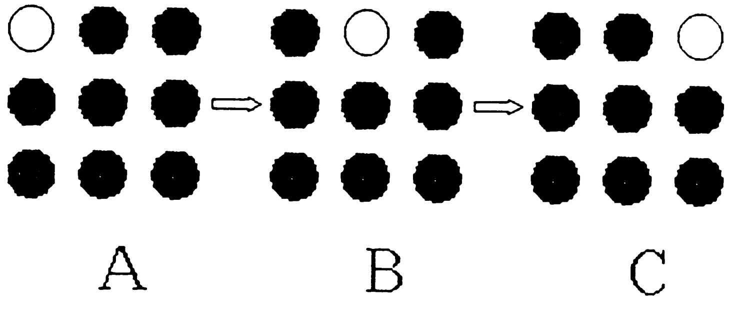


Fig. 3 Spatio-temporal pattern-ABC: A white circle corresponds to an exciting neuron, and a black circle to an inhibitory neuron.

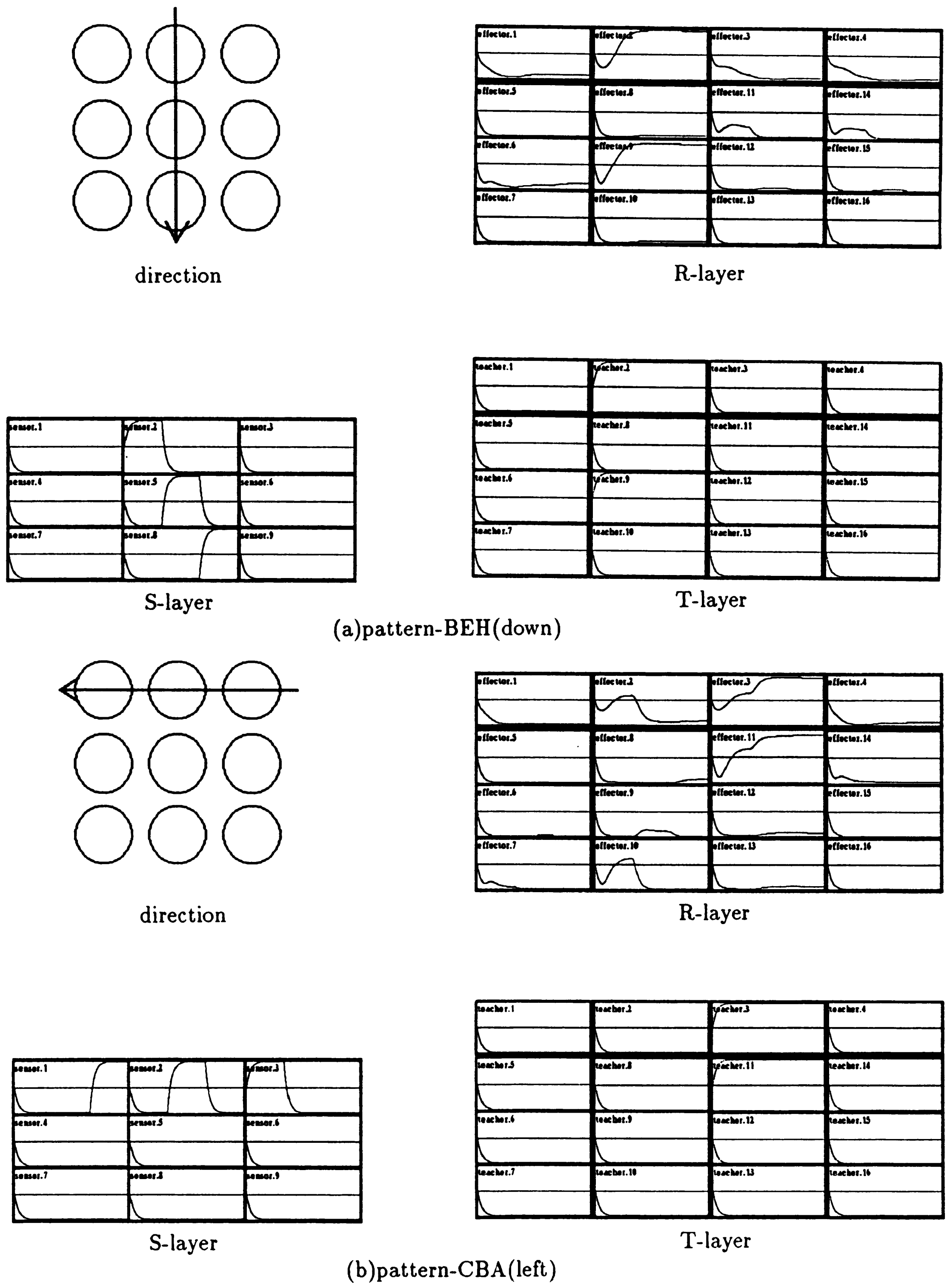


Fig. 4 The results of the discrimination of spatio-temporal patterns:Each graph shows the activation of a neuron changing with time. The first row of the graphs of R-layer and T-layer shows the directions, and others show the feature detections. The columns correspond to up, down, left and right directions, respectively.

7. Discussion

Adaptive Junction is successfully able to discriminate among the four directions created via the twelve patterns. However, during the simulations the following were noted:(a) Adaptive Junction can not discriminate the four directions using only four R-neurons (one corresponding to each of left-wards, right-wards, upwards, and downwards). Thus, Adaptive Junction needs neurons for "feature detection" in R-layer. (b) When the direction of the input pattern is obvious even though the full sequence of input patterns is not yet given, Adaptive Junction is able to decide on the correct direction.
These results support our previous hypothesis about the number of feature detection neurons and the behavior of Adaptive Junction. As such, by adding more S-neurons to Adaptive Junction, it may be able to discriminate the directions of a signal's movement.

8. Conclusion

In this paper, we presented a spatio-temporal neuron model, Adaptive Junction, and showed that it can discriminate the direction of a signal's temporal movement along one of two spatial dimensions. Although it is necessary tó add R-neurons, i.e., teacher signals, for feature detection, and to initialize all the neurons before learning and perception, we suggest the following:

1. Adaptive Junction runs on real time, not on step time.
2. The network and the learning rule have very simple structures.
3. Adaptive Junction needs only the same number of memory elements as neurons, different from various types of recurrent networks[9] etc.
4. If the input signals are fixed, the output signals are also fixed, that is, Adaptive Junction has stable states. [2][3][7]

References

[1] Y.Ajioka et al. :Neural network for the discrimination of time-series patterns -Adaptive Junction-, EIC Technical Report, MBE88, pp. 197-202, 1989

[2] Y.Ajioka,Y.Anzai:A study of supervised signals and the discrimination on Adaptive Junction, EIC Technical Report, PRU89, pp. 23-30, 1989

[3] Y.Ajioka,Y.Anzai,H.Aiso:Discrimination of two dimensional spatio-temporal pattern on Adaptive Junction, submitted to Japan Imaging Technology '89

[4] K.Doya and S.Yoshizawa:Memorizing Oscillatory Patterns in the Analog Neuron Network, IJCNN, Vol. 1, pp. 27-32, 1989

[5] M.Fujita:Adaptive Filter Model of the Cerebellum, Biological Cybernetics, pp. 195-206, 1982

[6] M.I.Jordan:Attracter dynamics and parallelism in a connectionist sequential machine, Proc. of 8th Annual Conference of the Cog. Sci. Soc. , pp. 531-546, 1987

[7] F.J.Pineda:Generalization of back-propagation to recurrent neural networks, Physical Review Letters, Vol. 59, pp. 2229-2232, 1987

[8] D.Rumelhart&L.McClelland:PARALELL DISTRIBUTED PROCESSING, Cambrige, MA, MIT Press, Vol. 1&2, 1986

[9] R.J.Williams and D.Zipser:A learning algorithm for continually running fully recurrent neural networks, Neural Computation, 1989

COMPETITIVE LEARNING WITH MODIFIABLE THRESHOLDS FOR VISUAL PATTERN RECOGNITION

Dimitrios Bairaktaris
University of St.Andrews - Computational Science -North Haugh - FIFE KY16 9SS- Scotland
TEL : (0334) 76161 ext. 8106 - e-mail : jim%uk.ac.st-and.cs

ABSTRACT
In this paper we describe a method for implementing connectionist theory on visual pattern recognition. Our basic connectionist model is based on a modified competitive learning weight modification rule. In this work we have usedthe modifiable threshold technique from our previous work on the Dynamic Performance Training Algorithm. The resulting basic network provides the required stability and it performs succesfully for a character recognition task. Finally, a method based on parallel scanning of a digital image along with the use of the basic network is discussed.

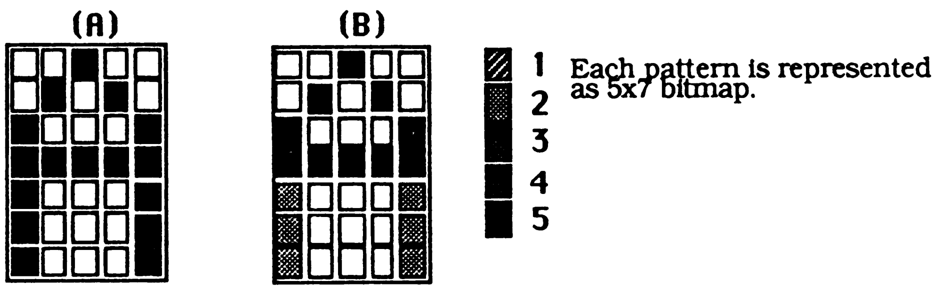
1. INTRODUCTION

Several attempts have been made to apply the powerful properties of parallelism and fault tolerance which neural networks demonstrate, for pattern recognition[1][2][3][4][8]. The neural network we consider here has a layered architecture and is ruled by the competitive learning (winner-take-all)[5]weight modification rule. We describe how this synaptic modification rule is applied for pattern recognition tasks and we demonstrate how information persistence problems appear when certain conditions of the training data are not met. Then, we borrow the modifiable threshold technique from the Dynamic Performance Training Algorithm[6] and we explain how information persistence problems are resolved. Having described the construction and training of a primary network we propose a method for recognizing overlapping patterns regardless of their position on an digital image(position invariant). The theoretical discussion of this work is coupled with simulation results. The set of patterns used in our simulations consist of the capital letters of the English alphabet represented as bitmap images(see figure 1).

2. COMPETITIVE LEARNING AND INFORMATION PERSISTENCE PROBLEMS

First let us give an outline of the set of patterns where the network will be applied. Assume a finite number of two-dimensional patterns each having a fixed size of NxM. Each pattern consists of on and off elements and it is translated into a K = NxM size vector of 1's and -1's when it is encoded in the network(figure 1(A)). Alternatively we can also include a density measure for the patterns where each element is allowed to vary within a range of possible density values(figure 1(B)). In this case the corresponding vector representation is no longer binary but consists of real numbers. The simulation results described in this paper were achieved using this second form of representation.

Figure 1.



Corresponding vectors

Char[A] = (-1,-1,1,-1,-1,-1,1,-1,1,-1,1,-1,-1,-1,1,1,1,1,1,1,1,-1,-1,-1,1,1,-1,-1,-1,1,1,-1,-1,-1,1)
Char[B] = (-1,-1,5,-1,-1,-1,5,-1,5,-1,5,-1,-1,-1,5,5,5,5,5,5,2,-1,-1,-1,2,2,-1,-1,-1,2,2,-1,-1,-1,2)

Consider a network whichconsists of two layers of units. The input units layer and the output units layer. Competitive learning networks support local knowledge representations . That is, for a number of patterns to be encoded in the network an equal number of units should be made available in the output layer. The number of units in the input layer depends on the size of the data. For every element of the vector(figure 1) an input unit should be made available. Every input unit is connected to every output unit with a link which has a weight value. The network is asked to store the patterns and then to be able to recall them. The storage of the patterns is achieved by the modification of the weights. The weights are modified according to the competitive learning scheme described in [6]. This training scheme is expected to assign one output unit to every pattern to be encoded. It is questionable here whether this encoding scheme is efficient. For example if we want to encode 2000 patterns then we have to use 2000 output units. It is not within our intention to discuss this issue any further, however it should be noted that distributed representations[7] would certainly require an even larger number of units for the same purpose. Assume that we completed the weight modification phase and we want to use the network. The following algorithm shows how this is done :

- Present the pattern to be recognized to the input units.
- Calculate the activity of every unit in the output layer as shown in (1).
- Find the winning unit among the units of the output layer.
- Calculate the activation of every input unit(i) as follows :

Activation of input unit (i) = activation of the winner-unit * weight of the connection between i and winner-unit.

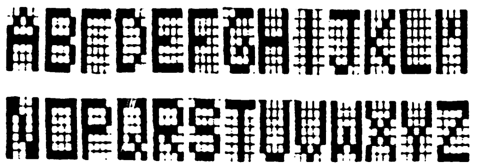
The activation of every input unit could now be translated back into a bitmap representation. Alternative uses of the network emerge from the fact that there is only one winner-unit for every pattern. One may want, after the winner is found, to propagate a signal to a next layer or through a mapping table to initiate a process. For example, in the case of alphabetical characters, we may assign the ascii value of the character to the corresponding winner-unit and create in this way an ascii file. The computation of the activation levels of each unit depends on locally available information and therefore the activation level of all the units is computed in parallel. In addition to parallelism the bias of a particular output unit to a particular pattern, expressed by the weights, allows the network to find the correct winner-unit even if the pattern presented has noise or it is corrupted.
The network functions properly under the assumption that every pattern is assigned to only one winner unit. It is simpleto prove that this is not always the case. Consider the following example. We have two patterns (1,1,1) and (1,-1,1) and a network with three input and two output units. Initially the weights are set to zero. We clamp the first vector at the input units and we calculate the activation of each output unit. Since the weights are zero, both output units have activation equal to zero. We choose the first unit to be the winner. We increase the weights on the connections between the winner-unit and the input units by 10. We clamp the second pattern (1,-1,1) at the input units and we calculate the activation of the output units. The first unit has an activation of 10 while the second has an activation of 0. According to the competitive learning synaptic modification scheme the winner-unit is the first output unit. The network failed to assign the second output unit to the second pattern. If we allow weight modification according to competitive learning information about the first pattern will be lost. In order to avoid this, an information persistence mechanism has to be found.
We introduce for every output unit a threshold value. Every time we modify the weights on a particular output unit we compute the activation level that the unit would have with the new weights and we assign the resulting value to the threshold. We augment the conditions under which a unit is a winner-unit by requiring its activation level to exceed or to be equal to its threshold. Consider the previous example. The activation level of the first unit after the modification of the weights, is 30. We set its threshold value to 30. When the second pattern is clamped at the input units, the first output units activation level is 10 but because it is under its threshold level it cannot be assigned as the winner. Therefore the second unit is assigned as the winner, since it meets the restrictions(initially threshold values are set to a negative value). It is obvious that setting the threshold value of a unit to its activation level reduces the fault tolerance properties of the network drastically. A unit will only be a winner if the com-

plete corresponding pattern is available. For this reason the threshold value should be a percentage of the activation value. Again consider the example we discuss. If the threshold value of the first unit is set to 20 information will still be encoded properly and will also allow 33% fault tolerance.

3. A NEURAL NETWORK MODEL FOR CHARACTER RECOGNITION

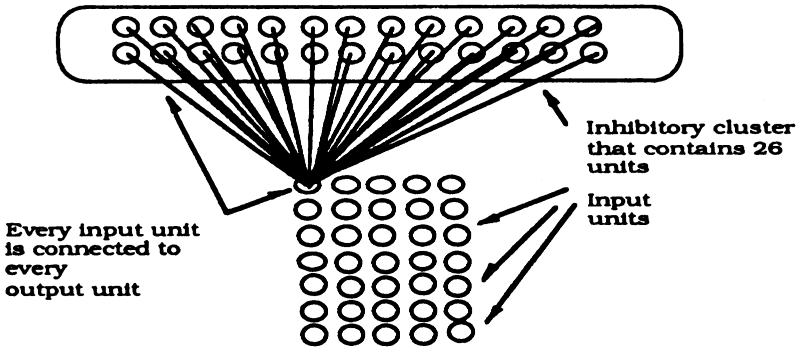
We choose our domain to be the set of patterns representing the capital characters of the English alphabet(figure 2).

Figure 2.



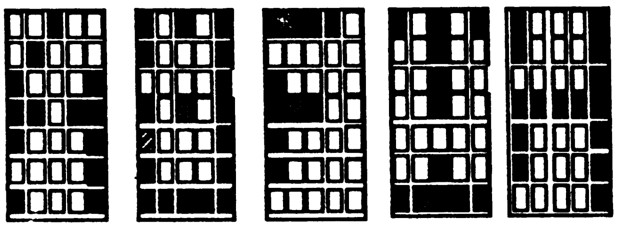
As we mentioned earlier we need an output unit for every pattern. Therefore the network has 26 output units. Each character is represented in a 5x7 grid. The size of the input vector is 35(5x7) and therefore the number of input units is 35. A full connectivity pattern is applied(figure 3).

Figure 3.


Inhibitory cluster that contains 26 units
Input units
Every input unit is connected to every output unit

During the weight modification phase each pattern was presented to the input units. For every pattern an output unit was assigned as a winner and the weights were modified as described earlier(2.) together with the provision of setting the threshold value of every output unit appropriately. Threshold values were set to 80% of the activation value of every output unit. This means that during evaluation noise up to 20% will be eliminated by the network. Further refinements on the threshold values of specific units would allow greater fault-tolerance for specific patterns. We tested the network with several noisy patterns and the network responded with the correct pattern in all of the cases where the error was less than 20%. Some examples of the patterns we tried are shown in figure 4.

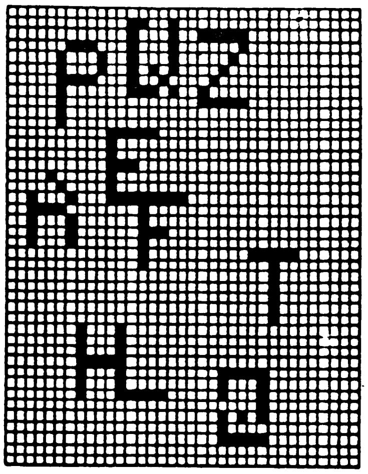
Figure 4.



4. RECOGNIZING OVELLAPING PATTERNS ON A DIGITAL IMAGE

So far in our discussion, we constructed a basic network that recognizes the patterns presented to it during training. Unfortunately, the network is only able to recognize a character when it is presented in the specified format at the input units. S.Grossberg and G.A.Carpenter in their work on Adaptive Resonance Theory (ART)[8] describe a three stage preprocessor that transforms the input data into the required form for the final network ART2[8] to recognize them. The basic network we described can be used in the same way ART2 is used at the top of the preprocessor The first stage of the preprocessor is responsible for extracting the pattern from the image. In the case were two patterns overlap each other extraction of one pattern, although it is desirable, it is not possible. In order to solve this problem we take a rather different approach where scanning of the image is performed in parallel. The method requires a fine-grain parallel architecture computing device in order to perform efficiently. Assume a digital image of size NxK pixels. Also suppose that the size of our original patterns is LxM. Assume that we have trained the appropriate network to recognize the patterns. We assume that the patterns in the image are of size LxM. The algorithm is based on the idea of examining every possible area of the image where a pattern could exist. Every search area has size LxM. The total number of possible search areas is (N-L+1) x (K-M +1). Using a copy of the basic network described in (3.) for every search area the computation could be done in parallel for all the search areas. Given (N-L+1) x (K-M +1) computing agents, the computation is performed in one machine cycle. Furthermore, the method takes full advantage of any degree of parallelism available. The main advantage of the method is the ability of recognizing patterns even when they overlap each other. Figure 5 is one of the test images we used for testing the algorithm. The example given in figure 5 has some noisy characters like A,Q, and O and overlapping characters like H and L. The method we described together with the basic network did recognize all the characters of the image.

<u>Figure 5.</u>



6. FURTHER ADVANCES

This method requires that the characters on the image have to be of the same size as the characters upon which the basic network was trained as well as that they have to be in the upright position. These are severe limitation towards the implementation of the model for a real-world pattern recognition task. Our current research is concentrated on these problems.

REFERENCIES

[1] Aleksander Igor (1986). *Adaptive pattern recognition systems and Boltzmann machines : A rapprochement.* Pattern Recognition Letters, 6, 113 - 120.

[2] Longstaff I.D. , Cross J.F. (1987). *A pattern recognition approach to understanding the multi-layer perceptron.* Pattern Recognition Letters, 5, 315 - 319.

[3] Fukushima K., Miyake S., Ito T. (1983). *Neocognitron : A Neural Network Model for a Mechanism of Visual Pattern Recognition.* IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-13, No5, 826 - 834.

[4] Bairaktaris D. (1989). *A Connectionist Model for Pattern Recognition.* To be appeared at Parallel Computing '89, Proceedings, Amsterdam : Elsevier.

[5] Rumelhart D.E., Zisper D. (1986). Feature Discovery by Competitive Learning. In Rumelhart, McClelland and the Parallel Distributed Group , *Parallel Distributed Processing Processing , Volume 1 : Foundations*, pp. 151-193. A Bradford Book, MIT Press.

[6] Bairaktaris D. (1989). *Dynamically Performing Neural Networks A Winner-Takes-All Model.* Technical Report, University of St.Andrews, Computational Science (also submitted to Neural Networks).

[7] Hopfield, J.J. (1982). *Neural Networks and Physical Systems with emergent collective computational abilities.* Proceedings of the National Academy of Sciences of the United States of America, 79, 2554-2558.

[8] Carpenter G.A., Grossberg S. (1988). The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network. IEEE COMPUTER, March 1988, pp 77-88.

A SPATIO-TEMPORAL NOVELTY DETECTOR USING FRACTAL CHAOS MODEL

J.M. BERTILLE and J.C. PEREZ
IBM France
Function 1510
B.P. 1021
34006 MONTPELLIER Cedex
FRANCE

I. INTRODUCTION

A major difference between humans and machines is the capabilities of the former to naturally store a great amount of knowledge, always immediatly at hand, and to use it to detect consciously novelty emergence in a great variety of situations.

In this paper we shall first recall the Fractal Chaos model previously presented in the INNS 1988 Boston Conference /BER88a/. Then we shall explain the use of this model as a spatio-temporal novelty detector. To illustrate this property we shall finally present an application: loop detection in dynamical signal analysis (analog signals).

II. THE FRACTAL CHAOS MODEL

The Fractal Chaos model is a dynamical system with a discrete time, discrete space and continuous states. It is a nonlinear model, which consists of succesive parallel procedures on a network. In the present paper we will use the following diffusively coupled model:

$$x_{n+1}(i,j) = (1-e)f(x_n(i,j)) + e/4\,(f(x_n(i-1,j)) + f(x_n(i+1,j)) + f(x_n(i,j-1)) + f(x_n(i,j+1))) \qquad (1)$$

where n is a discrete time step
(i,j) represents a network site (i,j=1,...,N=system size)
and e is the coupling strength within the range (0,1).

The function f(x) is a nonlinear mapping, the logistic map, that is:

$$f(x_n) = 1 - a_n x_n^2 \qquad (2)$$

where x_n represents the state of the function at time n within the range (-1,1), and a_n is the nonlinearity parameter of the function at time n within the range (0,2). The evolution of a is governed by the following equation:

$$a_{n+1}(i,j) = a_n(i,j) + c * LD(i,j) \qquad (3)$$

where c represents the updating rate of the array a and LD(i,j) is the local difference between the state of site (i,j) and the states average of its four neighbours.

Here we apply a two-dimensionnal periodic boundary condition on the network, that is, each site communicates via a Von-Neuman neighbourhood. Thus, the network (an N*N array) can be viewed as a torus.

This model and its properties have been investigated elsewhere (/PER88/,/PER89/,/BER88b/ and /BER89/). Here we shall only summarize the main features it provides:

(a) The use of a highly nonlinear transformation realizes an elementary component with an infinity of possible responses depending on the disturbances the site receives from the external environment. In the case of the logistic map, the fractal may reach a single attractor which can be a fixed point, a limit cycle or a chaotic attractor depending on the value of the nonlinearity parameter a.

(b) The use of a recursive function provides a component capable of integrating the history of its coupling with the external environment.

(c) The coupling of several maps gives to the system an exponentially increasing number of attractors depending on the nonlinearity parameter set a, the coupling strength e and the size N of the network. These attractors are all potentially reachable. The choice of a particular attractor lies on the coupling history of the system, that is, on the "inputs" or disturbances we have submitted it to. Note that the attractor depends on the initial conditions but also on the particular system context. Further analysis have been done showing that these attractors may be hierarchically structured /KAN89/.

In fact, the points (a) and (b) shows that the system dynamics can be viewed as a model of autonomous behaviour /VAR80/. All these points lead to a system that has the capability of compressing a great amount of information while creating its own internal dynamical representations (unsupervised learning) summarizing the spatial distribution (communication process) and the temporal organization (context dependance) of the external disturbances.

This system can just be viewed as an illustration of the high dimensionnal chaotic phenomena properties:

- data compression and history integration
- great selective power essentially due to the property of sensitive dependance on initial conditions
- spontaneous information and order generation.

Now let us focus on the problem of novelty detection in dynamical signal processing.

III. THE NOVELTY DETECTOR

Here, we will introduce the problem of novelty detection by considering a very simple example (Fig 1).

Consider a 3*3 array of sensors observing a process and providing our net with analog values within the range of (0,1). At each discrete time step these sensors will send disturbances to our system. In this example we have N=3, the sensors correspond to the inputs x given to the network. During an a priori unknown time interval the successive informations provided by the sensors are not related with each other. Suddenly, the process becomes cyclic with an a priori unknown period T (Fig 2). The problem consists then, to analyse the behaviour of the process via the sensors informations and to detect in real time the cycle apparition in order to advise the process supervisor.

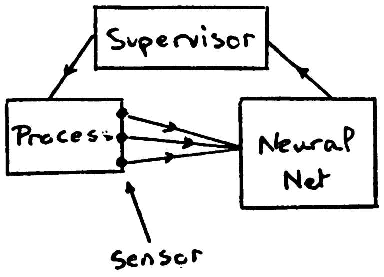
The system compresses the sensors informations in its nonlinearity array a (Fig 3) and then acts just as a recall memory ("déjà vu"), that is:

If the external disturbances have been already sufficiently seen by the system, that is, no novelty appears in these informations, the system then has nothing to learn and the a array remains constant.

On the other hand, if these disturbances present some novelty according to the system past then the latter one has to adapt itself to this new configuration. This adaptation involves the modifying of the a array which becomes unstable.

In fact, all the supervisor has to do is to observe the net behaviour. When this behaviour becomes stable (a array), it has to determine more precisely which attractors path the system passes through in order to check what is going on and to take the appropriate decisions. The figure 4 shows an example related to the detection of a cycle with period 72.

Figure 1



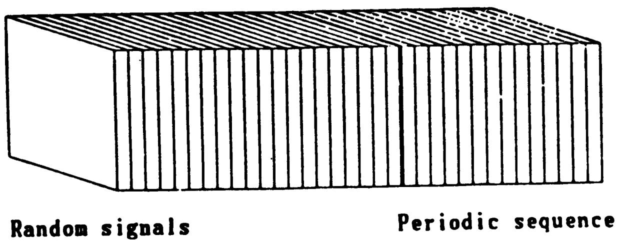


Figure 2

Figure 4

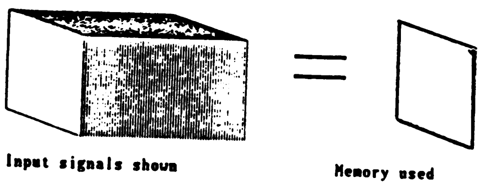
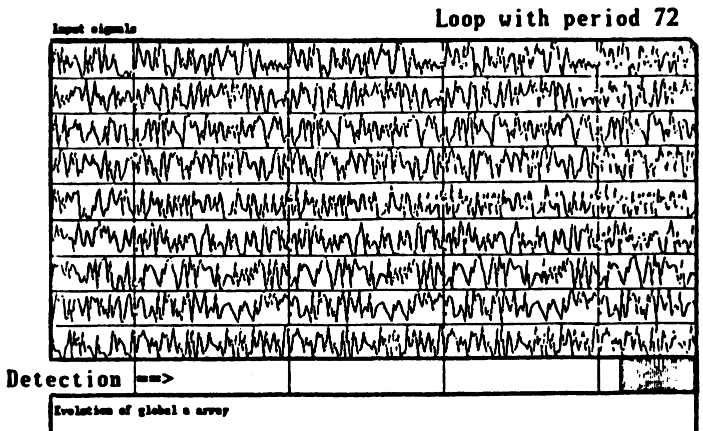


Figure 3



Loop detection with unpredictable period

IV. CONCLUSION

In this paper, we have presented a simple novelty detection application dealing only with loops involving the whole set of sensors. We think that this kind of model can also deal with much more complex structures such as embedded sequences, sequences involving subsets of sensors...

This could be an issue to by-pass the very well known problem of Artificial Intelligence : the way to handle embedded levels of knowledge (meta-knowledge) /HOF80/. We have studied the model capabilities to dynamically generate hierarchically organized attractors, and our conviction is that these hierarchical structures are closely related to the system history so that with finer observation tools we could detect much finer organization among the environment disturbances.

Another emergent feature appears within the model's nonlinearity array (a array). This internal memory like component shares some interesting properties with optical holograms. This array is created via the competition of two antagonist processes : an autonomous chaotic process in each fractal (data compression in the time dimension) and an autonomous communication process involving the whole network (data delocalization in the space dimension). So in fact, as in a hologram, we mix together on the same support spatio-temporal information. As a result, each site of the network integrates a kind of summary of the whole system dynamics.

Fractal Chaos model offers a lot of interesting perspectives. Actually we focus our reseach on the embedded levels of knowledge in order to realize the coupling of Fractal Chaos with an Expert System, and later to realize an Informal Expert System acquiring its knowledge from experience based learning.

V. REFERENCES

/BER88a/ J.M. Bertille and J.C. Perez : "Fractal Chaos: A new neural network holographic model", INNS 1988 Boston Conference, "Neural Networks", PERGAMON JOURNALS

/PER88/ J.C. Perez : "News ways towards Artificial Intelligence: Pluri-discilinarity, Self-organization and Neural Networks", MASSON 1988 Paris (french)

/PER89/ J.C. Perez : "The New Cybernetics of Fractal Holograms", MASSON Paris, to be published (french)

/BER88b/ J.M. Bertille and J.C. Perez : "The Fractal Chaos Neural Holographic Model: Theoric Backgrounds and Industrial Applications", NEURO-NIMES Conference 1988, EC2 Paris, (french)

/BER89/ J.M. Bertille and J.C. Perez : "A new neural network family: the Fractal Chaos holographic model and its industrial applications", CONVENTION IA Conference 1989, HERMES Paris, (french)

/KAN89/ K. Kaneko : "Clustering, Coding, Switching, Hierarchical Ordering, and Control in Network of Chaotic Elements", to be published

/VAR80/ F. Varela and H. Maturana : "Autopoiesis and Cognition", BOSTON, D. REIDEL, 1980

/HOF80/ Douglas R. Hofstadter : "Godel, Escher, Bach: An external golden braid", Ed VINTAGE Books, New-York, 1980

Additive Automata and Associative Memories *

M. Ceccarelli, A. Petrosino
Centro di Studio sui Calcolatori Ibridi, CNR
Via Claudio, 21
I-80125 Napoli, ITALY

R. Tagliaferri
Dipartimento di Informatica ed Applicazioni
Università degli Studi di Salerno
I-84081 Baronissi (Salerno), ITALY

SUMMARY

The aim of this paper is to analyze the dynamics of a class of boolean nets, Additive Automata [3], on the basis of laws of the theory of Cellular Automata [8,11], and present a learning rule for such class to accomplish Associative Memories. We shall refer to the deterministic and discret model of neural network presented by Caianiello in 1961, [1], which has been adapted to accomplish Associative Memories [2,4,5]. In particular, in the model under consideration the function computed by each neuron can been seen as the direct correlation of some characteristic stimuli given in input. In formula

$$\xi_i^{t+\tau} = \prod_{k \in K_i} \xi_k^t \tag{1}$$

where $\xi_i^t \in \{-1, 1\}$ is the state of the $i-th$ unit at the time t, τ is the synaptic delay and $K_i \subseteq \{1, \ldots, N\}$. The dynamics of this class is similar to that of Additive Cellular Automata. In fact after a simple transformation (1) becomes

$$x_i^{t+\tau} \equiv_2 \sum_{k \in K_i} x_k^t \tag{2}$$

* This work was supported in part by CNR, Progetto Finalizzato "Sistemi Informatici e Calcolo Parallelo", by MPI 40 % and by IIASS.

where $x_i^t \in \{0,1\}$, $1 \leq i \leq N$, which written in vectorial form is

$$\underline{x}^{t+\tau} \equiv_2 A\underline{x}^t \quad \underline{x}^t \in \{0,1\}^N \tag{3}$$

where A is a matrix with $a_{i,j} = 1$ if $j \in K_i$, $a_{i,j} = 0$ otherwise. This formulation yields a very simple solution of the "Inverse Problem", i.e. given a *priori* a set of state transitions of type (3), to determine the additive automaton which accomplishes them. It can be obtained as follows:

$$A \equiv_2 M'M^{-1} \tag{4}$$

where M is an $N \times N$ matrix whose columns are linearly independent states, M' is the next–state matrix, and the inverse of M is calculated in algebra modulo 2.

The main feature of an iterative dynamical system as that described above is the existence of transient and cyclic states, which are easily outlined from the permutation matrix P_N such that

$$\phi' = \phi P_N$$

where $\phi_{k,h} = \lfloor \frac{h}{2^k} \rfloor \ mod \ 2, \ \ 0 \leq k \leq N-1, \ \ 0 \leq h \leq 2^N - 1$, ϕ contains all the N-component binary states and ϕ' is the next-state matrix. It has been shown that the characteristic equation of P_N is given by

$$\lambda^{n_0} \prod_i (\lambda^{l_i} - 1)^{n_i} = 0 \tag{6}$$

where n_i, $i \geq 1$, is the number of cycles of length l_i and n_0 the number of transient states. Let be $r = rank(A)$ and m the least integer such that $rank(A^m) = rank(A^{m+1})$, we have the following properties:

i) the number of transient states is

$$n_0 = 2^N - 2^{mr-(m-1)N};$$

ii) any state reaches a cycle in at most m steps;

iii) the attraction basin of the $\underline{0}$ state has cardinality

$$\frac{2^N}{2^{mr-(m-1)N}} = 2^{m(N-r)};$$

iv) the cardinality of each attraction basin is a multiple of $2^{m(N-r)}$;

v) the cyclic lengths divide $k = max\{i : n_i \neq 0\}$;

A study in the case $l_i = 1$ has been developed to accomplish Instantaneous Associative Memories (I.A.M.), defined as dynamical systems such that

$$P_N = P_N^2$$

For the class under consideration the following holds:

Theorem

A necessary and sufficient condition to obtain an I.A.M. is that $r = N - v$, with $r = rank(A)$ and $v = rank(A + I)$.

For these systems we have determined a learning rule which permits the net to storage an input state $\underline{x}^{(i)}$. It looks as, set $a_{i,j}^{(0)} = 0$ and $\underline{r}_j(A)$ the $j-th$ row of matrix A:

$$A^{(i)} = A^{(i-1)} + (A^{(i-1)}\underline{x}^{(i)} + \underline{x}^{(i)})(\underline{\rho}^{(i)} + \underline{r}_j(A^{(i-1)}))^T \tag{7}$$

where the operations are intended modulo 2 and, set $\underline{z}^{(i)} = A^{(i-1)}\underline{x}^{(i)} + \underline{x}^{(i)}$, $\underline{\rho}^{(i)}$ presents only one non-vanishing element in the $j-th$ position such that $z_j \neq 0$ for $j \notin K$, K being the set of indices of the components choosen as characteristic stimuli in the input pattern $\underline{x}^{(i)}$. Some results are shown to validate the correcteness of this rule. This learning rule permits the storage N states or their associations with only N^2 bits. Furthermore the matrices $A^{(i)}$ are processed to obtain structured attraction basins.

The last section is dedicated to the interesting case $\xi_k^{t+\tau} = \pm\xi_j^t$, for some fixed j. The rule for this class is as follows:

$$A^{(1)} = \underline{\xi}^{(1)}\underline{v}^{(1)^T} \tag{8}$$

$$A^{(i)} = A^{(i-1)} + \sum_j \frac{1}{2}(\underline{\xi}^{(i)} -_{k_j} A^{(i-1)}\underline{\xi}^{(i)})\underline{v}_j^{(i)T} \qquad (i \geq 2)$$

where $-_{k_j}$ is a predefined operator. The computational complexity of (8) is linear in the number of neurons in the most significant cases, [8], using a suitable representation of matrix A called "columnar".

Finally, the paper opens the way to a study of applications of these rules, e.g. in the field of Pattern Recognition.

REFERENCES

[1] Caianiello E. R., "Outline of a Theory of Thought Processes and Thinking Machines", *J. Theor. Biological*, 1, 209, 1961.

[2] Caianiello E. R., "Neuronic Equations Revisited and Completely Solved", in *Brain Theory*, Springer-Verlag, 1986.

[3] Caianiello E. R., Marinaro M., "Linearization and Synthesis of Cellular Automata: The Additive Case", *Physica Scripta*, Vol. 34, 1986.

[4] Caianiello E. R., Marinaro M., Tagliaferri R., "Associative Memories as Neural Networks", in *Cybernetics and Systems*, Kluwer Academic Publishers, 1988.

[5] Caianiello E. R., Marinaro M., Tagliaferri R., "The Inverse problem for Linear Boolean Nets", in *Neural Computers*, Springer-Verlag, 1988.

[6] Ceccarelli M., Petrosino A., Tagliaferri R., "Convergence and Processing in Learning for Neural Nets. The AAM Model", Proceedings of *Second Italian Workshop on Neural Networks '89*, World Scientific, 1989.

[7] De Benedictis A., Tagliaferri R., "Self-associative neural nets: some examples and properties", Internal Report, 1989.

[8] Guan P., He Y., "Exact Results for Deterministic Cellular Automata", *Journ. of Statistical Physics*, 43, 463, 1986.

[9] Petrosino A., Savastano F., Tagliaferri R., "A Learning Rule for a Class of Linear Neural Nets", submitted to EURASIP, Lisbon (Portugal), 1989.

[10] Widrow G., Hoff M. E., "Adaptive switching circuits", IRE Wescon Convention Records, Part 4, pp. 96-104, 1960.

[11] Wolfram S. ed., "Theory and Applications of Cellular Automata", *Physica*, 10D, 1984.

On the training of a multi-layered Neural Net

C. B. CHITTINENI
Du Pont Company
600 Eagle Run Road
Newark, DE 19702

Abstract

An iterative algorithm for the training of a multi-layered Neural Net is presented in this paper. The algorithm is based on Taylor's series expansion and the prinicple of minimum disturbance. Conditions under which the algorithm converges are derived. Furthermore, the use of apriori information and handling of noisy observations are discussed.

1. Introduction

In recent years there has been increasing interest in the problem of learning a general input-output relation using layered Neural Net. Iterative algorithms with error back propagation based on mean squared error minimization are proposed [1-3]. For some special classes of nets such as Boltzman Machines [4], some theoretical results have been obtained for a qualitative understanding of their long run behaviour. Very few general results have been obtained on the convergence properties of the training algorithms. The objective of this paper is to present a training algorithm for multi-layered Neural Net and study its convergence properties. Also discuss approaches for taking into account apriori information and handling of noisy observations.

2. The training algorithm

In this section, after presenting the basic network equations, we derive a training algorithm and analyze its convergence properties.

2.1. Basic Network equations

A Neural Net with a hidden layer is shown in figure 1.

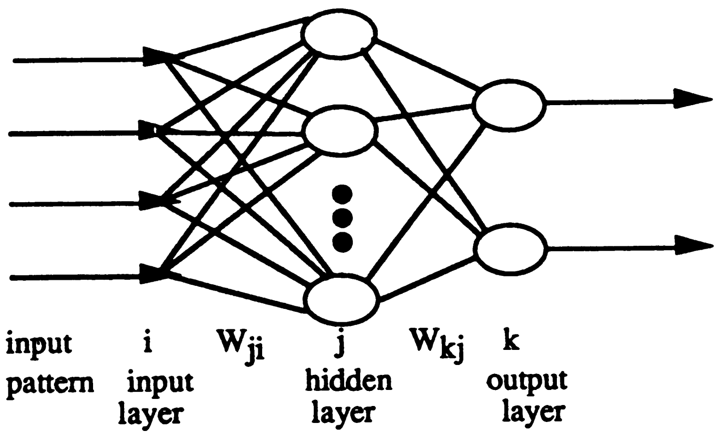


Figure 1. A layered Neural Net

The basic equations describing the net are

$$Net_j = \Sigma W_{ji} O_i \text{ and } O_j = f(Net_j) \quad (2.1)$$

Where W_{ji} are weights and O_i are responses at input-layer.
Similarly for the output layer k,

$$Net_k = \Sigma W_{kj} O_j \text{ and } O_k = f(Net_k) \quad (2.2)$$

2.2 The training algorithm and its convergence properties

The weights are adjusted iteratively so that the net responses will be equal to the desired responses. At iteration r, the corrections ΔW^r_{kj} to the weights W^r_{kj} are chosen so that the desired responses d_k will be equal to the estimated responses. Using the principle of minimum disturbance, the correction terms are obtained as

$$\Delta W^r_{k.} = (\Delta d^r_k \frac{\partial O^r_k}{\partial W^r_{k.}}) / (\frac{\partial O^{rT}_k}{\partial W^r_{k.}} \frac{\partial O^r_k}{\partial W^r_{k.}}) \quad (2.3)$$

Where $W^r_{k.}$ is a weight vector, $\Delta d^r_k = d_k - O_k(W^r_{k.})$, and

$\frac{\partial O_k}{\partial W^r_{k.}} = (\frac{\partial O_k}{\partial W^r_{k1}} \frac{\partial O_k}{\partial W^r_{k2}} \dots \frac{\partial O_k}{\partial W^r_{kJ}})^T$. The effective correction weight vector at iteration r is computed as an average of correction weight vectors for all the training patterns.
Treating the net responses as a function of $\{W^r_{ji}\}$, we obtain

$$\Delta d^r_k = \sum_j \sum_i O_i W_{kj} f'(Net_j) f'(Net_k) \Delta W^r_{ji} \quad (2.4)$$

Using the principle of minimum disturbance, the correction terms ΔW^r_{ji} can be shown to be

$$\Delta W^r_{ji} = \Delta d^r \alpha^r_{ji} / (\sum_j \sum_i \alpha^{r2}_{ji}) \quad (2.5)$$

Where $\Delta d^r = \sum_k \Delta d^r_k$, $\alpha^r_{ji} = O_i f'(Net_j) \sum_k W_{kj} f'(Net_k)$ and $\Delta d^r_k = \sum_i \sum_j O_i W_{kj} f'(Net_j) f'(Net_k)$ ΔW^r_{ji}. The individual pattern based corrections are combined to yield a overall correction term. At r th iteration, for the jth component of weight vector, equations (2.3) and (2.5) are of general form

$$W^{r+1}_j = W^r_j + S_j \Sigma_i [\beta^r_{ji} \Delta d^r_i / (\sum_\vartheta \beta^{r2}_{\vartheta i})] \quad (2.6)$$

Where S_j is the step size and $\Delta d^r_i = d_i - \Sigma_\vartheta \beta^r_{\vartheta i} W^r_\vartheta$. At the convergence of (2.6), we have

$$(D^{\frac{1}{2}} \beta^{rT})^T (d^{\frac{1}{2}} \beta^{rT}) W = (D^{\frac{1}{2}} \beta^{rT})^T D^{\frac{1}{2}} d \quad (2.7)$$

Where D^r is a diagonal matrix with diagonal elements $D^r_{jj} = 1 / \sum_l \beta^{r2}_{lj}$, W^r is a weight vector and β^r is a matrix. If $[(D^{\frac{1}{2}}\beta^{rT})^T (D^{\frac{1}{2}}\beta^{rT})]$ has full rank then the weight vector W converges to a unique solution. But that solution depends on the initial weight vector. In general, this will more likely be the case if the number of patterns is far greater than the number of pattern components. Let S be a diagonal matrix. We can write

$$W^{r+1} = W^r + S\beta^r D^r e^r \quad (2.8)$$

Let the diagonal elements of matrix E^r contains the eigenvalues λ^r_l and let the columns of matrix T^r contains the corresponding eigenvectors of $S^{\frac{1}{2}} \beta^r D^r \beta^{rT} S^{\frac{1}{2}}$. Then (2.8) reduces to

$$U^{r+1} = (I - E^r)\, U^r + T^{rT} S^{\frac{1}{2}} \beta^r D^{r\frac{1}{2}} D^{r\frac{1}{2}} d \quad (2.9)$$

Interms of the components of vectors U^r, the recursions are decoupled and are split into one dimensional recursions. Let the first m eigenvalues be non zero and let the remaining eigenvalues be zero. Then we can write

$$U^{r+1}_{\vartheta} = (1 - \lambda^r_{\vartheta})\, U^r_{\vartheta} + c^r_{\vartheta} \text{ for } 1 \le \vartheta \le m \text{ and } U^{r+1}_{\vartheta} = U^r_{\vartheta} \text{ for } \vartheta > m \quad (2.10)$$

Where c^r_{ϑ} is the ϑ th component of $(T^{rT} S^{\frac{1}{2}} \beta^r D^{r\frac{1}{2}} D^{r\frac{1}{2}} d$. The above equation can be expressed interms of initial guess as

$$U^{r+1}_l = [\prod_{s=0}^{r} (1 - \lambda^s_l)]\, U^o_l + \sum_{t=0}^{r-1} c^t_l [\prod_{s=t+1}^{r} (1 - \lambda^s_l] + c^r_l \quad (2.11)$$

hence, for convergence the eigenvalues should lie in the range $o \le \lambda_l \le 2$.

2.3 Use of apriori information

The apriori information about the Solution vector in solving the estimation problem can be taken into account as follows. Suppose apriori we know that the solution weight vector W is close to vector W_o and the covariance of W is given by C_o. Now the problem can be formulated as finding W by minimizing

$(W - W_o)^T Co^{-1} (W - W_o)$ subject to the constraints $o(W) = d$. Fixed point interative solution of this optimization problem is

$$W^{r+1} = W^r + Co\, F\, (F^T Co F)^{-1} [F^T (W^r - W_o) - (O(W^r) - d)] \quad (2.12)$$

Where $F = \dfrac{\partial O}{\partial W}$

2.4 The case of noisy measurements

When the measurements are contaminated with additive noise, the system of equations (2.4) can be written in the form

$$A\,x + n = y \quad (2.13)$$

Where x is the weight or parameter vector, n is noise and y is the observation vector. Using the statistical approach, an estimate for x can be obtained as

$$\hat{x} = \Phi x\, A^T (A\Phi x\, A^T + \Phi_n)^{-1}\, y \quad (2.14)$$

Where $\Phi x = E(xx^T)$, $\Phi_n = E(xnT)$ and $E(xnT) = 0$

If the covariance matrixes Φ_x and Φ_n are diagonal with equal diagonal elements σ_x^2 and σ_n^2 respectively and $\gamma = \sigma_n^2 / \sigma_x^2$, then x can be obtained as a solution of the system of equations

$$(A^TA + \gamma I)\hat{x} = A^Ty \quad (2.15)$$

Let γ be the noise to signal ratio. Let $g = A^T y$. Then an estimate for γ can be obtained as

$$\gamma = \frac{E(g^Tg)\, tr(A^TA) - E(y^Ty)\, tr(A^TA\, A^TA)}{E(y^Ty)\, tr(AA^T) - NE(g^Tg)} \quad (2.16)$$

Where N is the number of data observations. Also in terms of number of weight vector components m, noise vector $\hat{n} = (y - \hat{y})$ where $\hat{y}$ is the estimate of observation vector y, the noise to signal ratio can be estimated as

$$\frac{\sigma_n^2}{\sigma_s^2} = \frac{tr(A^TA)\, E(y^T\hat{n})}{(N - m)\, E(y^Ty) - N\, E(y^T\hat{n})} \quad (2.17)$$

3. Conclusions

In this paper, we presented a training algorithm for layered net based on Taylor's series expansion and the principle of minimum disturbance. We have shown how the algorithm converges and presented the conditions of its convergence. We discussed the use of apriori information and handling of noisy observations in Network training.

References

[1]. R. P. Lippman, "An introduction to computing with Neural Nets", IEEE Trans. ASSP Magazine, pp 4-22, April 1987.
[2]. B. Widrow, and M. E. Hoff, "Adaptive Switching Circuits", 1960 IRE WESCON-CONV. Record, part 4, 96-104, August 1961.
[3]. C. B. Chittineni, "Iterative Methods for the training of Pattern Classifying Machines", MSEE Thesis, IISC., Bangalore India, 1968.
[4]. Sussmann, "on the convergence of Learning Algorithms for Boltzman machines", Rutgers University Center for Systems and Control Technical report, SYCON-88-03, New Brunswick, NJ, 1988.

Classitron: A Flexible Generalization of the Perceptron

Tomas B. Co
Process Modeling and Control Center
Lehigh University, Bethlehem, PA 18015

1 Introduction

The perceptron is an important pattern classification structure devised by Rosenblatt (1962) but is limited to solve only *linearly separable* problems (Minksy and Papert, 1988), i.e. cases where the n-dimensional feature space containing both positive and negative instances can be separated by a single $n-1$-dimensional hyperplane. Several modifications of the basic perceptron structure have been proposed in order to allow effective separation of *non-linearly separable* problems including: multi-layer perceptrons (Rumelhart, et al., 1986), Φ-machines (Nilsson, 1965), multithreshold perceptrons (Takiyama, 1978) and decision-tree layered perceptrons (Koutsougeras and Papachristou, 1988). All three methods have problems in their architectural designs, however: how many hidden nodes, what nonlinear Φ-transformations, and how many threshold levels the devices should contain prior to learning the classification problems, respectively. Although the decision-tree perceptrons address the problem of architecture through structural fexibility, the learning rules depend on the minimization of certain entropy measures; this turns out to be a very difficult problem by itself.

In this paper, we propose a different structure that unifies the three modifications of the basic perceptron structure we have cited above, and we refer to this new structure as the classitron. The classitron structure involves the introduction of a nonlinear mapping box located between the summing junction and the threshold box of the perceptron (see Figure 1). This structural change allows us to decompose the learning procedure into two parts: the first part involves the adjustment of the weights, W, while the second part involves the construction of the nonlinear mapper, f. Consequently, we obtain two important advantages:

1. *The search for the vector of weights is efficient.* A sufficiently smooth objective function can be formulated such that the search space is only $n-1$-dimensional and compact, where n is the number of input nodes.
2. *The structure is flexible.* If the problem is linearly separable, we can simply set $f(\omega) = \tau - \omega$, where ω is the scalar input to the nonlinear mapper and τ is the threshold value. Otherwise, the functionality of the nonlinear mapper can be constructed using direct (i.e. noniterative) algorithms.

2 The Classitron

Assume that the number of input nodes is fixed and, without loss of generality, we have only one output node. The problem can then be formulated as:

The Classitron Problem: Given a set of n-dimensional training instances,

$$\mathcal{U} = \{U_1, U_2, \ldots, U_m\} \quad ; \quad U_i = \begin{pmatrix} u_{1i} \\ \vdots \\ u_{ni} \end{pmatrix}$$

which can be partitioned into two disjoint sets,

$$\mathcal{P} = \{P_1, P_2, \ldots, P_k\} \subset \mathcal{U} \quad ; \quad \mathcal{N} = \{N_1, N_2, \ldots, N_{m-k}\} \subset \mathcal{U} \quad \text{and} \quad \mathcal{P} \cap \mathcal{N} = 0$$

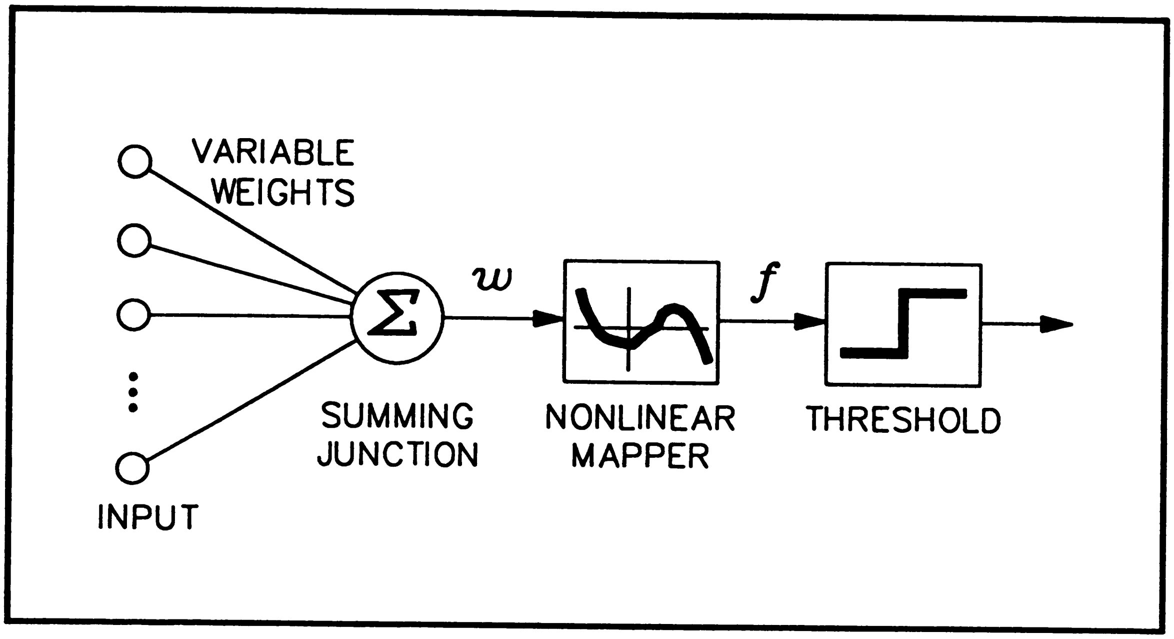


Figure 1: The classitron structure.

find the vector of constant weights, W, and the nonlinear functional, f, such that

$$f(P_i^t W) > 0 \quad \text{and} \quad f(N_j^t W) < 0 \quad \text{for } 1 \le i \le k \ , \ 1 \le j \le m-k \tag{1}$$

Implicit in the problem formulation is the sequence in which the problem can be approached. Observe that the arguments of the nonlinear mapper, f, are the scalar inner products, $U_i^t W$. Thus we could search for the vector of weights first and then construct the nonlinear mapper afterwards.

2.1 Evaluation of the Optimal Vector of Weights

Let us begin by obtaining the set of constraints on W. For the problem to be meaningful, we must have, for all i and j,

$$P_i^t W \neq N_j^t W \quad \text{or} \quad [(P_i - N_j)^t W]^2 > 0 \tag{2}$$

and we refer to this condition as the *Linear Projectability Condition*. Some of the constraints, however, are redundant, thus we need to satisfy only an essential (non-redundant) set of constraints, $\Phi_{ESS}^t W \neq 0$, where Φ_{ESS} can be obtained from $\{P_i - N_j\}$ such that no two elements of Φ_{ESS} are scalar multiples of each other.

Having obtained Φ_{ESS}, we can define a scalar function,

$$J(W) = \prod_{\phi \in \Phi_{ESS}} \phi^t W \tag{3}$$

which is equal to zero if and only if any constraints in (2) are violated. Further, the square of this function increases with increased separability. Here, we use the naive notion that a line containing two classes of instances is more separable if instances of the same class are closer together (i.e. formation of clusters) while, at the same time, instances of different classes are farther apart. An optimization problem that exploits this notion of separability to get the desired vector of weights, W, is

$$\max_W J^2(W) \tag{4}$$

Since the orientation of W is more important than its magnitude, we can choose the form of W to be

$$W = \begin{pmatrix} \cos\alpha_1 \cos\alpha_2 \cos\alpha_3 \cdots \cos\alpha_{n-2} \cos\alpha_{n-1} \\ \cos\alpha_1 \cos\alpha_2 \cos\alpha_3 \cdots \cos\alpha_{n-2} \sin\alpha_{n-1} \\ \vdots \\ \cos\alpha_1 \sin\alpha_2 \\ \sin\alpha_1 \end{pmatrix} \quad ; \; n \geq 2 \tag{5}$$

which has the property: $||W||^2 = W^t W = 1$. The necessary conditions for the solution of the optimization problem, (4), then becomes

$$\sum_{i=1}^{\ell} \left(\prod_{k \neq i}^{\ell} \phi_k^t W \right) \frac{\partial(\phi_i^t W)}{\partial \alpha_j} = 0 \quad ; \quad j = 1, \ldots, n-1 \tag{6}$$

Here, we have recast the problem to that of finding α_i which in turn allows us to restrict the search domain to be a closed and bounded region: $-\frac{\pi}{2} \leq \alpha_i \leq \frac{\pi}{2}$. Thus we know where the global maxima reside. It is important to recall, however, that this optimization is done only for maximal separability, i.e. it is not always necessary to obtain a global maximum.

2.2 Construction of the Nonlinear Mapper

Having obtained W, we can rearrange the set of projected instances in ascending order

$$\mathcal{U}_{\text{PROJECTED}} = \{\mu_1, \ldots, \mu_m\} \quad \text{where,} \quad \mu_i = U_i^t W \quad \text{and} \quad \mu_i \geq \mu_j \quad \text{if} \quad i > j$$

Now for i ranging from 1 to $m-1$, gather a set of points, σ_k, that lie in the middle of μ_i and μ_{i+1}, where μ_i and μ_{i+1} are not both positive or both negative projected instances, and collect these points into the set, Σ. Next, if μ_1 is a positive projected instance, set a parameter $\chi = 1$, otherwise set $\chi = -1$. The nonlinear mapper can then be constructed as

$$f(\omega) = \chi \prod_{\sigma \in \Sigma} (\sigma - \omega) \tag{7}$$

Theorem 1 : *Given a W that satisfies the* Linear Projectability Conditions, *(2), then the nonlinear mapper, f, defined by (7) together with W, solve the* Classitron Problem.

Proof: The value of f is zero only at points included in Σ and changes sign as it passes through these points. Thus (1) is satisfied with the possible exception of reversed inequalities. The value of χ assures the correct sign for the leftmost projected instance. This in turn guarantees that the correct inequality is achieved throughout. Since W satisfies the *Linear Projectability Condition*, such a set Σ could always be constructed.

QED

Example 1: For the 3-input parity problem, we have

$$\mathcal{P} = \left\{ \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right\} \quad \text{and} \quad \mathcal{N} = \left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \right\}$$

Then we get,

$$\Phi_{\text{ESS}} = \left\{ \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix} \right\}$$

and $J = (w_1)(w_2)(w_3)(w_1+w_2+w_3)(w_1+w_2-w_3)(w_1-w_2+w_3)(-w_1+w_2+w_3)$, where $w_1 = \cos\alpha_1 \cos\alpha_2$, $w_2 = \cos\alpha_1 \sin\alpha_2$ and $w_3 = \sin\alpha_1$. A global maximum for J^2 can then be obtained by numerical means to be $\alpha_1 = \tan^{-1}(\sqrt{2}/2)$ and $\alpha_2 = \pi/4$. Finally, we have $f(\omega) = -(\frac{0.5}{\sqrt{3}} - \omega)(\frac{1.5}{\sqrt{3}} - \omega)(\frac{2.5}{\sqrt{3}} - \omega)$.

3 Discussion

We have obtained a flexible generalization of the perceptron which unifies its existing modifications. If the problem happens to be linearly separable, obtaining the globally optimal vector of weights via the classitron procedure will result in a perceptron. Otherwise, a higher order polynomial mapper defined in (7) is used to separate the positive from the negative projected instances. It can be shown (Co, 1989) that an equivalent layered perceptron network exists for every classitron. Also, changing the form of the nonlinear mapper to square pulse instead of polynomial results in a multiple threshold structure. Lastly, if we fully expand the polynomial in (7), then the various additive terms determine the Φ-transformations required by the Φ-machines. This fact is due to the shift from Sigma-Pi structures (Rumelhart, et al., 1986) to Pi-Sigma, i.e. we have focused on locating the roots of the polynomials instead of the coefficients of the polynomials.

In certain cases where the other methods are more advantageous, the classitron can quickly determine both structure and initial conditions for these methods. At present, the learning procedure is nonincremental, and future work will include an incremental learning version to accomodate noisy data and forgetting of obsolete data.

4 References

Co, T. (1989). Classitron: A New Structure for Pattern Classification. (*Submitted for publication*).

Koutsougeras, C. and C. A. Papachristou. (1988). Training of a neural network to pattern classification based on entropy measure. *Procedings of the IEEE Internaltional Conference on Neural Networks, San Diego. Pt. 1*, (pp. 247-254).

Minsky, M. L. and S. A. Papert. (1988). *Perceptrons: An Introduction to Computational Geometry, (Expanded Edition)*. Cambridge, MA: MIT Press.

Nilsson, N. (1965). *Learning Machines: Foundations of Trainable Pattern Classifying Systems*. New York, NY: McGraw Hill Inc.

Rosenblatt, F. (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Washington, DC: Spartan Books.

Rumelhart D. E., G. E. Hinton and R. J. Williams. (1986). Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland and PDP Research Group (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition (Vol 1: Foundations*. Cambridge, MA: MIT Press.

Takiyama, R. (1978). Multiple threshold perceptron. *Pattern Recognition, 10*, 27-30.

GLOBAL MINIMA WITHIN THE HOPFIELD HYPERCUBE

Bruce R. Copeland
Center for Manufacturing Research and Technology Utilization
Tennessee Technological University
Cookeville, Tennessee 38505

ABSTRACT

This paper examines the behavior of the Hopfield network when the connection matrix is negative definite or semi-definite with global minima inside the hypercube of allowable variable space. The original Hopfield circuit is compared with the modified circuit which is in wide use. Using the original Hopfield equations of motion the minima are displaced inward slightly. If the connection matrix is negative definite the error is exactly determined. It is shown that a large class of problems, least squares estimation, have negative semi-definite connection matrices.

INTRODUCTION

The model that Hopfield proposed in 1984 [1] has been used extensively for combinatorial optimization problems, but its application to other problems has been infrequent. However, this network lends itself exceptionally well to another general class of problems. If we have a negative definite or semi-definite connection matrix then the network will converge to the global minimum within the hypercube of the constrained variable space. This paper will discuss several issues related to this problem.

The original Hopfield model is described and compared to the modified model that is in common use. It is shown that the modified model has no error, however the modified equation of motion cannot be implemented in analog hardware. Since most researchers are only simulating the differential equations on sequential digital machines, the modified version is best. However, they cannot make the claim that it can be implemented in an analog parallel form. If we want real-time operation, then the error due to the Hopfield approximation must be studied.

This paper will discuss the use of the Hopfield model for possible real-time applications where the desired result is analog within fixed limits. Thus in this case we want the net to converge to an interior point of the hypercube, rather than a vertex. It is also shown that a negative semi-definite connection matrix always results from a least squares estimation problem.

It should be noted that analog real numbers (as opposed to binary on/off numbers) can be represented as outputs of the Hopfield net quite simply without resorting to the number representation scheme of Takeda and Goodman [3] or other binary representation schemes. By scaling the asymptotic limits of the transfer function to the actual variable limits the output of the net will be the optimum constrained values. This result has also been observed by Matsuda and Akimoto [2]. Equivalently, the variables can be scaled between the asymptotes with appropriate scaling of the other parameters. The method chosen will depend on the implementation. The latter method is appropriate for analog implementation, the former is simpler for digital simulation.

ORIGINAL VS MODIFIED HOPFIELD MODEL

Hopfield proposed his model by considering real neural nets and implementing the important properties of these nets in an analog RC circuit with active nonlinear elements (operational amplifiers). Each neuron is composed of two amplifiers: one inverting and the other normal (or non-inverting). Each neuron is potentially connected to every other neuron in the circuit through some resistance. This particular circuit results in the following equation of motion

$$C_i \frac{du_i}{dt} = \sum_j T_{ij} V_j - u_i / R_i + I_i \tag{1}$$

where the subscript i refers to the ith neuron. C_i is the input capacitance, u_i is the voltage input, V_i is the voltage output and I_i is some fixed input current. The resistance connecting the jth neuron's output to the ith neuron's input is R_{ij}. The magnitude of T_{ij} is given by $1/R_{ij}$ and its sign is negative if connected to the output of an inverting amplifier and positive otherwise. $1/R_i = \sum_j 1/R_{ij} + 1/\rho$ where ρ is the input impedance of the amplifier. The output of the neuron is given by $V_i = g_i(u_i)$ where g_i is some sigmoidal function. Hopfield used a hyperbolic tangent function, but in this paper the logistic function $V_i = 1/(1 + e^{-\lambda_i u_i})$ is used.

He then showed how this equation of motion would minimize a Liapunov function of the form

$$E = -\frac{1}{2}\sum_i\sum_j T_{ij}V_iV_j + \sum_i (1/R_i)\int_{V_{0_i}}^{V_i} g_i^{-1}(x)dx - \sum_i I_iV_i \tag{2}$$

where $V_{0_i} = g_i(0)$. These equations will be referred to as the original Hopfield equations. Hopfield noted that in the high gain limit the integral term would be zero. Thus in the high gain limit the Liapunov function can be written as

$$E_m = -\frac{1}{2}\sum_i\sum_j T_{ij}V_iV_j - \sum_i I_iV_i \tag{3}$$

where the subscript m refers to the modified energy function.

Normal usage of the Hopfield net involves constructing a Liapunov function in the form of equation (3). It has been shown numerous times that E_m is minimized by the modified equation of motion

$$\frac{du_i}{dt} = \sum_j T_{ij}V_j + I_i. \tag{4}$$

However, this equation can no longer be implemented in the analog form proposed by Hopfield, although some digital parallel array may be feasible. It is important to note that most researchers are simply numerically integrating the equations of motion on digital serial computers, while claiming the power of analog parallel processing. Thus they use the modified equation of motion without regard to its implementation.

THE HOPFIELD APPROXIMATION ERROR

The modified energy equation (3) can be rewritten in matrix form as

$$E_m = -\frac{1}{2}v^tTv - v^tb \tag{5}$$

where b and v are nx1 vectors and T is an nxn matrix. The classical approach to finding the value of v that minimizes a cost function of this form is to take the partial derivative of E_m with respect to v and set the result equal to zero. Thus

$$\frac{\partial E_m}{\partial v} = -Tv - b = 0. \tag{6}$$

The points in n-space, v, which solve this equation are critical points and define either a minimum, maximum or saddlepoint. If the Hessian of E_m is positive semidefinite

$$\left|\frac{\partial^2 E}{\partial v^2}\right| = |-T| \geq 0 \tag{7}$$

then the v that solves (6) minimizes E_m. Thus if T is negative semidefinite, any v that solves (6) is a global minimum of E. If T is negative definite, and thus invertible, then (6) can be solved explicitly to give

$$v_m = -T^{-1}b \tag{8}$$

This approach has three major limitations. First, if there are constraints on any of the variables then (8) may give an unrealizable solution. This can be dealt with only by fixing the variables whose constraints have been violated at their limit, reformulating the problem in terms of fewer variables and repeating the problem. An optimum solution is still not guaranteed. Second, T must be inverted which is time consuming for large matrices. If the matrix is ill conditioned or very large (n on the order of 100) then numerical problems can cause inaccurate results. Third, if T is not invertible then the problem cannot be easily solved.

Assuming that the minima lie within the variable hypercube, we can take a similar approach with the unmodified energy equation. In vector notation equation (2) can be written as

$$E_{hop} = -\frac{1}{2}v^tTv - v^tb + G^{-1}(v) \tag{9}$$

where

$$G^{-1}(v) = \sum_{i=1}^{n} \frac{1}{R_i} \int_{v_{0_i}}^{v_i} g_i^{-1}(x)\, dx.$$

Differentiating E_{hop} with respect to v gives

$$\frac{\partial E_{hop}}{\partial v} = -Tv - b + R(v) \tag{10}$$

where

$$R(v) = \left[\frac{g_1^{-1}(v_1)}{R_1} \frac{g_2^{-1}(v_2)}{R_2} \dots \frac{g_n^{-1}(v_n)}{R_n} \right]^t$$

Setting this result equal to zero and solving for v gives us the value of v which minimizes E_{hop}, which we will denote as v_{hop}. Thus

$$0 = -Tv_{hop} - b + R(v_{hop}) \tag{11}$$

The difference, or error, between this result and the value which minimizes E_m is given as

$$v_{hop} = v_m + v_{err} \tag{12}$$

Substituting equation (12) into equation (11) gives

$$0 = -T(v_m + v_{err}) - b + R(v_{hop}) = -Tv_m - b - Tv_{err} + R(v_{hop}) \tag{13}$$

Since $-Tv_m - b = 0$ from equation (6) we have

$$0 = -Tv_{err} + R(v_{hop}) \qquad \text{or} \tag{14}$$

$$v_{err} = T^{-1} R(v_{hop}) \tag{15}$$

if T is invertible.

This simple result can aid in understanding the error. There are three sets of variables that can be modified: g_i^{-1}, $1/R_i$, and T^{-1}. By decreasing any of these factors, we can decrease the steady state error. By increasing,λ_i, the slope of g_i we decrease the slope of g_i^{-1} and thus decrease the error. This result is the same as Hopfield's, since in the high gain limit $E = E_{hop}$. The two terms $1/R_i$ and T^{-1} interact. By decreasing T^{-1} we increase T thus increasing $1/R_i$ since $1/R_i = 1/\rho_i + \sum_j T_{ij}$ where ρ_i is the input impedance of neuron i. So effectively we can minimize the error only by increasing ρ_i or λ_i.

LEAST SQUARES ESTIMATION

Determining a set of real-valued (as opposed to on/off binary) variables that minimize a cost or energy function is a common need in engineering and other fields. In many cases this function is quadratic with respect to these variables. If the cost function is put in the form of the Hopfield energy equation quite often the connection matrix T is negative semi-definite. An example is minimizing the errors in a least squares sense. If there are constraints on the values of these variables then the Hopfield net is an excellent choice.

The general problem addressed in a least squares minimization is to find the vector v that minimizes

$$E' = \frac{1}{2}(Av - k)^t (Av - k) \tag{16}$$

where A is an mxn matrix, v is an nx1 vector, and k is an mx1 vector. Multiplying the terms gives an equation of the form

$$E' = \frac{1}{2}(v^t A^t A v - 2v^t A^t k + k^t k) \tag{17}$$

Since minimizing E' also minimizes $E' - k^t k$ the problem is to find v such that

$$E = \frac{1}{2}(v^t A^t A v - 2v^t A^t k) = \frac{1}{2} v^t A^t A v - v^t A^t k \tag{18}$$

is minimized. This can be rewritten as

$$E = -\frac{1}{2}v^tTv - v^tb \tag{19}$$

where $T = -A^tA$ is a symmetric negative semi-definite nxn matrix and $b = A^tk$ is an nx1 vector. This is the proper form for the Hopfield network with the connection matrix given by T and the set of input bias currents given by b. By appropriate scaling of either the sigmoidal function's asymptotes or by scaling T and b to give per unit values of v the constrained solution can be found by an n neuron Hopfield net.

EXAMPLE

A simple example of a two neuron net with a negative definite connection is presented. The optimum solution lies within the allowable variable space. The solution is found using both the original and modified equations of motion. This error equation is applied. A simple Euler integration was performed with step size of 10^{-4}. The following parameters were used: $\lambda_i = 10^4$, $C_i = 1.0$, $\rho = \infty$, $R_i = 1/3$,

$$T = \begin{pmatrix} -2 & -1 \\ -1 & -2 \end{pmatrix} \quad ; \quad b = \begin{pmatrix} 1.005 \\ 1.995 \end{pmatrix} \quad ; \quad u_{initial} = \begin{pmatrix} 4.5000 \\ 4.5314 \end{pmatrix} x\,10^{-4}.$$

The converged values for the modified method, v_m and the original method, v_{hop}, are

$$v_m = \begin{pmatrix} 0.0050 \\ 0.9950 \end{pmatrix} \text{ in 926 iterations;} \quad v_{hop} = \begin{pmatrix} 0.0065 \\ 0.9935 \end{pmatrix} \text{ in 750 iterations.}$$

The modified method converged exactly to the minimum. Applying the inverse logistic function $g^{-1}(v) = -1/\lambda \ln(1-v)/v$ we find

$$R(v_{hop}) = \begin{pmatrix} 5.0294 \\ -5.0294 \end{pmatrix} x\,10^{-4} \quad \text{and} \quad T^{-1}R(V_{hop}) = \begin{pmatrix} -0.0015 \\ 0.0015 \end{pmatrix}$$

which is correct, demonstrating the validity of the result.

CONCLUSIONS

The Hopfield network is exceptionally well suited to quadratic minimization problems, such as a least squared error. Since the network can be implemented using analog components for fast, stable results, it lends itself well to problems requiring real-time operation. Both negative definite and negative semi-definite connection matrices will give optimal solutions if the modified Hopfield equation is used. If the original equation, or a real analog circuit, is used a small error results, but this error can be easily reduced. An exact solution to the error is given if the connection matrix is negative definite and the solution lies in the allowable region.

Solutions using the original equation of motion are displaced towards the center of the allowable region. If the connection matrix is negative semi-definite (thus with an infinite number of solutions) this can be an advantage since the network will converge to one of the solutions close to the center of the allowable variable space.

Only one neuron is needed for each real-valued variable; there is no need to use large numbers of neurons to represent such a variable. Scaling can be accomplished either through changing the asymptotic limits of the sigmoidal input/output relation or by scaling the variables and associated equations.

If the Hopfield net is to be used in a real-time application it will have to be implemented in hardware. The error introduced by the finite input impedance and finite slope of the op-amps has been discussed at some length in this paper, but other design issues such as component tolerance, sensitivity to asymmetry of the interconnection network, and finite output impedance will all need to be addressed before the Hopfield net can be fully implemented in hardware. This paper demonstrates, I believe, that this effort would be worthwhile.

REFERENCES

[1] Hopfield, J. J. "Neurons With Graded Response Have Collective Computational Properties Like Those of Two-State Neurons," *Proc. Natl. Acad. Sci.* Vol. 81, pp. 3088–3092, 1984.

[2] Matsuda, S. and Akimoto, Y. "The Representation of Large Numbers in Neural Networks and Its Application to Economical Load Dispatching of Electric Power," *Proceedings of the International Joint Conference on Neural Networks*, Washington, D.C. June, 1989, Vol. 1, pp. 587-592.

[3] Takeda, M. and J. W. Goodman,"Neural Networks For Computation: Number Representations and Programming Complexity," *Applied Optics*, Vol. 25, No. 18, pp. 3033–3046, Sept. 1986.

A Neural Network for Explicitly Bounded Linear Programming

Jean-Christophe Culioli and Vladimir Protopopescu
Engineering Physics and Mathematics Division
Charles L. Britton, Jr., and Milton N. Ericson
Instrumentation and Control Division
Oak Ridge National Laboratory, Oak Ridge, TN 37831-6364

Abstract: The purpose of this paper is to describe a neural network implementation of an algorithm recently designed at ORNL [1] to solve the Transportation and the Assignment Problems, and, more generally, any explicitly bounded linear program.

1. Introduction. In a companion paper [2], we study two general neural network models for Linear Programming. Here, we introduce a different model. This model applies only to explicitly bounded linear problems, i. e. problems for which a priori bounds are available for the optimization variables. In particular, it is well suited to the Transportation and the Assignment Problems (TP and AP). Due to the very structure of the explicitly bounded linear problems, the architecture complexity of the network is much simpler. For example, a $K \times L$ AP will require only $K + L$ neurons and $K \times L$ connections, instead of $K \times L + K + L$ neurons and $(K \times L)(K + L)$ connections for the Primal-Dual model studied in [2] or for the circuit proposed by Hopfield and Tank in [3]. The neural network proposed here could also be used to solve the Analog Decoding problem proposed in [4]. In the next Section, we will briefly describe the TP and AP models. In Section 3, we present a network that implements a solution of a slightly more general problem, via a parameterization of the primal variables with respect to the dual variables. We also discuss a possible implementation. Some simulations results are given in Section 4.

2. Two Examples of Explicitly Bounded Linear Problems. The TP can be formulated in the following way: we have to ship some goods from k different sources with stocks S_i, $i = 1, 2, ...K$, to L destinations with associated demands D_j, $j = 1, 2, ..., L$. A transportation cost $c_{ij}x_{ij}$ is associated with the shipment of the (positive) quantity x_{ij} from the source S_i to the destination D_j. One assumes that there is no loss during the process, i.e. for every source i, $\sum_j x_{ij} = S_i$, and also that the demand is met for every destination, that is $\sum_i x_{ij} = D_j$. This defines a "balanced" TP: $\sum_{i,j} x_{ij} = \sum_j D_j = \sum_i S_i$. It is possible, at the price of adding shadow sources or destinations, to transform any unbalanced problem into a balanced one. The problem of minimizing the cost of the transportation leads to the linear program:

$$(1) \qquad \min \sum_{i,j} c_{ij}x_{ij}, \quad \textit{subject to} \;\; x_{ij} \geq 0, \;\; \sum_j x_{ij} = S_i, \;\; \sum_i x_{ij} = D_j$$

The AP has the same mathematical formulation as the TP, except that each stock S_i is equal to 1 and each demand D_j is also equal to 1. A typical application is to assign K jobs to L machines, with operation costs $[c_{ij}]$. As noted before, one can assume, without loss of generality that $K = L$. The AP can be viewed as a combinatorial (0,1)-programming problem. However, it has been shown [5] that it can be expressed as a continuous linear program with $0 \leq x_{ij} \leq 1$. There exist several algorithms of complexity $O(K^3)$ dedicated to solving both problems (see for example [6,7]). From their above formulation, one can notice that explicit bounds on the optimization variables are available $(0 \leq x_{ij} \leq \min\{S_i, D_j\} =: X_{ij})$. We now introduce a generic problem that is somewhat more general that the TP and the AP, but preserves their fundamental properties. We seek for the solution of

$$(2) \qquad \min < c, x > \quad \textit{subject to} \;\; Ax = b, \;\; x \geq 0,$$

where c and x are vectors in R^n, b is a vector in R^m, and A is an $m \times n$ matrix, with $m \leq n$. The brackets $< ., . >$ denote the scalar product in R^n. We assume that the problem (2) has a bounded solution x^* and, for the purpose of the forthcoming derivations, that the rank of A is m. We also assume that the entries of A and the entries of b are positive. This implies that one can compute an explicit bound X^{max} for the

variable x. One choice is X^{max} with entries $X_i^{max} = \frac{\min_j b_j}{\max_j\{a_{ji} \mid a_{ji} \neq 0\}}$. Note that if $X_i^{max} = 0$, this implies immediately that $x_i^* = 0$, and we can remore this variable from the original problem. We thus assume in the following that all components of X^{max} are strictly positive.
The dual problem associated with (2) is [8]

$$\max < b, p > \quad \textit{subject to} \quad A^T p \leq c, \tag{3}$$

where A^T denotes the transpose of matrix A and the scalar product in R^m is also denoted by $< .,. >$. By definition, a vector x such that $Ax = b$, $x \geq 0$ or a vector p such that $A^T \hat{p} \leq c$ will be called *admissible.*
The fundamental result of duality is (for a proof, see [8]):
Proposition 1. *If $\hat{x}$ is admissible for (2) and $\hat{p}$ is admissible for (3) then the duality gap $\delta := < c, \hat{x} > - < b, \hat{p} >$ is positive. If $\delta = 0$, then $\hat{x}$ is a solution of (2), and $\hat{p}$ is a solution of (3).*

3. The Parameterized Neural Network Model. To solve problem (2), we propose to parameterize the primal variables x in the following way

$$x = X^{max} \bullet g_\lambda(c - A^T p), \quad g_\lambda(y) := \frac{2}{1 + e^{\lambda y}}, \tag{4}$$

where the function g_λ is applied componentwise on the vector $c - A^T p$, and the operation "$\bullet$" denotes the Kronecker product of vectors, that is $(y_1, y_2, ..., y_n) \bullet (z_1, z_2, ..., z_n) := (y_1 z_1, y_2 z_2, ..., y_n z_n)$
With this parameterization, we wish to solve in p the equation $Ax(p) = b$ which now writes $AX^{max} \bullet g_\lambda(c - A^T p) = b$. To do that, we consider the variables p as input-states of neurons with output states x (the implementation will be clarified in the next Section) and assume the following dynamics:

$$\frac{dp}{dt} = -(Ax(p) - b). \tag{5}$$

The sigmoid function g_λ has two useful properties that we shall take advantage of in the future derivations. Namely,
$(i) \forall \lambda > 0, \ \forall y, \ \ g'_\lambda(y) = -\frac{\lambda}{2} g_\lambda(y) g_\lambda(-y) \ < 0,$
$(ii) \forall \lambda > 0, \forall y \geq 0, \ \ y \, g_\lambda(y) \leq \frac{2}{\lambda e}.$
We now address the convergence of the network.
Proposition 2. *If p in system (4-5) is bounded, then the network converges to stable states $(\bar{x}, \bar{p})$ which are admissible solutions of (2) and (3).*

Proof: We introduce the Lyapunov functional $E = \frac{1}{2}||Ax - b||^2$. The functional E is bounded below (it is positive) and above, because x given by (4) is bounded. We study the variation of E along the trajectories of (5). We have $\frac{dE}{dt} = < A\frac{dx}{dt}, Ax - b > = < \frac{dx}{dt}, A^T(Ax - b) > = - < \frac{dx}{dt}, A^T \frac{dp}{dt} >$. From (4), we get $\frac{dx}{dt} = \frac{\lambda}{2} X^{max} \bullet A^T \frac{dp}{dt} \bullet g(c - A^T p) \bullet g(A^T p - c)$, which leads to

$$\frac{dE}{dt} = -\frac{\lambda}{2} < X^{max} \bullet g_\lambda(c - A^T p) \bullet g_\lambda(A^T p - c) \bullet A^T \frac{dp}{dt}, A^T \frac{dp}{dt} > .$$

We can rewrite this as $\frac{dE}{dt} = -\frac{\lambda}{2} < DA^T \frac{dp}{dt}, A^T \frac{dp}{dt} > \leq 0$ with $D(t) := diag(X^{max} \bullet g_\lambda(c - A^T p) \bullet g_\lambda(A^T p - c))$, a strictly positive diagonal matrix. Thus E is decreasing along the trajectories of (5). If p is bounded, then each entry of the vector $g_\lambda(c - A^T p) \bullet g_\lambda(A^T p - c)$ is bounded below by a strictly positive constant. The matrix $\tilde{A}(t) := ADA^T$ is, at any time t, an $m \times m$ positive definite matrix, with $||\tilde{A}(t)|| \geq a > 0$, which implies $\frac{dE}{dt} \leq -\lambda a E$. Then E converges to 0 and the trajectories of p and x converge to fixed points $\bar{p}$ and $\bar{x}$ such that $A\bar{x} = b$, $\bar{x} = X^{max} \bullet g_\lambda(c - A^T \bar{p})$. Also, by construction of X^{max}, and due to the factor 2 in the definition of g_λ, the satisfaction of the constraint $A(X^{max} \bullet g_\lambda(c - A^T \bar{p})) = b$ implies that $c - A^T \bar{p}$ is positive. ∎

Remarks.
1. Although we had to assume a "boundedness hypothesis" for p in the theoretical derivation, it appears that in practice, this condition is not very restrictive. It is always possible to limit the iterates of p in a ball of radius R in a computer implementation (which, for R large enough does not perturb the system), but it proved unnecessary.
2. The assumption $rank(A) = m$, needed for the proof of convergence (strict positivity of $\tilde{A}(t)$), can also be relaxed in the numerical tests. In particular, in the case of the TP or the AP, $m = K + L$ but the rank of A is $K + L - 1$. This fact did not seem to alter the simulations either.

Now that we have obtained admissible solutions for (2) and (3), we need to evaluate their associated duality gap δ.

Proposition 3. *The duality gap associated with $\bar{x}$ and $\bar{p}$ is positive and bounded above by $\frac{M}{\lambda}$, where M is a positive constant which depends on the data A and b.*
Proof: The duality gap is positive since $\bar{x}$ and $\bar{p}$ are admissible. We have

$$\delta = < c, \bar{x} > - < b, \bar{p} > = < c - A^T\bar{p}, \bar{x} > = < c - A^T\bar{p}, X^{max} \bullet g_\lambda(c - A^T\bar{p}) > .$$

By using property (*ii*) and the Cauchy-Schwarz inequality, we conclude $0 \leq \delta \leq \frac{2\sqrt{n}||X^{max}||}{\lambda e}$. ∎

In order to do some comparison with "standard implementations", and address the complexity of implementation, we define the vector $z := c - A^Tp$. With this notation, we can write

$$x = g_\lambda(z), \quad \frac{dz}{dt} = A^T(Ax - b). \tag{6}$$

System (6) defines a neural network comprising n neurons with input states z, output states x, activation functions g_λ and thresholds $-A^Tb$. Neurons (z_{i_1}, x_{i_1}) and (z_{i_2}, x_{i_2}) are connected with a connection strength equal to $-\sum_j A_{ji_1}A_{ji_2}$. In conclusion, the only difference between the network (z, x) and Hopfield and Tank's network (u, V) is the absence of a time constant τ. However, it is not necessary to consider so many neurons and connections. The system (4-5) is naturally expressed as

$$\frac{dp}{dt} = -(A(X^{max} \bullet g_\lambda(c - A^Tp)) - b), \tag{7}$$

for which we are led to an implementation whith only m neurons. In the case of the TP or the AP, the implementation is even more simplified. In both cases, $n = K \times L >> m = K + L$. Also, due to the matricial structure of these problems, we can denote the vector x by x_{ij} (with $x_{ij}^{max} = 1$), and associate dual variables p_i and q_j to the rows and columns of the constraints equations ($p_i \leftrightarrow \sum_{j=1}^{L} x_{ij} = 1$ and $q_j \leftrightarrow \sum_{i=1}^{K} x_{ij} = 1$) respectively. With this notation, the system (7) reduces to

$$\frac{dp_i}{dt} = -(\sum_{j=1}^{j=l} g_\lambda(-c_{ij} + p_i + q_j) - 1), \quad \frac{dq_j}{dt} = -(\sum_{i=1}^{i=k} g_\lambda(-c_{ij} + p_i + q_j) - 1). \tag{8}$$

The corresponding neurons p_i and q_j are somewhat different from the "standard neurons" of Hopfield and Tank. They include some "feedback" and a whole vector of internal thresholds (a row or a column of the cost matrix $[c_{ij}]$) in the activation function. Their interconnection is however very simple: each neuron p_i is connected to itself and to all the neurons q_j. The same is true for the neurons q_j. The connection strength are equal to -1 and the external thresholds are all equal to -1.

4. Numerical Application. We report the test of the numerical simulation of (8) for Assignment Problems with $K = L$ rangint from 10 to 100. Note that this corresponds to $10^2 \leq n \leq 10^4$ and $20 \leq m \leq 200$, i. e. fairly large problems. The entries of the matrix $[c_{ij}]$ were generated using a uniform distribution law on $[0, 1]$. The simulation of equation (8) was performed with a step size $\epsilon = \frac{1}{\lambda}$. The parameter λ was taken equal to 10^3. We had the following results (compare the duality gap with an optimal cost of approximately 1):

$K = L$	10	20	30	40	50	75	100
δ	10^{-7}	$2\ 10^{-4}$	10^{-3}	$2\ 10^{-3}$	$7\ 10^{-3}$	$2\ 10^{-2}$	$3\ 10^{-2}$
# iterations	240	170	100	90	110	70	60

One notices that the number of iterations needed to reach a stable state is not increasing with the number of variables. On the other hand, the duality gap deteriorates with the dimensions, as predicted by the above analysis. We have also simulated the network with $\lambda = 10^4$ (and $\epsilon = 10^{-4}$). We noticed the following improvement of the duality gap:

$K = L$	30	50	75	100
δ	10^{-5}	10^{-4}	10^{-4}	$2\ 10^{-4}$
# iterations	820	960	490	460

5. Conclusions. We have presented a neural network model for solving explicitly bounded linear programs. Its low architectural complexity is due to the parameterization of the primal variables (x) with respect to the dual variables (p) which applies very well to matricial problems like the TP and the AP. The theoretical work presented here and many computer simulations seem to prove its applicability. We are now in the process of designing an analog circuit implementation.

Acknowledgements

This research was partially sponsored by the U.S. Air Force Wright Aeronautical Laboratory under DOE Interagency Agreement, DOE-1570-1579-A1, by the Office of Basic Energy Sciences and the Exploratory Studies Program of Oak Ridge National Laboratory, U. S. Department of Energy, under contract # DE-AC05-84OR21400 with Martin Marietta Energy Systems, by DARPA under contract # 1868-A037-A1, and by an ORNL Postgraduate Research Appointment administered by Oak Ridge Associated Universities.

References

[1] J.-C. Culioli and V. Protopopescu, "An Algorithm for Linear Programming That is Easy to Implement", Applied Mathematics Letters, Vol. 2, N. 2, pp.125-129, 1989.

[2] J.-C. Culioli, V. Protopopescu, C. Britton, and N. Ericson, " Neural Networks Models for Linear Programming", this Conference Proceedings.

[3] J. J. Hopfield , and D. W. Tank, "Simple "Neural" Optimization Networks : an A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit", IEEE Trans. on Circuits and Systems, Vol. CAS-33, N. 5, May 1986.

[4] J.-C. Platt and J. J. Hopfield, "Analog decoding Using Neural Networks", Proc. of the AIP Conference, Snowbird, UT 1986.

[5] J. von Neumann, "A Certain Zero-Sum Two-Person Game Equivalent to the Optimal Assignment Problem", in H. J. Kuhn and A. W. Tucker (eds), Contribution to the Theory of Games, Vol. 2, Annals of Mathematics Study N. 28, Princeton University Press, 1953, pp. 12-15.

[6] N. Christophides, Graph Theory. An Algorithmic Approach., Academic Press, London, 1975.

[7] F. Bourgeois and J.-C. LaSalle,"An Extension of the Munkres Algorithm for the Assignment Problems to Rectangular Matrices", Communications of the ACM, algorithm 415, December 1971, Vol. 14.

[8] G. B. Dantzig, Linear Programming and Extensions, Princeton University Press, 1964.

[9] J.-C. Culioli and V. Protopopescu, "Bifurcating Optimization Algorithms and their Possible Application", ORNL/TM-10976, Nov. 1988.

LANGEVIN EQUATIONS AND THE FORMAL FOUNDATIONS OF NEURAL NETWORKS

J. G. Figueroa[£], M. Romero, E. Vargas & C. Flores
£ UNIVERSIDAD AUTONOMA DEL ESTADO DE MEXICO
UNIVERSIDAD AUTONOMA METROPOLITANA-IZTAPALAPA

Laboratorio de Sistemas Complejos
Ap. Postal 70-499. C.P. 04510
México, D.F. MEXICO

Statistical Mechanics offers an interesting theoretical framework for the study of the neurocomputational systems, because it allows working with large sets of interactive elements. In this work we consider the application of certain formalisms of Non-equilibrium Statistical Mechanics in the study of neurocomputers. Particularly, we propose that the formalism that describes Brownian Motion is convenient for studing certain kind of systems, exemplified by the BAM (Bidirectional Associative Memory). This work considers a two-layer network (input and output), interconnected by a synaptic weight matrix.

In the formal derivation of the generalized relaxation equations (Zwanzig, 1961; Mori, 1965) it was found that a set of arbitrary properties of complicated (statistical) systems can be described by analogous equations of the generalized Langevin equations of Brownian Motion. The evolution of the arbitrary set $\mathbf{A} = \{A_1, \ldots, A_U\}$ of dynamical variables is described by the vectorial equation:

$$D_t\mathbf{A} = i\Omega\cdot\mathbf{A}(t) - \int_0^t d\tau\ K(\tau)\cdot\mathbf{A}(t-\tau) + F(t) \qquad (1)$$

where Ω and K are well defined frecuency and memory matrixes, while F is the random force vector. These equations, frecuently obtained by spin relaxation methods, can be derived from Licuville equation through the projection operator formalism of Quantum Mechanics.

We can assume that the BAM network (Kosko, 1987) is a highly stable system, we considered that this neural network is a system at equilibrium, only disturbed during the recognition by the patterns we intend to be recognized by the net. The convergence shown by the BAM is considered as a relaxation towards equilibrium. Brownian Motion is a phenomena that can be explained within this theoretical framework, making its application feasible.

In order to apply the dynamical equations (1) to our particular problem, we consider each neuron's state at the input layer as a continuos dynamical variable. Then the set of states in the input layer makes a set of dynamical variables $\mathbf{A}=\{a_1,\ldots\ldots,a_n\}$ described by eq. (1). Applying the same reasoning to the output layer, represented by $B=\{b_1,\ldots\ldots,b_m\}$, we obtain two sets of equations:

$$D_t\mathbf{A} = i\ \Omega\cdot\mathbf{A}(t) - \int_0^t d\tau\ K(\tau)\cdot\mathbf{A}(t-\tau) + \mathbf{I}(t)$$

$$D_t\mathbf{B} = i\ \Omega\cdot\mathbf{B}(t) - \int_0^t d\tau\ L(\tau)\cdot\mathbf{B}(t-\tau) + \mathbf{J}(t) \qquad (2)$$

We identify **I** and **J** as the input and output initially presented to the net and that has to identify as one of the training patterns. Ω is the frecuency matrix that describes the oscilatory motion of **A** and **B** around equilibrium state. We ask for an exponential decay of the activity at the state fields F_A and F_B when the last two terms do not exist, we impose the condition $\Omega = i\cdot\mathbf{I}$, where **I** is the unitary matrix. Then the first term of each equation becomes $-A_i(-B_j)$ for all i,j.

The second term can be obtained if we analyse the time scales at which the net functions. Even though if the input and output neurons could have internal(inner) structure, it is not relevant for our analysis because in order to study it, we would have to use an smaller time-scale than the one given by the response time of the neuron. Being the second term of (2) a convolution of the form

$$\Xi(t) = \int_0^t d\tau\ K(\tau)\cdot\mathbf{A}(t-\tau) \qquad (3)$$

the temporal evolution of **A** depends of the previous states of the system, not just upon the one in which it currently is. Because it is obvious that the former state is associated with the values that the system has in the opposite field (F_B in this case), we consider this term as the one representing endogenous feedback. The easiest way to represent it in this case is as an adding term of all signals in the opposite field, such as $\Sigma\ S\ (b_j)\cdot m_{ij}$. $S(b_j)$ is the signal that each neuron of the output field sends back to the input field. m_{ij} is one of the elements of the synaptic weight matrix that represents the union (interconnection) of both fields. We make the identification

$$\Xi(t) = \int_0^t d\tau K(\tau)\cdot\mathbf{A}(t-\tau) \text{ ----} > \sum_j S(b_j)m_{ij} \qquad (4)$$

Finally, we change the notation from A_i to a_i (and B_j to b_j). Then, considering all the previous arguments, we rewrite the dynamical equations in scalar form as:

$$D_t a_i = -a_i + \sum_j S(b_j)\cdot m_{ij} + I_i$$

$$D_t b_j = -b_j + \sum_i S(a_i)\cdot m_{ij} + J_j \qquad (5)$$

being the ones reported by Kosko (1987).

This relationship and way to analyse the BAM model shows that it is possible that some of the formalisms currently at use in neural networks are grounded in more general theories of Statistical Mechanics.

REFERENCES

Amari, S. I.; Yoshida, K. & Kanatani, K. I. (1977). A Mathematical Foundation for Statistical Neurodynamics. SIAM J. Appl. Math. , 33 , #1.

Gyorfy, L. (1981). The Rate of Convergence of kn-NN Regression Estimates and Classification Rules. IEEE Trans. Inf. The. IT-27, #3, 362-364.

Kosko, B. (1987). Adaptive Bidirectional Associative Memory. Appl. Opt., 26, 23.

Mori, H. (1965). Prog. Theor. Phys. (Kyoto) 33, 423; 34, 399.

Romero Bastida, M.; Cervantes Orozco, A. & Figueroa Nazuno, J. (1989). The Mathematical Fundations of Neurocomputers. VI National Meeting of A.I., 57-73, (in spanish).

Rumelhart, D. E.; McClelland, J. L and PDP Group. (1986). Parallel Distributed Processing, vol. 1, MIT Press.

Zwanzig, R. Ann. Rev. Phys. Chem. 1965, 16, 67.

CLASSIFIER VOTING IN THE NEURAL NETWORKS

MICHAEL L. GARGANO

A I G Research and Development
70 Pine Street (8th Floor)
New York City, New York 10270
and
Pace University, New York City

Consider a neural network in which input patterns are presented as sensory data to a set of neural classifiers. Each sensor is linked to a neural classifier. Each neural classifier determines as its output that result which is most strongly suggested by the input pattern. An arbitrator then combines the outputs of the individual neural classifiers and develops an output condition which it considers representative of all the individual neural classifiers (see figure 1).

Let the set of conditions be Cond = { C1, C2,...,Cm). Assume the output of each classifier is a permutation of the elements of Cond. This permutation reflects that classifier's preference ordering as to which condition seems the most appropriate output, next most appropriate, and so on. If classifier j has output preference (Cj1, Cj2,.....,Cjm) then classifier j considers Cj1 most appropriate, Cj2 next most appropriate and Cjm least appropriate as its output based on the input presented to it through sensor j. The arbitrator is now presented with preference orderings from each classifier and must combine all these preferences in a "reasonable " way.

As an example, consider a classifier which has to order the elements of the set Cond = {C1, C2, C3} (see figure 2). There are six binary output nodes each representing a preference ordering of Cond. The output of classifier j is a winner-take-all permutation which is then passed to the arbitrator.

The arbitrator must accept a permutation from each neural classifier and in a reasonable way choose one condition as that output most representative of the entire system. We will consider in this paper five well-known voting methods. These are:

1. Majority rule.
2. The Borda count.
3. The Condorcet method.
4. The Runoff method.
5. The Eliminate the loser method.

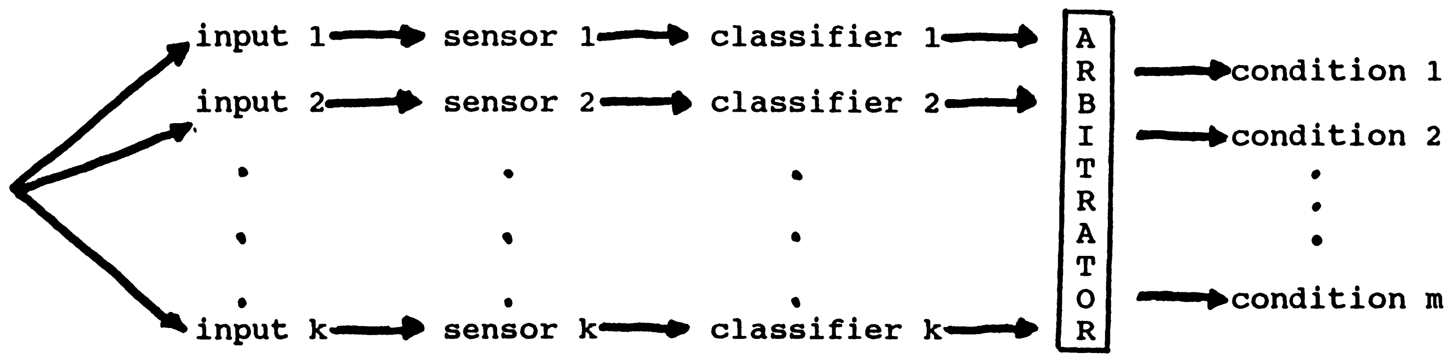


Figure 1. A neural network with arbitrator.

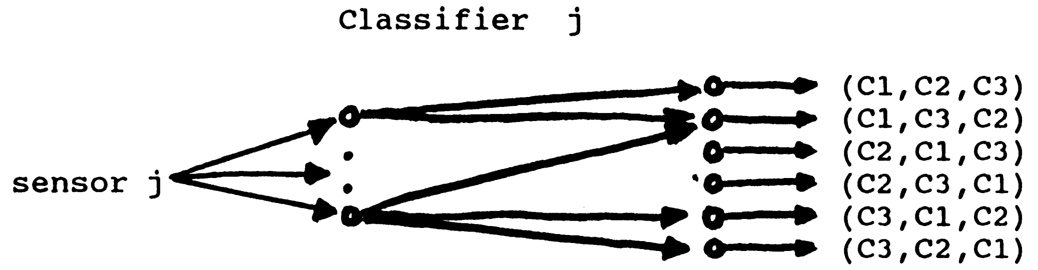


Figure 2. Classifier with Cond = {C1,C2,C3}

The main concern of the arbitrator is developing a result through the use of social choice theory. The neural net engineer should pick a voting method which will give desirable results. Voting, however, is not as straight forward as one might hope and the subject of social choice poses some very difficult theoretical problems. Kenneth Arrow's Impossibility Theorem shows that by using an axiomatic, normative model of social choice no "nice" voting schemes exist. Therefore, none of the five methods mentioned is "best".

The purpose here is to make the neural net engineer aware of the fundamental problem of voting schemes and to give some simple examples of voting methods.

Now let us assume we have a neural network system with 31 classifiers and five conditions so that Cond = {C1,C2,C3,C4,C5}.
Suppose that:
Ten classifiers output the permutation (C3,C1,C2,C4,C5);
Seven classifiers output the permutation (C5,C2,C1,C4,C3);
Six classifiers output the permutation (C4,C5,C2,C1,C3);
Five classifiers output the permutation (C1,C4,C2,C5,C3);
Two classifiers output the permutation (C2,C5,C1,C4,C3); &
One classifier outputs the permutation (C2,C4,C1,C5,C3).

We now consider the actions different arbitrators might take.

1. Majority Rule Voting

The arbitrator simply counts first place votes. C1 gets 5 votes; C2 gets 3; C3 gets 10; C4 gets 6; and C5 gets 7. The winner and output of the arbitrator is C3. Notice, however, C3 is very low in the ratings of the other 21 classifiers.

2. Borda Count Voting

Calculate for each condition Ci its Borda Count defined as the sum, as n ranges from 1 to m, of (m-n+1)*(# of votes in position n). C1 gets a score of 107; C2 gets 106; C3 gets 71; C4 gets 92; and C5 gets 89. The winner and output of the arbitrator is C1 which has the highest Borda score.

3. The Condorcet Method

Here the arbitrator chooses that condition which can beat every other in a head to head contest. C2 beats C1 (16 to 15); C2 beats C3 (21 to 10); C2 beats C4 (20 to 11); and C2 beats C5 (18 to 13). Therefore, C2 is the Condorcet winner.

4. The Runoff Method

First, the arbitrator chooses those two conditions with the largest number of first place votes. Here, C3 and C5 with 10 and 7 first place preferences respectively are chosen. Next, the arbitrator pits C3 and C5 in a head to head contest. Since C3 has only 10 votes to C5's 21 votes the arbitrator chooses condition C5 as the appropriate output.

5. The Eliminate the Loser Method

First, the arbitrator chooses that condition with the lowest number of first place votes and eliminates it. Here C2 would be eliminated. This procedure continues until only one choice is left. Here C1 would be eliminated next, followed by C5 and finally C3. This leaves C4 as the winning condition.

It is interesting to note that in this particular example each method yields a different result: C3 for Majority Rule; C1 for Borda Count; C2 for the Condorcet Method; C5 for the Runoff Method; and C4 for Eliminate the Loser. Thus we see that these different arbitrator mechanisms are independent of one another.

As was stated earlier, the purpose of this paper is to make the neural net engineer aware of the theory of social choice. There are many interesting ideas, as well as, much research literature in this field which may be worthwhile for neural net engineers to to consider when building their neural nets.

References

1. Social Choice Theory - An Introduction; Kelly, Jerry S.; Springer - Verlag 1987.

2. For All Practical Purposes - Introduction to Contemporary Mathematics; Part 3 by Lucas, William F.; Project Director Comap Garfunkel, Solomon; W.H. Freeman and Company 1987

Incomplete Learning Paradigms In Neural Network Computing Models
(Extended Abstract)

David B. Hertz
Guolin Deng
Koushik Basu

Intelligent Computer Systems Research Institute
University of Miami
Coral Gables, FL 33124

Abstract

Learning (human, animal, or machine) by example seems to take place when pieces of some pattern are perceived more or less simultaneously. The computational properties of the resulting combinations result from the collective action of the many parts linked together in a network. Most neural network learning paradigms are essentially learning from examples, either provided by a teacher (supervised learning), or by the environment (unsupervised learning). In this paper we describe some neural network learning paradigms and investigate the incompleteness and complexities of those paradigms. We postulate that for any intelligent system, in order to learn anything, it must be able to compute everything. We examine the relationship of this postulate to certain neural network models.

1. Introduction.

The issue of learning is central to the study and application of advances in artificial intelligence. The capability to 'learn' may be the most important factor separating the so-called intelligent machines from the more conventional ones. Theories of learning are important because models derived from them are relevant to building learning devices that can change their output behaviors in some rational remembered fashion in response to environmental inputs. Our objective in this paper is to examine what the learning processes proposed in certain neural network computing models cannot achieve.

It is appropriate at this point to formalize a definition of learning. The easy way out would be to borrow from the abundant psychology literature and come out with a definition from the psychologist's point of view. However, most psychologists' definitions of learning lack the appropriate mathematical structures which are essential if we hope to investigate the incompleteness of the learning capabilities of popular neural network models. In addition, there is the greater danger that we might be entrapped in a generalized description of the learning process which might be impossible to model with the currently over-simplified neural network computational paradigms. For example, psychologists deal with the higher level cognitive processes of the brain, e.g. learning by discovery, learning by being told, learning by analogy, learning from examples, some of which are extremely difficult to simulate on any current computer models which can at the best to mimic the low level congnitive processes of the brain. Among all those learning strategies described by the psychologists, only learning from examples has been successfully modeled thus far.

Both Valiant and Baum have formalized a mathematical structure for learning by examples and have investigated it in depth. In particular, developments have been made in the definition of plausible learning protocols and in the investigation of classes of concepts that can be recognized computationally, using such protocols, and exploring the class of concepts that can be learned in a reasonable polynomial number of steps.

According to Valiant, there are 'specific classes of concepts that can be learned via protocols that allow for specific kinds of information supply. The protocol involved in learning by examples recognizes the availability of typical data that positively exemplify each concept. The positive examples are assumed to follow a fixed (but unknown) probability distribution. Computational models that can learn by examples under these conditions make use of algorithms that make a polynomial number of calls to a routine which generates the examples (called EXAMPLES in Valiant's work)

Classes of concepts that can be characterized by bounded conjunctive normal form (k-CNF) expressions, can be learned by example in this context. This class of representations (k-CNF) are used in representing boolean functions and can thus be easily incorporated into computational models of learning.

Baum recognized that although any boolean function can be expressed in CNF, not all classes of concepts characterized by CNF could be learned in reasonable time. Simply, a class of such functions can be learned in exponential time, with the algorithm making a polynomial number of calls to the examples routine. Thus if the class contains more than 2^{c^n} distinct functions (for $1 < c \leq 2$), it can not be learned in polynomial time. For practical reasons, this limitation constrains the classes of functions which can indeed be learned by example in a reasonable (polynomial) amount of time. The classes that can be learned in polynomial time are those with $2^{p(n)}$ functions for any polynomial p.

Moreover, it can be stated reasonably that most of the neural network learning paradigm functions are essentially learning from examples. Such examples might be provided by a teacher (supervised learning), or by the environment (unsupervised learning).

As for the relationship between the learnability and computability, we can postulate that: for any intelligent system, in order to learn anything, it must be able to compute everything.

In this paper, we will discuss several most popular learning paradigms in neural network computing models and investigate the incompleteness and complexities of those learning paradigms.

2. Neural Network Computing Models.

The neural network devices can be defined as follows:

D = { (A, P) | A is a network of computing units (neurons), P is a programming procedure }.

The parameter A defines the special classes (or "architectures") of networks of computing units; P defines procedures by which a specified architecture may be programmed.

From the structure point of view, neural networks can be defined as single-layer or multi-layers. The neural elements can be distinguished as input units, ouput units, or hidden units.

From the connectivity point of view, the neural networks can be specified as feedforward neural networks or feedback neural networks; the connections can be specified as fully connected, locally connected, or sparsely connected.

The mathematical representations of neural elements may be defined as networks of linear threshold functions (LTFs) or semi-linear activation functions (SAFs).

A given LTF v_i may be defined as:

$$v_i = \mathrm{sgn} (\Sigma\, w_{ij}\, v_j + t_i)$$

where w_{ij} represents the strength of the connection between the ith and the jth LTF, t_i is a threshold value.

An LTF is a limiting case of a more general class of functions, the SAFs which take the form:

$$v_i = g (\Sigma\, w_{ij}\, v_j + t_i)$$

where g is a monotonically increasing, differentiable function, and t_i is termed the bias.

For example, the logistic function employed by Rumelhart takes the form:

$$v_i = \frac{1}{1 + e^{-\beta\Sigma(w_{ij}v_j + t_i)}}$$

in which t_i is the bias and β is a parameter.

From the view point of the power of the computational approach, we need to investigate and discuss the following four aspects for any given neural network computing device D(A, P):

(1) Define A, i.e. define the special architectures of neural network computing device.
(2) Define P, i.e. define learning procedures by which a specified architecture may be learned.
(3) Computability, i.e. examine the class of functions that learnable by D(A, P).
(4) Complexity, i.e. examine time and space complexities that with respect to the class of functions that are learnable by D(A, P).

3. Special Learning Paradigms.

3.1. Single-layer Perceptrons.

Single-layer perceptron is one of the simplest and probably the best understood types of feedforward neural network models. The single-layer perceptron generated much interest when it was initially developed in the 1950s by Rosenblatt because of its ability to learn to recognize simple patterns. It can be trained using the perceptron convergence procedure or the LMS (Least Mean Square) algorithm to classify a continuous-valued or binary-valued input vector into one of two classes. Therefore, it can be used as classifier and for adaptive signal processing.

However, only linearly separable functions are learnable by the single-layer perceptrons. Some examples of the linearly separable functions are: Boolean AND, OR functions, and "at least 1 of N", "at least k of N" functions.

A case in point for the limitations of the single-layer perceptron is the computation of the Boolean function XOR (exclusive or) which is not a linearly separable function. Therefore XOR can not be computed by the single-layer perceptron.

From the complexity point of view, the time to learn an arbitrary linearly separable function grows exponentially with the number of inputs. The time t is bounded by:

$$2^N < t < N^N$$

where N is the number of inputs to the perceptron.

However, it has been proved that the time to learn the "at least k of N" function grows only polynomially with the number of inputs. The time t to learn this function is bounded by:

$$t < N^3$$

where N is the input size.

It has also been proved that the time t to learn the "at least 1 of N" function is bounded by:

$$t < N^2$$

where N is the input size.

3.2. General Feedforward Neural Networks.

The structure of the general feedforward neural network device is demonstrated in Fig. 1.

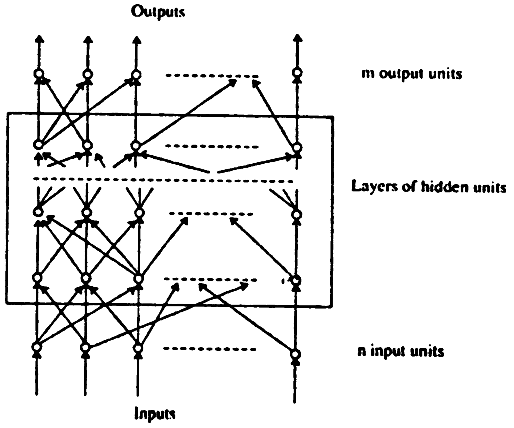


Fig. 1. General feedforward neural network structure.

The general feedforward neural networks are characterized by multi-layer neural networks whose connections exclusively feed inputs from lower layers to higher layers. In contrrast with feedback neural network, a feedforward neural network operates only until its inputs propagate to its output layer.

In 1967, Minsky showed that the general feedforward neural networks possess the same computational power as the Turing machines. Therefore, we can say that the general feedforward neural networks are the complete computing devices in terms of the computability under the Turing machine computing models, or the Kleene recursive function computing models, or the von Neumann computer programming models.

We list some simple examples of functions that learnable by the feedforward neural networks and the spacial complexities as follows:

(a) Parity testing function.

Spacial complexity: need $O(n)$ hidden units and $O(n^2)$ connections; where n is the input size.

(b) Symmetry testing function.

Spacial complexity: need n input units, 1 output unit, 2 hidden units, and 2n+2 connections; where n is the input size.

(c) Encoding function.

Spacial complexity: need n input units, n output units, $\log_2 n$ hidden units; where n is the input size.

(d) The function for discriminating between alphabetic characters.

From the incompleteness point of view, some functions that can not be computed under the Turing machine computing models probably can not be learned by the feedforward neural networks either. For instance, we can show that, it is impossible to build a feedforward neural network device to compute the function Π, which for any two functions f_i and f_j that learnable by the feedforward neural networks as the inputs to Π; if f_i is identical to f_j, then Π will output 1, otherwise Π will output 0. i.e. we define Π as follows:

$$\Pi(f_i, f_j) = \begin{cases} 1 & \text{if } f_i = f_j \text{, and } f_i, f_j \text{ are learnable by feedforward neural networks.} \\ 0 & \text{otherwise.} \end{cases}$$

Then Π is not learnable by any feedforward neural network device.

3.3 General Feedback Neural Networks

The feedback neural network is an extremely significant part of the neural network learning tool kit. However in contrast to the feedforward model it requires additional consideration to deal with the learning from examples function. The learning function for backward propogation that parallels the Rumelhart logistic function is:

$$\delta_{pj} = \begin{cases} (t_{pj} - O_{pj})\, O_{pj}(1 - O_{pj}) & \text{output layer} \\ O_{pj}(1 - O_{pj})\, \Sigma \delta_{pk} w_{kj} & \text{hidden layer} \end{cases}$$

The local processing element j must have access to the connection weights to both the forward (w_{kj}) and backward (w_{ij}) connected to it. This brings the problem that the weights are external and must be shared among the processing elements so that the connection weights can be updated according to:

$\Delta w_{ji}(n + 1) = \eta(\delta_{pj} O_{pi}) + \alpha \Delta w_{ji}(n)$

where η is the learning rate and α is the momentum.

And finally under a number of circumstances, the back propagation learning process is subject to the arrival at a processing element of both forward and backward propagating signals simultaneously. This can reduce the number of learnable classes significantly.

4. Conclusions and Discussions.

Examples present available information from which learning is potentially possible to one or another of the various neural networks. Learning means that, at some time in the future, when a program is faced with particular examples it will respond with suitable input/output replications, within a preconfigured time frame, in a statistically measurable fashion. The time frame for the programmed computations (polynomial, exponential) and the statistical error measures give a basis for judging learnability.

At present, programs that require polynomial computation steps for non-trivial learning are available. Learned and remembered patterns that can fill gaps (e.g. visual or aural) are governed by linkages of individual remembered parts. Analogies to the brain are somewhat murky, and certainly chemical. Since a single neuron can receive up to 200,000 signals on its dendritic tree, a given single sensory pattern probably uses only a small fraction of the sites available to it. Incomplete learning is likely to be continuous in the chemical structure, but opportunity to add to, and reinforce is equally powerful. Paralleling the forms of chemical activity in an electronic structure may be the next step for going beyond the present limitations on programming learnability, since these processes probably operate in the equivalent of polynomial time.

References.

[1] Valiant, L. G., *A Theory of the Learnable*, Communications of the ACM, Nov. 1984.

[2] Baum, E. B., *Complete Representations For Learning From Examples*, 1987.

[3] Egecioglu, O. et al., *Computable Functions and Complexity in Neural Networks*, "Real Brains, Artificial Minds", edited by John L. Casti, 1987.

[4] Alkon, D. L. and Rasmussen, H., *A Spatial Temporal Model of Cell Activation*, Science, Vol. 239, Feb. 26, 1988.

[5] DARPA, *Neural Network Study*, 1988.

[6] Rumelhart, D. E., et al., *Learning Internal Representations by Error Propagation*, Parallel Distributed Processing, Vol. 1, MIT Press 1986.

A Performance of Neural Network Classifiers for the 1-Class Classifier Problem

Don R. Hush and John M. Salas
Department of Electrical Engineering and Computer Engineering
University of New Mexico
Albuquerque, NM, 87131 USA.

ABSTRACT

This paper compares the performance of four different classifiers. The classifiers included in this study are the multi-layer perceptron (MLP), high-order neural networks (HONN), localized receptive fields (LRF), and the learning vector quantization (LVQ) method. The performance of each of these methods is studied for the I–4I problem. This problem is used to represent a class of problems called "1-class classifier" problems. In these problems the classifier is required to from a decision boundary that completely surrounds a class of data. The classifiers are compared in terms of their minimum classification error, computational complexity, learning rate, number of training samples required, and sensitivity to network size. Our results show that the localized receptive field model provides the best overall performance for this problem.

1. Introduction

This paper presents a comparison between four different nonparametric classifiers. The multi-layer perceptron (MLP) is without question the most commonly used neural network classifier, and thus provides a useful standard for comparison. The high-order neural network (HONN) model can be viewed as an extension of the MLP network. Most conventional methods which perform nonparametric classification are based on the "distance classifier" concept. The k-nearest neighbor (k-NN) classifier is a well-known example. The learning vector quantization (LVQ) method considered here is also a distance classifier, and in most problems provides better performance than the k-NN method at a greatly reduced computational cost. Finally, the localized receptive field (LRF) model can be viewed as a combination of the LVQ and MLP models. These classifiers are compared in terms of their minimum classification error, computational complexity, learning rate, number of training samples required, and sensitivity to network size.

2. The 1-class Classifier Problem

The class of pattern recognition problems dealt with in this paper are called "1-class classifier" problems. In the simplest case these problems are characterized as having a single class of data to be recognized. All other patterns are to be recognized as not belonging to this class. For example, we may be interested in recognizing a particular type of vehicle in an image. There may be several other types of vehicles in the image and some may look very similar to the one we are interested in. However, our information about the other vehicles is incomplete. In a sense this can be viewed as a 2-class pattern recognition problem; one class being *target* and the other *non-target*. However, our information about the non-target class is incomplete. In such cases the only alternative is to design a classifier that forms a decision boundary that completely surrounds the target class. All patterns that fall within the boundary are classified as target, and all of those outside the boundary as non-target.

The specific pattern recognition problem chosen here is the I–4I problem. The I–4I problem is a two-class problem in which both class distributions are Gaussian with zero means. The covariance matrix for the first class is I (the identity matrix) and for the second is 4I. If the classes are equally-likely, and we assume a zero-one loss function, then the Bayes classifier for this problem takes on the form: if $d(\mathbf{x}) = \mathbf{x}'\mathbf{x} - 1.85n < 0$ then assign $\mathbf{x}$ to class ω_1, else assign to class ω_2. $\mathbf{x}$ is the pattern vector, $\mathbf{x}'$ is the transpose of $\mathbf{x}$, and n is the dimension of $\mathbf{x}$. The decision boundary, described by the equation $d(\mathbf{x}) = 0$, is an n-dimensional hypersphere of radius $r = \sqrt{1.85n}$. In this paper we will be presenting results for the case

This work was supported in part by Sandia National Laboratories, Albuquerque, Mew Mexico, under contract number 05-8801.

where $n=8$. Results for other dimensions are similar. With $n=8$ the Bayes classification error rate is approximately 9%. This problem is characteristic of the 1-class classifier problem because it requires that the classifier form a decision boundary that completely encloses one class. The advantage of choosing this problem is that it is well defined so that we have an exact measure of optimal performance with which to compare.

3. Multi-Layer Perceptron, MLP

The multi-layer perceptron network is probably the most widely used neural network classifier. The back-propagation algorithm is used to train the network (Rumelhart, McClelland, & Williams 1986). When comparing the computational complexity of the different methods in this paper all comparisons will be made relative to the complexity of a MLP node. With $n=8$ each node in the first layer requires 8 multiplications, 8 additions, and 1 sigmoid operation.

The classification results for 2-layer (1 hidden layer) perceptron networks are shown in Figures 1 and 2. The horizontal axis represents the number of hidden layer nodes, N_1, and the vertical axis the percent classification error. Results are shown for training sets of sizes 400 and 3200 samples per class. A significant improvement in performance is noted as the number of training samples is increased. The curves also show a significant improvement in performance as the number of first layer nodes is increased beyond $N_1=5$. This performance begins to level off around $N_1=20$ nodes which would be the optimal network size for this problem in terms of the trade-off between performance and computational complexity. The best performance is near 11%, which is still approximately 2% above the Bayes error for this problem.

4. High-Order Neural Networks, HONN

The basic architecture of high-order neural networks is the same as that of MLPs except that they are allowed to form higher order terms (correlation terms) at the input to each layer (Giles & Maxwell 1987). In this paper the high order terms are created only at the first layer, not between layers. In addition we create only second order terms. All $n(n+1)/2$ second order terms are formed. Thus, the HONN model used here can be viewed as a MLP net with $n+n(n+1)/2$-dimensional data vectors which are formed from the original n-dimensional data as follows,

$$\mathbf{x}_{HONN} = \left[x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots, x_n^2, x_1x_2, x_1x_3, \cdots\right] \tag{1}$$

Because of the increased dimensionality of the input vector the computational complexity of a first layer node in the HONN model is approximately $1+(n+1)/2$ times that of an ordinary (linear) MLP node. Training for the HONNs is the same as for the MLP. Learning is often faster with the HONN model, however, because fewer nodes/layers are required.

The classification results for the HONN model are also shown in Figures 1 and 2. In this case the horizontal axis represents the complexity of the nodes in the first layer relative to a MLP node. The first point on each of the four curves corresponds to a 1-node 1-layer HONN net, while the remaining points are for 2-layer nets with one node in the second layer. The best performance is achieved with a single node, and approaches the Bayes error as the number of training samples is increased. The reason for this is that the second order terms at the input allow the net to form a hyperspherical decision boundary with a single node. Even though the computational complexity of a HONN node is higher than that of a MLP node it is less than that of ~20 MLP nodes, so in this problem the HONN model provides a more efficient solution.

As the number of nodes in the HONN model is increased the classification performance deteriorates. This suggests that the HONN method is very sensitive to the size of the model selected. As expected, however, as the number of training samples is increased the oversized networks are forced to learn the correct classification.

5. The Learning Vector Quantization Method, LVQ

The LVQ method is described in (Kohonen 1988). When the LVQ classifier is cast into a neural net architecture it can be viewed as a 2-layer net. The first layer (hidden layer) contains one node for each representative pattern. The function of each node is to compute the Euclidean distance between the input pattern and the corresponding representative pattern. These distance values are then fed forward to the second layer node which then selects the smallest of its inputs and makes the appropriate class assignment. The computational complexity of an LVQ node is approximately equal to that of a MLP node.

The classification results for the LVQ method are shown in Figures 1 and 2. Again the horizontal axis represents the number of first layer nodes (i.e. the number of representative patterns). The results here are similar to those for the MLP classifier. The performance begins to level off at around $N_1=20$ nodes. The classification errors, however, are slightly worse than those of the MLP. Also, we note an improvement in performance as the number of training samples is increased, similar to the previous two methods.

6. Localized Receptive Fields, LRF

The Localized Receptive Field, LRF, model is similar to LVQ in the sense that the first layer nodes perform a similar function. In (Moody & Darken 1988) the first layer nodes are trained with the K-means algorithm. The function of each node is then to compute a *receptive field response function* which is a monotonically decreasing function of the distance between the input pattern and the node representative.

The LRF model used here works as follows. Rather than compute a receptive field response function at each first layer node we simply feed the Euclidean distances forward as we did in the LVQ method. Subsequent layers then function as a MLP. That is, rather than restrict ourselves to a single second layer node we allow for several nodes and several layers beyond the first. These subsequent layers are trained with the back-propagation algorithm. Because the first layer nodes are equivalent to LVQ nodes, and nodes in subsequent layers are simply MLP nodes, the average computational complexity of a LRF node is approximately equal to that of a MLP node.

The classification performance results for the LRF method is shown in Figures 1 and 2. All results are for 2-layer nets with one node in the second layer. Once again the horizontal axis represents the number of nodes in the first (unsupervised) layer. Optimal performance is achieved in this case with a single node in the first layer. In addition this classifier comes much closer to the Bayes error than any of the previous methods. A single node in the first layer is sufficient here because the Euclidean distance which is fed forward from this node carries second order terms which can be properly weighted by the second layer node to form the desired hyperspherical decision boundary. The performance of this method is approximately constant as N_1 is increased beyond the optimum, and improves slightly as the number of training samples is increased.

7. Summary

In terms of classification performance the LRF method gave the best results. The second best is the HONN classifier whose performance is very close to that of the LRF model for a large number of training samples. The MLP and LVQ methods rank third and fourth in classification performance. Both of these methods solve this problem by placing a large number of linear decision boundaries around the inner class, and then connecting these boundaries. The MLP classifier connects these boundaries in a smooth continuous fashion while the LVQ classifier connects them in a piecewise (discontinuous) manner. Clearly any approach that builds up a decision boundary in this manner will perform worse than one that implements it directly in parametric form as in the LRF and HONN classifiers.

The LRF classifier not only provides the best classification performance but is also the most efficient method. It should be noted that the HONN method can be made just as efficient if the linear and cross terms in Eq. (1) (which are not needed in this problem) are removed. The computational complexity of the MLP and LVQ methods is an order of magnitude more than either the LRF or HONN methods because they require several first layer nodes to adequately solve this problem. This effect is even worse in higher dimensional problems. The LRF and HONN methods will never require more than one first layer node to enclose a region of the decision space while the number of nodes required in the MLP and LVQ methods will continue to grow as the dimension of the problem increases.

The learning algorithms for all methods in this study can be viewed as gradient search algorithms applied to a nonlinear optimization problem. As such it is extremely difficult to make meaningful quantitative comparisons of the learning rates. Qualitatively however we can make some very strong statements. The slowest learning algorithm by far is the back-propagation algorithm used to train the MLP classifier. This is a well-known limitation of this approach. Even though the same basic algorithm is used for learning in the HONN method, learning is faster because this method requires fewer layers and fewer nodes. Two forms of learning are used in the LRF model. An unsupervised algorithm, e.g. K-means, is followed by a supervised algorithm, namely back-propagation. The unsupervised learning algorithm is generally very fast relative to back-propagation. As with the HONN method, the supervised learning phase which uses back-propagation is faster for LRF than it is for MLP because fewer nodes/layers are required. The LVQ learning algorithm is the fastest of all. The fundamental reason for this is that learning is performed in only one layer. Because of this the nature of the optimization problem is greatly simplified.

As expected, all four methods give an improvement in performance as the number of training samples is increased. This improvement is largest with the MLP and LVQ methods and smallest with the LRF method. This suggests that the MLP and LVQ methods require the largest number of samples to produce accurate generalizations and that the LRF method requires the least. This happens primarily because the LRF method has the fewest number of free parameters (weights) and the MLP and LVQ methods have the most.

All networks, regardless of their structure, will perform poorly if their size is too small. A more interesting question, and the one of concern here, is what happens to the performance when the optimal network size is exceeded? In the MLP and LVQ methods the performance changes (improves) very gradually as the number of first layer nodes exceeds the desired operating point (near $N_1 = 20$). In the LRF method little or no degradation is noted as the optimal number of nodes is exceeded. However, in the HONN method a significant degradation in performance is observed as the optimal number of nodes is exceeded. Thus, this method is very sensitive to network size selection.

8. References

Giles, C.L., & Maxwell, T. (1987). Learning, invariance, and generalization in high-order neural networks. *Applied Optics*, **26**, 4972-4978.

Kohonen, T., Barna, G., & Chrisley, R. (1988). Statistical pattern recognition with neural networks: benchmarking studies. In *Proceedings IEEE 2nd International Conference on Neural Networks*, **1**, 61-68.

Moody, J., & Darken, C. (1988). Learning with localized receptive fields. In *Proceedings of the 1988 Connectionist Models Summer School*, 133-143.

Rumelhart, D.E., McClelland, J.L., and Williams, R.J. (1986). Learning internal representations by error propagation. In D.E. Rumelhart & J.L. McClelland (Eds.), *Parallel Distributed Processing*. Cambridge, MA: MIT Press.

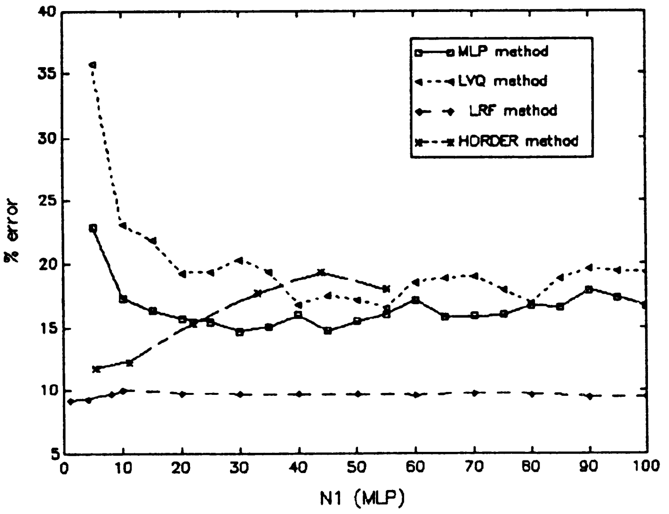


Figure 1 – Performance Comparison: N=400, n=8

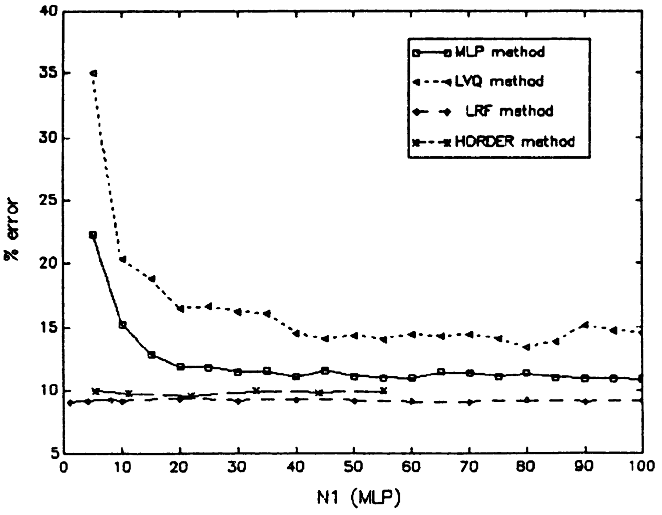


Figure 2 – Performance Comparison: N=3200, n=8

A Synchronous Equivalent To Asynchronous Network Dynamics

Yoshio Izui[1] and Alex Pentland

Vision Science Group, The Media Laboratory
Massachusetts Institute of Technology
20 Ames St., Cambridge MA 02139

Abstract

We show that the dynamical behavior of an asynchronous neural network can be closely approximated by a synchronous network whose dynamics take into account the Hessian of the energy function. Thus the advantages of asynchronous networks — biological realism, improved convergence speed, and lack of oscillation — may be achieved using a synchronous machine with only small additional effort.

1 Introduction

Understanding the behavior of asynchronous networks is a topic of considerable interest for several reasons. First, of course, virtually all biological neural networks appear to be asynchronous. Further, in some networks this asynchrony is critical to the correct functioning of the the network. An example is the Hopfield network [1] which can show sharp, random oscillations when using synchronous state updates unless parameters are adjusted carefully, but which is stable when updates are applied asynchronously [2]. An intuitive understanding of this oscillatory behavior is that asynchronous updating causes smaller changes in the states space, so that sharp variations in network energy can be more accurately tracked.

The problem with investigating asynchronous networks, however, is that they are very expensive to simulate because very small time steps must be employed. In this paper we prove that asynchronous operation may be closely approximated by considering the Hessian of the network's energy function when computing the state update function. This result allows the asynchronous behavior of a network to be simulated at approximately the same cost as simulating its synchronous behavior.

2 Approximation of asynchronous operation

The state update equation of a neural network may be associated with an energy function E which controls the behavior of the network over time according to the following equation:

$$\frac{dW}{dt} = -\eta \frac{\partial E}{\partial W} \tag{1}$$

[1]Current address is Industrial Systems Lab., Mitsubishi Electric Corp., 8-1-1, Tsukaguchi, Amagasaki, Hyogo 661 Japan

where W is the state vector of the network. In synchronous operation equation (1) is calculated for all components weights W_j at each time step, while in asynchronous operation only one weight (and the system energy E) is updated at each time step.

We will approximate the behavior of asynchronous networks by using time averaging. Let the time step for each weight in the asynchronous networks be a random variable with mean Δt, so that on average every weight in the network is updated every Δt increment in time. We will then define a larger time interval $\Delta T = K\Delta t$, and times $T_j = T_{j-1} + \Delta T$, where K is a arbitrary number. The goal of this paper, then, is to define a synchonous network with time steps T_j whose behavior matches that of the asynchonous network averaged over K time steps.

2.1 Asynchronous State Update Equations

We first define the first and second order derivatives of E at (asynchronous) times t_k to be

$$A_k = \left.\frac{\partial E}{\partial W}\right|_{W=W(t_k)} \qquad B_k = \left.\frac{\partial^2 E}{\partial W^2}\right|_{W=W(t_k)} \tag{2}$$

and define that all subscripts of A, B, and t are taken to be modulo K, where we omit subscript j of weight W for simplicity.

We will next note that the first order derivative at time t_{k+1} can be obtained by using the first order derivative at time t_1 and second derivative at times $t_1 \cdots t_k$ as below:

$$\begin{aligned} A_{k+1} &= A_k + B_k \Delta W_k & (3)\\ &= A_k\left(1 - \eta B_k \Delta t\right) & (4)\\ &= A_1 \prod_{l=1}^{k}\left(1 - \eta B_l \Delta t\right) & (5) \end{aligned}$$

Assuming that ΔT is small, and thus that Δt is also small, then at time T_i the K-time-step time-averaged first derivative is

$$\begin{aligned} \left.\widehat{\frac{\partial E}{\partial W}}\right|_{W=W(T_i)} &= \frac{1}{K}\sum_{k=1}^{K} A_k & (6)\\ &= \frac{A_1}{K}\sum_{k=1}^{K}\left\{\prod_{l=1}^{k}\left(1 - \eta B_l \Delta t\right)\right\} & (7)\\ &\approx \frac{A_1}{K}\sum_{k=1}^{K}\left(1 - \eta \Delta t \sum_{l=1}^{k} B_l\right) & (8)\\ &\approx A_1\left(1 - \frac{\eta \Delta T}{2} B_1\right) & (9)\\ &= A_1\left(1 - \frac{\eta \Delta T}{2} \left.\frac{\partial^2 E}{\partial W^2}\right|_{W=W(T_i)}\right) & (10) \end{aligned}$$

Thus the time-averaged state update equation for an asynchronous network is

$$\left.\frac{dW}{dt}\right|_{W=W(T_i)} = -\eta A_1\left(1 - \frac{\eta\Delta T}{2}\left.\frac{\partial^2 E}{\partial W^2}\right|_{W=W(T_i)}\right) \tag{11}$$

2.2 Synchronous Second-Order State Update Equation

A more sophisticated version of the update function in equation (1) takes into account the curvature of the energy surface by employing the Hessian of the energy function [3]:

$$\frac{dW}{dT} = -\eta(\nabla^2 E)^{-1}\nabla E \tag{12}$$

As is conventional in employing this update function, off-diagonal terms of $\nabla^2 E$ are assumed to be zero and we write W_j to indicate the j^{th} component of W. Thus the second-order update equation for each weight j is simply:

$$\frac{dW_j}{dT} = -\eta\frac{\frac{\partial E}{\partial W_j}}{\frac{\partial^2 E}{\partial W_j^2}} \tag{13}$$

When the second derivative is small, however, ΔW_j becomes unstable, and so it is common to use either the following "stabilized" second-order update function:

$$\frac{dW_j}{dT} = -\eta\frac{\frac{\partial E}{\partial W_j}}{1+\rho\frac{\partial^2 E}{\partial W_j^2}} \tag{14}$$

or its first-order Taylor expansion about $\partial^2 E/\partial W_j^2 = 0$,

$$\frac{dW_j}{dT} = -\eta\frac{\partial E}{\partial W_j}\left(1 - \rho\frac{\partial^2 E}{\partial W_j^2}\right) \tag{15}$$

Equations (14) and (15) are functionally equivalent given that $\left|\rho\partial^2 E/\partial W_j^2\right| << 1$, and in particular are equivalent given the choice of ρ used in the following section.

2.3 Equivalence of Asynchronous and Synchronous Rules

Equation (15) is a *synchronous* second-order update rule that is identical to the time-averaged *asychronous* update rule of equation (11), as can setting $\rho = \eta\Delta T/2$ in equation (15). The only assumptions required to obtain this equivalence is that the time step ΔT is small enough that the approximations of equations (8), (9) and (15) are valid.

Investigating equation (15) reveals the source of the advantages enjoyed by asynchronous update rules. It can be seen that in the first stages of the convergence process (where the energy surface is normally concave upward) $\partial^2 E/\partial W_j^2$ is negative and thus larger updating steps are taken, speeding up the overall convergence rate. On the other hand during the last stages of convergence (where the energy surface is concave downward) $\partial^2 E/\partial W_j^2$ is positive and thus smaller updating step are taken, preventing undesired oscillations.

3 Summary

We have proven that the asynchronous operation of neural network may be accurately approximated by a synchronous network whose update rule incorporates information about the curvature of the energy surface. In addition, simulation using synchronous and asynchronous Hopfield networks to solve the traveling salesman problem has experimentally verified the equivalence of these two types of network.

Thus, for example, if asynchronous operation is difficult to implement because of hardware restrictions (as is true of most computers) then the synchronous update rule of equation (15) may be used instead. Or, if computation of Hessian, or $\partial^2 E/\partial W_j^2$ is expensive and asynchronous operation is cheaper, one can use an asynchronous operation instead.

Finally, use of equation (15) can allow synchronous networks to enjoy the speed and stability advantages associated with asynchronous operation. Using curvature information contained in the Hessian matrix allows improved convergence speed, especially in algorithms such as back propagation. It is also important for preventing unwanted oscillations within the network, as can easily happen in, for instance, Hopfield networks.

References

[1] J.J. Hopfield and D.W. Tank. (1985). Neural Computation of Decisions in Optimizatiom Problems. *Biol. Cybern.* **52**, 141-152.

[2] M. Takeda and J.W. Goodman. (1986). Neural networks for computations: number representations and programming complexity. *Applied Optics*, Vol. **25**, No. 18, 3033-3046.

[3] S. Becker and Y.L. Cun. (1988). Improving The Convergence of Back-Propagation Learning With Second Order Methods. *Proc. of the Connectionist Models Summer School* **CMU, Pittsburgh**, 29-37.

Bounding Analysis of a Single-Layer Feedforward Neural Network For a Binary Hypothesis-testing Problem

Garry M. Jacyna, *Member, IEEE*
Manette B. Lazear, *Member, INNS*

Mitre Corporation
Signal Processing Center
7525 Colshire Drive
McLean, VA 22102

Research efforts in the field of artificial neural networks often focus on application issues rather than on theoretical performance prediction. That approach fails to address important issues related to modeling limitations, robustness at low SNRs, optimality criteria, and optimal performance bounds. The work presented here focuses on the theoretical detection performance of a single-layer linear feedforward neural network. It is our contention that an understanding of the linear network will aid in the analysis of general nonlinear networks. Therefore, a simple one-layer network of linear processing nodes was examined.

Performance bounds for a single-layer feed forward neural network were examined for a binary hypothesis-testing problem. The network consisted of N input nodes, N + 1 adaptable weights or interconnections (including the bias weight), a summer, and a 1-bit quantizer. Each input node was linearly filtered by a set of weights which are adaptively computed from a supervised set of examplars. Known classes of signal were presented to the network where the output node was constrained to a value of ± 1 consistent with the input signal class. Weights were updated using a variation of the least-mean-squares (LMS) algorithm based on a set of training sequences which reflect both the noise-only class (H_0) and the signal-in-noise class (H_1). The bias weight is used to modify the decision boundaries. This would be referred to as an adaptive threshold in classical detection theory. We addressed two distinct problems: steady state detection performance and dynamic detection performance.

Steady-State Detection Performance

It is well known that network performance is a function of the training schedule. We first examined the connection between training and detection performance assuming that the number of training sets is infinite. This is the classical Weiner problem for steady-state weight vectors.

The summed output y(k) of the network is

$$y(k) = w(k)^T X(k) \tag{1}$$

where w(k) is the weight vector, X(k) is the corresponding training vector, the index k references a particular training set and the superscript T denotes a vector transpose operation.

We consider the following binary hypothesis-testing problem:

$$H_1 : x(k) = s(k) + N(k) \tag{2}$$

$$H_0 : x(k) = N(k) . \tag{3}$$

Here, s(k) is a deterministic but unknown signal vector and N(k) is a zero-mean multivariate Gaussian random vector with autocorreclation matrix R_n.

The optimal set of weights can be derived as a solution to the Wiener problem and shown to be equal to:

$$w_0^* = \frac{(1-2p) - p(1-p)\, s^T R_N^{-1} s}{1 + p(1-p)\, s^T R_N^{-1} s} . \tag{4}$$

$$\tilde{w}^* = \frac{2p(1-p)R_N^{-1}s}{1+p(1-p)s^T R_N^{-1}s}. \tag{5}$$

Here w_0^* is the optimal bias weight, $\tilde{w}^*$ is the optimal reduced weight vector, p is the training set probability, R_N is the noise autocorrelation matrix, and s is the deterministic but unknown signal vector.

We next derived the following expressions for the probability of detection (Pd) and the probability of false alarm (Pfa) for a deterministic but unknown signal in zero-mean bandlimited white Gaussian noise:

$$Pd = erfc_* \left\{ -\frac{1}{2}(1+\beta)\sqrt{\frac{P}{\sigma_N'^2}} \right\} \tag{6}$$

$$Pfa = erfc_* \left\{ \frac{1}{2}(1-\beta)\sqrt{\frac{P}{\sigma_N'^2}} \right\} \tag{7}$$

where

$$\beta = \alpha\gamma_p \left(\frac{P}{\sigma_N'^2} \right)^{-1} \tag{8}$$

Here α is the ratio of the training noise variance to the input noise variance, P is the total signal power $(s^T s)$, $\gamma_p = (1-2p)/p(1-p)$, and $erfc_*(x)$ is defined as the modified complementary error function.

We then examined the dependence of Pd and Pfa on the training set probability p. Three cases were considered: p = 1/2, p > 1/2, and p < 1/2. If the training sets are equally represented then p = 1/2 and $\beta = 0$ so that the probability of a miss (P_m) is equivalent to the probability of a false alarm. This is the classical *Mini-Max Criterian*. For unknown prior probabilities and known costs, the optimal strategy is to minimize the maximum Bayesian risk.

For p > 1/2, $\beta < 0$ and from (7) we conclude that the resulting Pfa is less than the false alarm probability for p = 1/2. As expected, the detection probability is also smaller. Figure 1 depicts the false alarm probability as a function of the input SNR for various values of the training set probability p.

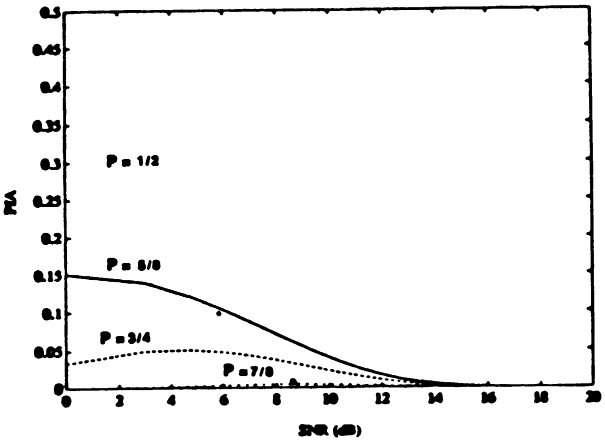


Figure 1 Probability of False Alarm (Pfa) vs SNR for various Training Set Probabilities

It is assumed that $\alpha = 1$. For p $\neq$ 1/2, Figure 1 also suggests that the false alarm probabilities can be upper bounded by a function which only depends on the training set probability. More specifically:

$$Pfa \leq erfc_* \left(\sqrt{-\alpha\gamma_p}\right). \tag{9}$$

This relationship between the maximum Pfa and the training set proability has a more classical analog. The classical *Neyman-Pearson Criterian* designs for a constant Pfa when neither the Bayes' costs nor the prior probabilities are known. For our problem, the maximum Pfa can be specified apriori. However, it can also be shown that the resulting Pd is degraded if the maximum Pfa is reduced.

These results are intuitively appealing. If p > 1/2 then class H_0 dominates the training schedule. It is reasonable to expect that the Pfa can be reduced if more noise-only training data is used. This appears to be the case. Similarly, if p < 1/2, then class H_1 dominates. We expect an increase in the Pd since more signal is present. Additionally, the Pfa degrades since less noise-only data is used. To achieve a desired upper bound on the Pfa, the training sets should be structure in the ratio p: 1–p, where p is determined from (9). That is, for N training sets, the H_0 training sets should be applied $N_0 \approx N_p$ times and the H_1 training sets should be applied $N_1 \approx N(1-p)$ times.

Figure 2 illustrates the close agreement between theory and simulation for p = 1/2:

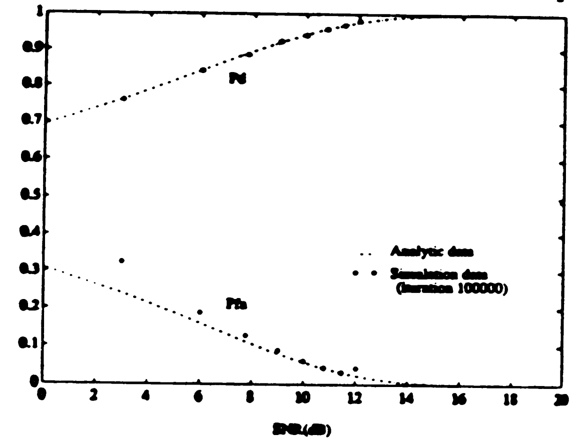


Figure 2 SNR vs Pd and Pfa at Convergence

The simulation was peformed for a single node linear neural network model using the LMS weight update algorithm. Over 10^5 training sets were used to reduce the effects of weight misadjustment. We also assumed that μ = 0.0001. Discrepancies in the Pfa values at low SNRs are reflective of the total number of training sets.

Dynamic Detection Performance

We next addressed the effects of training sequence length on network detection performance. The weight vector is a random variable related to the distribution of the training data. Under Gaussian assumptions, the first two moments of the random weight vector can be determined as a function of the learning rate parameter μ, the number of training sets M, and the first two moments of the training data. These expressions were then used to compute the Misadjustment Error (ME). This is known to be a function of the learning rate parameter and the number of components in the data vector. It relates the fractional increase in excess MS error to the optimal (minimum) MS error at a given SNR. It is a monotonically decreasing function of the number of training sets.

The Misadjustment Error is computed by determining the excess error: $\xi^2_{ms} - \xi^2_{min}$. When p = 1/2, this error is equivalent to :

$$Excess\ Error = Trace\left\{\tilde{R}\,Cov(w)\right\}, \tag{10}$$

where the Trace(A) denotes the trace of the matrix A. This is a generalization of the result quoted by Widrow. For our binary hypothesis-testing problem, the Excess Error takes on a more simplified form:

$$Excess\ Error \approx \mu\xi^2_{min}\left(1 + N\sigma^2_N + s^T s/2\right). \tag{11}$$

By definition, the ME is derived by dividing (10) by the minimum MS error to get:

$$ME \approx \mu N\sigma^2_N. \tag{12}$$

This equation is valid provided that the input SNR is much smaller than the corresponding number of input nodes N.

We can relate the Misadjustment Error to the number of training sets M. If the MS error is conditioned on the mean weight vector $\bar{w}(k)$, then:

$$\xi^2_{ms}(k|\bar{w}) = \xi^2_{min} + \{\bar{w}(k) - w^*\}^T \tilde{R}\,\{\bar{w}(k) - w^*\}. \tag{13}$$

This implies that the conditional MS error is formed by ensemble averaging over all training sets for a fixed weight vector.

This error can then be related to the average adaptation time by determining the corresponding eigenvalues of the data covariance matrix. We assume that the noise is a bandlimited white process. This implies $R_n = \sigma_N^2 I$. Under the assumption that $\sigma_N^2 \gg 1$:

$$\lambda_{max} \approx \sigma_N^2 (2 + SNR)/2, \tag{14}$$

$$\lambda_{min} = \frac{1}{2}\left(\frac{4+SNR}{2+SNR}\right), \tag{15}$$

with N-1 eigenvalues of the form:

$$\lambda_n \approx \sigma_N^2. \tag{16}$$

It is obvious that the learning rate parameter μ must be less than the reciprocal of twice the largest eigenvalue λ_{max}.

The largest time constant corresponds to the smallest eigenvalue. This time constant determines how quickly the network adapts. It is related to the learning rate parameter by the following expression:

$$T_{max} = \frac{1}{4\mu\lambda_{min}} \approx \frac{2+SNR}{2\mu(4+SNR)}. \tag{17}$$

Finally, using (12), the maximum network adaptation time can be related to the Misadjustment Error:

$$T_{max} \approx \frac{N\sigma_N^2(2+SNR)}{2ME(4+SNR)}. \tag{18}$$

We can also relate the number of training sets (M) to T_{max} by recognizing that $M \approx 4T_{max}$:

$$M \approx \frac{2N\sigma_N^2(2+SNR)}{ME(4+SNR)}. \tag{19}$$

Approximately $20N\sigma_N^2$ training sets are required at high SNR to produce a Misadjustment Error of 10%.

A more important measure of performance is the fractional false alarm error (PFE) which expresses the fractional increase in Pfa relative to a fully trained network. It can be shown that this error is also a monotonically decreasing function of the number of training sets and a function of the learning rate parameter, the number of components in the data vector, and the corresponding SNR. If we again assume a bandlimited white noise process then the following expression for PFE can be derived:

$$PFE \approx \frac{\alpha N\sigma_N^2\mu(4+SNR)}{8}, \tag{20}$$

where $SNR = s^T s/\sigma_N^2$, N >> SNR, and α is the ratio of the training noise variance to the input noise variance. It can be further shown that the maximum network adaptation time is a function of the fractional false alarm error:

$$T_{max} \approx \frac{\alpha N\sigma_N^2(2+SNR)}{16PFE}. \tag{21}$$

The total number of training sets (M) is approximately:

$$M \approx \frac{\alpha N\sigma_N^2(2+SNR)}{4PFE}. \tag{22}$$

assuming that $M \approx 4T_{max}$. The average number of training sets is approximately $30N\sigma_N^2$ if PFE = 10%, SNR = 10dB, and α = 1. Additionally, PFE = 100% implies approximately $3N\sigma_N^2$ training sets. Average adaptation time scales linearly with SNR.

Input Representation and Output Voting Considerations for Handwritten Numeral Recognition with Backpropagation

J.S.N. Jean   Y.C. Chan
Department of Computer Science and Engineering
Wright State University, OH 45435, U.S.A.

Abstract

In this paper, several input representations and output voting methods are examined to improve the recognition rate of feed-forward neural networks when applied to handwritten numeral recognition. Input representations discussed include a normal image input, an encoded (compressed) input, and a combined input. Neural networks are trained and tested for each representation on the same database. The resulting error rates on test data are 22%, 25.4% and 10% respectively. It is concluded that the incorporation of neighborhood information helps reduce the error rate. With the three neural networks in hand, an output voting mechanism can be readily applied. The resulting system has 88.8% correct recognition rate, 7% rejection rate, and 4.2% error rate. If an extra averaging operation is performed, then the recognition rate is increased to 93.2% with 6.8% error rate.

1 Introduction

Multilayer feed-forward neural networks have been shown to be promising for handwritten numeral recognition [3] [5]. For such an application, the recognition rate is a very important performance criterion. In this paper, we propose methods to improve the recognition rate. We first show that the recognition rate can be increased with proper input representation which preserves neighborhood information. Then another technique, output voting, is applied to further improve the recognition rate.

2 Input Encoding

To facilitate the simulations, a database of 1000 digits was created. It contains 100 examples of each digit, written by 10 different people. For each person, the first five digits were placed in the training set and the remaining 5 were placed in the test set. Each digit is coded as a 16×16 binary image. Some digits used are shown in Figure 1(a).

To perform the recognition, the popular backpropagation algorithm is adopted to train a neural network with 256 input units, 20 hidden units, and 10 output units. The resulting error rates for the traing data and test data are 10.2% and 22%, respectively. One interesting phenomenon we observed is that, for some input data, all the 10 output values can be fairly low. In other words, these input data, even if they do *look* like digits, are considered to match none of the ten digits. This is due to *the loss of the neighborhood information* during the input of the 16×16 data to the neural network. More specifically, the relative position of any two pixels in an image cannot be preserved since, from the point of view of each input neuron, all the other input neurons make

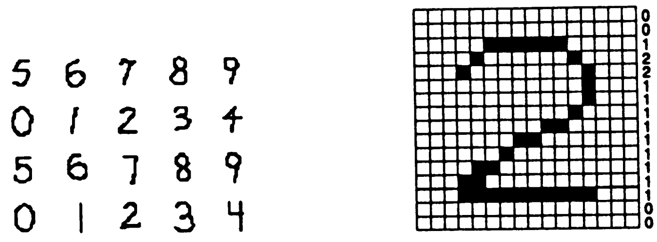


Figure 1: (a) Handwritten digit examples, (b) Encoding of a 16 × 16 image.

no difference. Therefore a slight position shift on an image would cause a similar effect as a big distortion would do. This of course is not a desirable property of an input representation.

Experiments with Input Encoding Another input representation we tested is an encoding scheme where each 16 × 16 image is encoded as four 6-digit strings [4]. The strings denote the number of black blocks when the image is viewed (and counted) *horizontally*, *vertically*, *diagonally*, and *anti-diagonally*. To compact each string into 6-digit long, redundancies within each string are removed by neglecting the initial zeros and the repetitive part and then appending some extra zeros. For example, Figure 1(b) shows a digit '2' whose horizontal view string before compaction is '0012211111111100'. This string can be compacted as '121000'. The vertical view string (from left to right) before compaction is '0002233333332000' which can be compacted as '232000'. Note that this input representation is shift-invariant and "roughly" size-invariant.

To perform the recognition, a neural network with 24 input units, 20 hidden units, and 10 output units is used. The resulting error rates are 6.2% for training data and 25.4% for test data. Compared to the previous scheme, the result is quite impressive since the number of input units has been reduced from 256 to 24.

Experiments with Combined Inputs Although the encoding process is not invertible and therefore is not totally information preserving, the encoding does preserve some partial information about the neighborhood relation which was lost using the 16 × 16 representation. In fact, the previous two neural networks do present quite different kinds of errors. To explore this property, a new input representation which simply attach the encoded strings to the 16 × 16 representation is adopted. The resulting neural network has 280 input units, 20 hidden units, and 10 output units. The error rates are 0.0% for training data and 10.6% for test data. When the number of hidden units is increased to 40, the error rate becomes 10.0% for test data (still 0.0% for training data).

3 Output Voting

In software development, it is usually difficult to discover all the bugs of a software system. To enhance software reliability, researchers have developed various methods to mask the effect of bugs. For example, N *independently developed* software versions, followed by a voter, can be used. Any bug which exists in less than half of the versions will not influence the system at all [2]. A drawback of this approach is the high cost of developing and executing the N software versions. Moreover, programs may have the same bug even when they were independently developed.

If a neural network is considered as a software system and recognition errors as design bugs, N versions of neural networks probably can be used together to mask recognition errors and therefore

Table 1: Error rates for various systems.

	Training Data	Test Data
16 x 16 Image Input	10.2%	22%
Encoded Input	6.2%	25.4%
Combined Input	0%	10.6%
Voting	0% + 1% rejection	4.2% + 7% rejection
Voting & Averaging	0%	6.8%

to improve the recognition rate. At first thought, it seems that the N neural networks can be easily developed by either using various number of hidden units or simply using different sets of initial weights during the training process. To check this out, we performed simulations and found that (1) neural networks with different number of hidden units possess similar recognition errors and (2) although different initial weights do lead to different sets of final weights, they still have quite similar recognition errors. Up to this point, the neural network seems to extract the information in a quite consistent way.

Then we tried to develop neural network versions with different input representations.[1] We inspected the recognition errors associated with the previous three networks (without input encoding, with input encoding, and with combined input) and performed a simple voting based on the three estimated digits, each from one neural networks. The majority of the three values are used as output. If the three estimated digits are totally different, then a rejection is issued. For test data set, this leads to a system with 88.8% recognition rate, 7% rejection rate, and 4.2% error rate.[2] (For training data, the system has 99.0% recognition rate and 1.0% rejection rate.)

In stead of issuing rejection signals, an alternative approach is to apply an averaging operation on the output neuron activations of the three systems when they are totally disagree (see Figure 2). In this case, the activations of three output neurons, each from one system, is averaged. The result is 10 averaged values from which the final estimated digits is produced with a winner-take-all operation. For test data, the final system has an error rate of 6.8%, significantly lower than that of any of the three subsystems (see Table 1). For training data, the system has a prefect recognition rate, 100%.

4 Conclusion

In this paper, input encoding and output voting methods are considered to reduce the error rate of the feed-forward neural networks. It is shown that the error rate is reduced when local neighborhood information is incorporated and output voting mechanisms are applied. Since the three neural networks are trained with the same backpropagation algorithm, the developing cost of the three versions is fairly reasonable. Furthermore the implementation of the three neural networks is quite straightforward with parallel processing techniques. Therefore we believe the three-version approach is justifiable for the problem, especially when the improved recognition rate can not be achieved otherwise.

[1]A corresponding notion in software fault tolerance is called *data diversity* [1].

[2]3.8% of the errors were generated when exactly two systems agree and the remaining 0.4% errors are produced when all of the three systems agree.

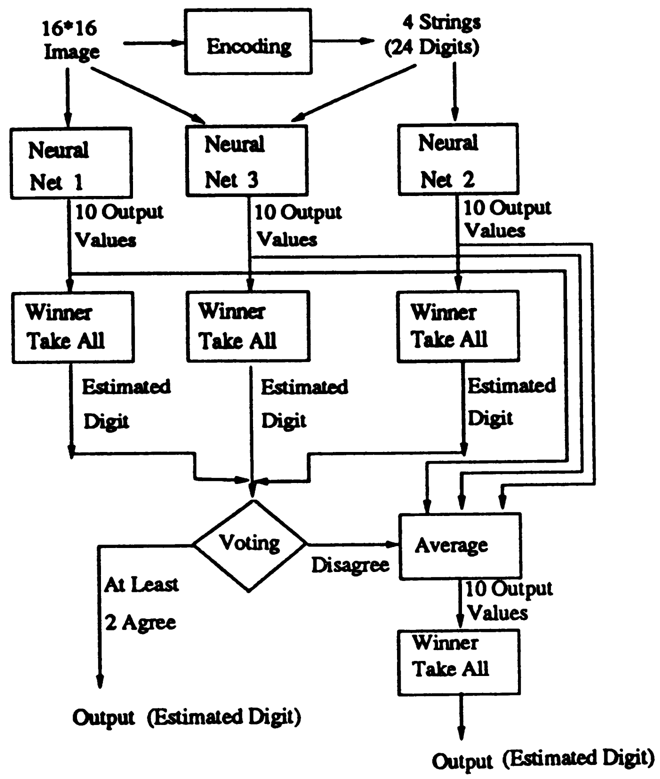


Figure 2: The system with output voting and averaging.

References

[1] P.E. Ammann and J.C. Knight, "Data Diversity: An Approach to Software Fault Tolerance," in IEEE Transactions on Computers, pp. 418–425, April 1988.

[2] A. Avizienis, "The N-Version Approach to Fault-Tolerant Software," in IEEE Transactions on Software Engineering, pp. 1491–1501, December 1985.

[3] I. Guyon et al., "Comparing Different Neural Network Architectures for Classifying Handwritten Digits," in *IJCNN89*, pp. II-127–II-132.

[4] S. Watanabe, PATTERN RECOGNITION (HUMAN AND MECHANICAL), John Wiley & Sons, 1985.

[5] K. Yamada et al., "Handwritten Numeral Recognition by Multi-layered Neural Network with Improved Learning Algorithm," in *IJCNN89*, pp. II-259–II-266.

Sejong-Net: A Dynamic Visual Pattern Recognition Neural Net

Yillbyung Lee and A-Yeun Chung
Department of Computer Science
Yonsei University, Seoul, Korea

I. Introduction

The main difference between the neural net model described here called Sejong[+](SElective Judgement Of Numerous Graphemes)-Net and most of the pattern recognition neural nets designed so far lies in that the input of Sejong-Net is not of a static visual pattern but of a dynamic one. Thus the extraction of temporal features through multilevel transformations of 2-dimensional visual input is taken as an equally important problem as the extraction of spatial features. We may think that problem solving becomes more complex as we manage more data. But, in general, more types of data may lead us to help solve a given problem more easily. Our own experiences as a machine of visual pattern recognition confirm this.

Moreover, we think the process of static character recognition can be treated as a special case of dynamic character recognition through scanning selectively over strokes according to writing order of a character. It might be considered a case of "analysis-by-synthesis". Hence Sejong-Net might be considered as a more general model of problem solver for the tasks of visual pattern recognition.

II. Overall Structure and Functions of each layer

Sejong-Net is composed of multiple layers of 2-dimensional arrays with different discrete densities (See figure 1). Each layer may contain several planes, each extracting different features. Signal flows unidirectionally from input layer to output layer(feed-forward) and operates between two adjacent layers except when the system has access to data in the past. Details of the structure and functions of each layer are described as follows:

* **IN layer and IN_1 layer:** Each element of IN(INput) layer can have a binary value and indicates whether input exists currently at each position in coordinates. (Currently mouse is used as an input device.) IN layer is represented by a 33x33 2-dimensional array. Each element of IN_1 layer marks whether there was input before one unit time at each position in coordinates, and we may consider this as a faded image.

* **TRACE layer:** Each non-zero element of this layer represents the order of positions written over the input character. This sequence may also be thought of as the sequence of tracing static character through time.

* **REC layer and REC_1 layer:** Elements of REC(photoRECepter) layer represent spatial information of points entered into IN layer as the strength of steady state as well as represent the time of their appearance as the strength of transient state. Each element in this layer may be considered as an artificial photorecepter in the retina. We define REC_1[i,j,k]=REC[i,j,k-1], and this represents the state of REC[i,j,k] at one unit time before.

*** SSF layer:** Elements of SSF(Simple Spatial Feature) layer extract various simple local spatial features. Elements of this layer can be considered a simple cell in the visual brain of an animal. Current filters for these features reflect basic local features used in structural recognition of Hangul - the Korean alphabet. SSF layer consists of sublayers called planes each of which extracts a given local feature and is composed of a 33x33 2-dimensional array with integer values.

*** STF layer:** Elements of STF(Simple Temporal Feature) layer represent temporal changes in the respective positions of the visual image. This layer may be considered as a set of transient amacrine cells or a set of ganglion cells of on-off type in the retina of an animal.

*** CSF layer:** Elements of CSF(Complex Spatial Feature) layer represents more complex spatial features constructed from features gathered from SSF planes. An element of this layer may be considered as a complex cell. CSF layer is composed of a set of 17x17 2-dimensional arrays.

*** CTF layer:** Elements of CTF(Complex Temporal Feature) layer represent termination of a stroke gathering their information from the STF layer. It consists of 17x17 2-dimensional array of integers.

*** STR layer:** Elements of STR(STRoke) layer combine elements of CSF layer and of CTF layer and then discriminate a particular stroke by indicating the start and the end of the stroke and the different types of connectivity. It consists of a set of 9x9 2-dimensional arrays.

*** GRA layer:** Elements of GRA(GRApheme) layer combine recognized strokes at various positions in the previous STR layer and then organize a particular grapheme. Hence, based on information of positions of strokes and writing orders, GRA layer recognizes a single/double vowel or a single/double consonant in Hangul.

*** SYL layer:** Elements of SYL(SYLlable) layer combine recognized graphemes in sequence to recognize an initial consonant, a vowel and a final consonant(if any) in Korean writing system, to find the syllable last entered. This layer is the final output layer.

III. Operations of Sejong-Net

*** REC layer:** Each element of REC layer with real value is updated as follows.

$$\text{REC}[i,j] \leftarrow w_1 * \text{REC}[i,j] + w_2 * \text{IN}[i,j] + w_3 * (\text{IN}[i,j] - \text{IN\_1}[i,j])$$
$$,\text{where } \text{IN\_1}[i,j,k] = \text{IN}[i,j,k-1]$$

Here, w_1 is decay rate and indicates that the responses to visual input stimulus decrease with the lapse of time.

*** SSF layer:** Current features extracted from SSF layer are SS, SE, SW, NE, NW, EE each representing different local features. Filters used for detecting these features are shown in figure 2. For example, each element of SS plane computes its value as follows :

$$SSF_{ss}[i,j] \longleftarrow \Phi_1(W_{SS}, \Theta_{SS})$$
$$\text{, where } \Phi_1(x, \Theta_{SS}) = \begin{cases} x & \text{if } x >= \Theta_{SS}, \\ 0 & \text{otherwise.} \end{cases}$$
$$W_{ss} = \sum FILTER_{ss[i,j]} * REC[i,j]$$

*** STF layer:** Element of STF layer is computed as follows :

$$STF[i,j] \longleftarrow \Phi_2((REC[i,j] - REC\_1[i,j]), \Theta_{STF})$$
$$\text{,where } \Phi_2(x, \Theta_{STF}) = \begin{cases} 1 & \text{if } x >= \Theta_{STF}, \\ 0 & \text{otherwise.} \end{cases}$$

*** CSF layer:** Elements of SS plane in the CSF layer is computed as follows :

$$CSF_{SS}[i,j] \longleftarrow \Phi_1((w_1 * SSF_{SS}[i,j] - (w_1/5) \sum_{k \neq ss} SSF_k[i,j]), \Theta_{STF})$$

*** CTF layer:** This layer has the information of terminal points (start point, end point) and middle point searching the STF. If the value of element is "1", it means a start point. If the value of element is "2", it means a middle point. If the value of element is "3", it means an end point.

*** STR layer:** Combines elements of CSF layer and terminal points of CTF layer and then find out a particular stroke.

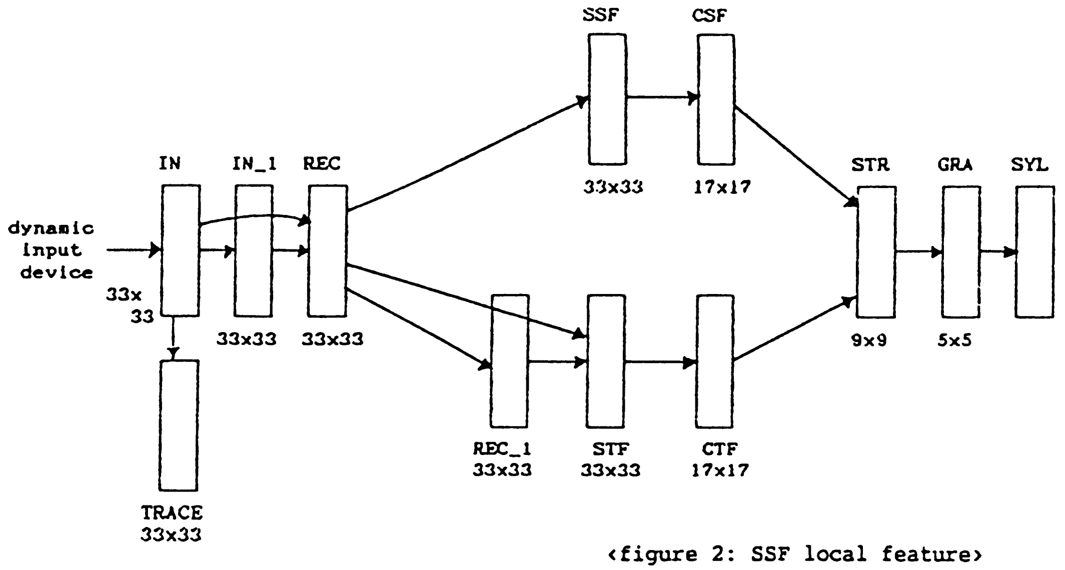
IV. Results and Conclusions

SEJONG-Net is implemented in C language on IBM PC/AT. The sytem is able to recognize written Hangul graphemes(10 simple vowels, 11 complex vowels composed of simple vowels, 14 simple consonants and 16 complex consonants composed of simple consonants) entered with mouse fairly well. We are training the system with characters written by different writers. It takes about 4 seconds to recognize the input. Figure 3 shows some intermediate layers during a simulation.

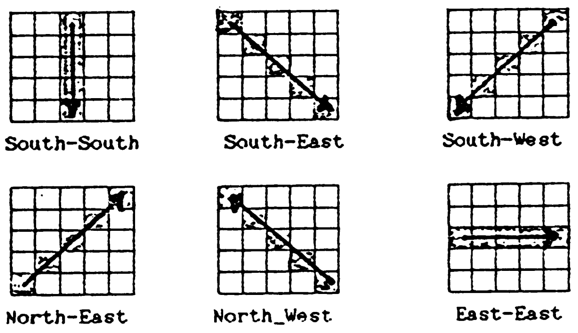
We are planning to try it with other written alphabets as well as expanding it so that it recognizes static character images.

+ "Sejong" is also the official name of the king who invented and publicized the use of Hangul - the Korean alphabet system. We thank Prof. M. A. Arbib of U.S.C. who suggested the name.

<figure 1: Overall Structure of Sejong-Net>



<figure 2: SSF local feature>



<figure 3: intermediate stages of simulation - when "ㄱ" is written>

CSF layer

```
-------------------------------------
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| 0 0 0 2 1 1 1 1 1 1 1 1 3 0 0 0 0 |
| 0 0 0 2 1 1 1 1 1 1 1 1 1 3 0 0 0 |
| 0 0 0 4 1 3 4 4 4 4 1 6 1 3 0 0 0 |
| 0 0 0 4 4 4 4 4 4 4 2 1 1 3 0 0 0 |
| 0 0 0 0 0 0 0 0 0 2 1 1 3 5 0 0 0 |
| 0 0 0 0 0 0 0 0 0 2 1 1 3 5 0 0 0 |
| 0 0 0 0 0 0 0 0 0 2 1 1 3 0 0 0 0 |
| 0 0 0 0 0 0 0 0 2 1 1 3 5 0 0 0 0 |
| 0 0 0 0 0 0 0 0 2 1 3 3 5 0 0 0 0 |
| 0 0 0 0 0 0 0 0 4 4 3 5 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 4 4 5 5 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
-------------------------------------
```

,where 1,2,3,4,5,6 represent SS, SE, SW, NE, NW, EE respectly.

CTF layer

```
-------------------------------------
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| 0 0 0 0 1 2 2 2 2 2 2 2 2 0 0 0 0 |
| 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 |
| 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 0 0 2 2 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 0 2 2 0 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
-------------------------------------
```

,where 1,2,3 represent the start, middle, the end point respectly.

Hangul Recognition using Neocognitron

Yillbyung Lee, Tae Cheon Kim and Eun Jin Kim
Department of Computer Science
Yonsei University, Seoul, Korea

1. Introduction

In this paper, we report on the initial results of applying a modified neocognitron as a pattern recognizer for Hangul-the Korean alphabet. Our network has sucessfully classified all 21 vowels(10 simple vowels and 11 complex vowels composed of 10 simple vowels) and 30 consonants(14 simple consonants and 16 complex consonants composed of 14 simple consonants) with considerable deformation or shift in position separately for now. Figure 1 shows a set of Korean graphemes. And we are working on a model that is able to recognize Korean characters(syllables) consisting of one vowel together with one or two consonants.

Our modified neocognitron consists of an input layer U_0 consisting of photoreceptor array, 3 U_s layers consisting of S cells and 3 U_c layers consisting of C cells. Figure 2 shows the overall structure of the network. The numbers "I x I x K" below the rectangulars on the figure means that "I x I" refers the array of cells and "K" refers the number of planes on a given layer. Basically structures and functions of each cells, planes and layers are the same as in Fukushima's neocognitron[1] except when explicitly mentioned otherwise.

2.Training

A learning with a teacher process has been used to reinforce the modifiable synapses. Training have been performed step by step from lower layer U_{s1} to highest layer U_{s3}. After finishing the training of lower layer, the higher layer has been trained.

Layer U_{s1} is trained to extract line components of different orientations. Each cell of this layer has a 3 x 3 receptive field. The training patterns are shown in figure 3. 20 training patterns have been used for training the layer U_{s1}. 8 half-line components are added to the 12 training patterns of Fukushima[2]. These half-line components are meant to extract the basic spatial features of termination[3]. These training patterns are useful in general because these are common basic components to all classes of visual patterns.

Figure 4 shows the training patterns used to train the planes of the layer U_{s2} for Korean vowel recognition. A cell plane is trained with 4 training patterns. These 4 training patterns have been made by shifting one pixel in each direction. The cell at the center of the cell plane to be trained is appointed as the seed cell. Good selection of training patterns is most important for the layer U_{s2} among all the layers. So, We need to know the structure of Korean graphemes. For example, Korean vowel graphemes usually consist of half-line components and "T"type components of various right angles and horizontal line or vertical line.

Hence, the training patterns for this layer have been made from deformed examples of these basic components.

Figure 5 shows the training patterns for the layer U_{s3} for Korean vowels. The standard Korean vowel patterns are used for training this layer. Because most of the distortions in shape of the input pattern have absorbed during the process in the previous stages, it was not necessary to train the deformed patterns.

3.Results and Discussions

Three level neocognitron was built and trained to classify the Korean vowels and consonants respectively. This network is modelled in PACAL on IBM PC/AT. Input image is presented by file or by a mouse interactively.

Figure 6 shows typical test vowel patterns. Most of patterns recognized incorrectly in thick box are " " and " ". These two Korean graphemes have two horizontal components and one vertical line and one "T"type component. This phenomenon is considered to be the result from generalization, which is one of the properties of neural networks. The difference between the two graphemes is the relative position of "T"type components. If the relative position of these graphemes in certain Korean character(syllable) is considered, two graphemes would be discriminated rather easily. Otherwise, it is a difficult job even for us human to distinguish the two.

One Korean grapheme has been recognized in 13 minutes on IBM PC/AT. For the recognition of a Korean character, we constructed a modified neocognitron model with backward path and it consists of larger input image. Hence the processing takes longer. Furthermore training takes for hours upto half a day. Implementation or purchase of a parallel hardware coprocessor, appears to be justified for the simulation of this type of model on a PC. As the number of training patterns of the layer U_{s2} increases, the speed of processing slows down. But the capability of recognition has been improved. Most of process time has been spent processing the layer U_{s2}. Good selection of the essential training patterns for the layer U_{s2} would reduce the overall training and processing time substantially.

We implemented neocognitron with backward path for the purpose of recognizing Korean syllables in C on microVAX. Futher work is needed to get equivalent performance for the recognition of Korean characters(syllables). Since a Korean character consists of two or three graphemes, selective attention which is one of the properties of this newer model as described by Fukushima[4] is very useful for the recognition of a Korean character.

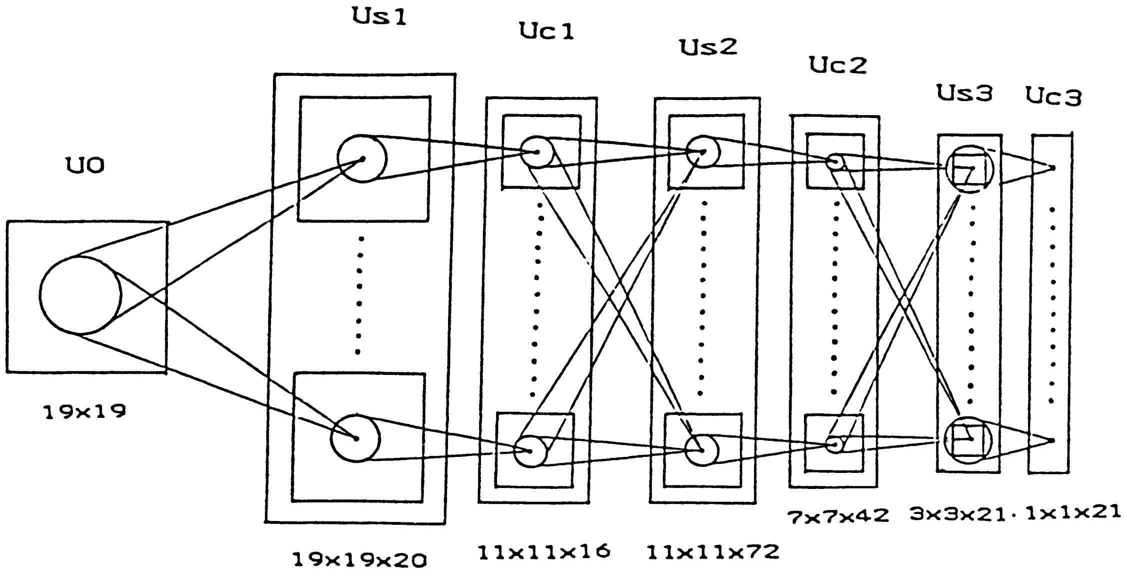
References

1. K.Fukushima, S.Miyake and T.Ito, "Neocognitron : A neural network Model for a mechanism of visual pattern recognition," IEEE Transactions on Syst. Man Cybernetics,SMC-13(5), pp. 826-834 (1983).

2. K.Fukushima, "Neocognitron : A hierarchical neural network capable of visual pattern recognition," Neural Networks, Vol.1, pp. 119-130 (1988).

3. D. Marr, "Vision" Freeman NY (1982).

4. K.Fukushima, "Neural network model for selective attention in visual pattern recognition and associative recall," Applied Optics, Vole.26, No.23, pp. 4985-4992 (1987)



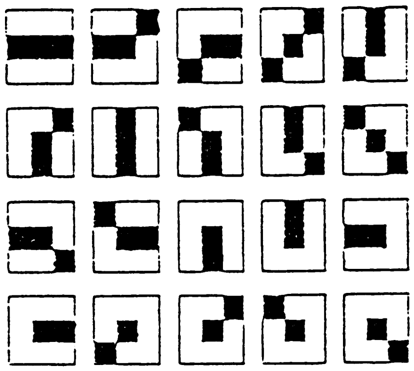
< Figure 2 : Schematic diagram of neural network model which is used in this study ("IxIxK" means that "IxI" refers the array of cells and "K" refers the number of planes) >

simple vowels (10)

ㅏ ㅑ ㅓ ㅕ ㅗ ㅛ ㅜ ㅠ ㅡ ㅣ

complex vowels (11)

ㅐ ㅒ ㅔ ㅖ ㅘ ㅝ ㅢ
ㅙ ㅞ
ㅚ ㅟ



< Figure 3 : Training patterns used to train layer U_{s1} >

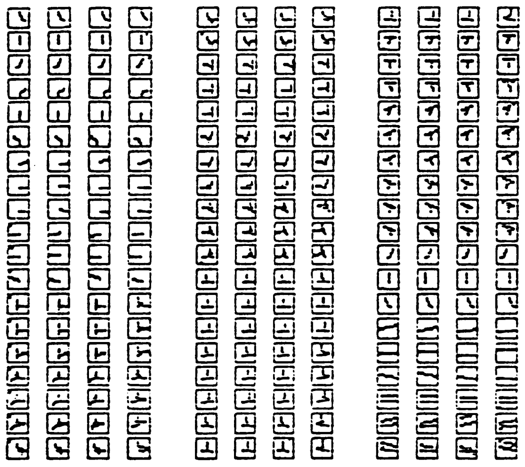
simple consonants (14)

ㄱ ㄴ ㄷ ㄹ ㅁ ㅂ ㅅ ㅇ ㅈ ㅊ ㅋ ㅌ ㅍ ㅎ

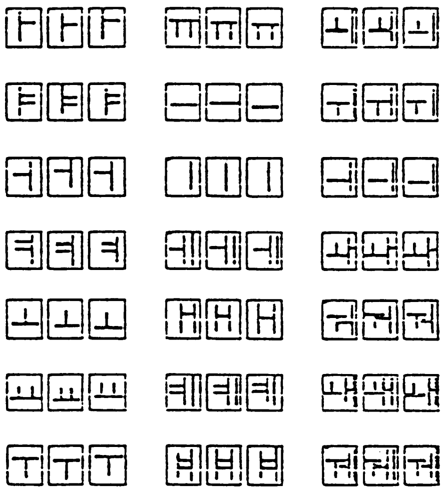
complex consonants (16)

ㄲ ㄸ ㅃ ㅆ ㅉ
ㄳ ㄵ ㄺ ㅄ
ㄶ ㄻ
ㄼ
ㄽ
ㄾ
ㄿ
ㅀ

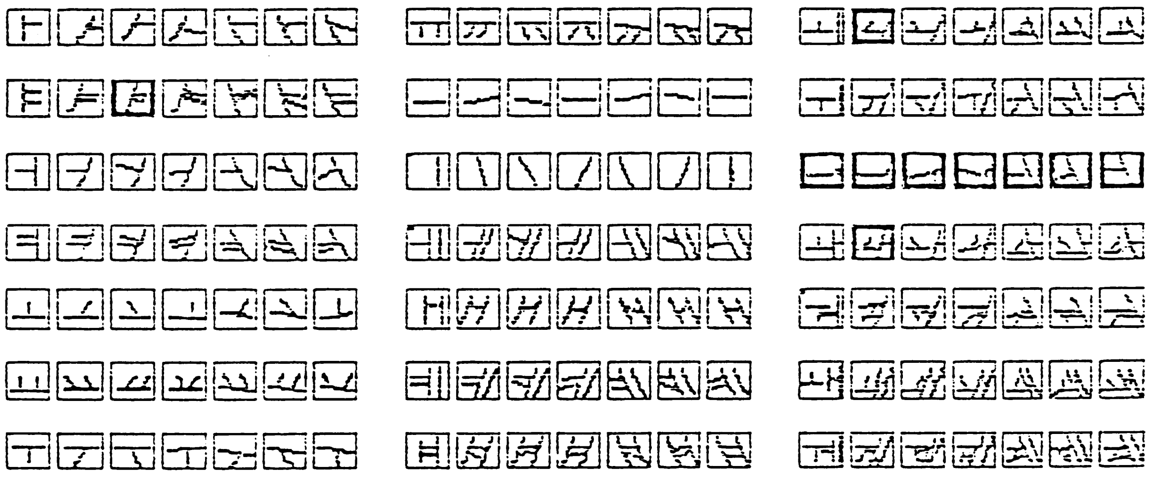
< Figure 1 : Korean characters >



‹ Figure 3 : Training patterns used to train layer U_{s2} ›



‹ Figure 4 : Training patterns used to train layer U_{s3} for the vowel recognition ›



‹ Figure 6 : Some examples of patterns which are recognized correctly or incorrectly (the patterns which are recognized incorrectly are in the thick box) ›

A New Neocognitron Structure Modified by ART and Back-Propagation

Dapeng Li and William G. Wee
Department of Electrical and Computer Engineering
University of Cincinnati
Cincinnati, Ohio 45221-0030 U.S.A.

1. Introduction

In artificial neural networks, models such as back-propagation[7], Adaptive Resonance Theory (ART)[1], and the neocognitron[2,3,4] have been studied extensively. Back-propagation networks have been also used in many practical applications. The ART systems and neocognitron systems are analogous to the human visual system. Each of them has capabilities and limitations. It is our expectation that these structure-rich models will have the capability of solving complex visual pattern recognition problems, such as these handled by the human visual system. Here we are presenting a new model that uses a neocognitron system as the basic structure. It incorporates ART to add on-line learning capability and back-propagation to add noise handling capability.

The neocognitron was chosen as the basic structure for several reasons. The neocognitron has a modular structure that enhances the flexibility of the system. The output of each module in the neocognitron brings a clear interpretation of feature abstraction making it suitable for the structural or syntactic vision pattern recognition. In addition, the neocognitron can handle the position shifting of patterns. The structure of the neocognitron has a well defined gain control system that can be used to recognize deformed patterns, overlapped patterns, and incomplete patterns. The neocognitron also has a stronger biological foundation than other neural models[6]. However, the present neocognitron system does have some shortcomings. The training algorithm of the neocognitron will only perform supervised off-line learning, and so can only operate on well-prepared training patterns.

The training algorithm of ART is unsupervised on-line learning. In ART, the on-line learning is carried out by the matching of an input pattern and an expectation pattern. These two patterns are matched through correlation with the bottom-up weights of long term memory (LTM) and the top-down weights of LTM . The learning activity is determined by the matching result. If the two patterns are matched, the learning will enhance the proper set weights of the LTM. Otherwise, the unmatched input pattern will be stored as a new pattern in the weights of the LTM if space can be found[1]. The fundamental mechanics of this matching process also exists in the structure of the neocognitron, where matching is obtained by this comparison of corresponding Uc and Wc planes. The output of the planes is used to control the gain parameters of neurons.

In the neocognitron, one Uc plane corresponds to one training pattern[5]. For similar, closely resemblant patterns, a separate Uc plane has to be assigned to each training pattern. In practical applications, the number of separate planes and processing time becomes prohibitively large and makes implementation impossible. On the other hand, a back-propagation structure has the ability to learn very closely resemblant training patterns using a three layer structure. Incorporating the back-propagation learning capability into a neocognitron system so that one plane can correspond to a class of closely resemblant patterns is a major step towards making the neocognitron system practically applicable.

Therefore, it is clear that a neocognitron structure imbedded with ART and back-propagation has the potential to be far superior to the original neocognitron model alone. In sections 2,3,and 4, a new network model is defined based on the former analysis.

2. New Model Description

Due to space limitations, mapping functions[4] are not included in this presentation, and the following formal language description is adapted:

Notations: <name>__ the component of the structure;
bold ____ terminal;
<x>|<y>__ either component x or y;
::= ______ defined as;
© _______ concatenating;
⊗ _______ interactive connecting;
' ________ inversely;

Abbreviations:
LTM_____ long term memory;
STM_____ short term memory;
F ________forward path;
B ________backward path;
other names' definition come from the neocognitron [4].

The original neocognitron model is described formally as follows:

<Network> ::= <Module>|<Network>©<Module>

<Module> ::= <Forward-path>⊗<Backward-path>
<Forward-path> ::= <Usv-plane>©<LTM><FSTM>
<Backward-path> ::= <Wsv-plane>©<LTM><BSTM>
<Usv-plane> ::= <Neuron-plane>
<LTM> ::= <Excitatory-synapses><Inhibitory-synapses>
<FSTM> ::= <Us-layer>©<Uc-layer>
<Wsv-plane> ::= <Neuron-plane>
<BSTM> ::= <Ws-layer>©'<Wc-layer>
<Excitatory-synapses> ::= **weights-array**
<Inhibitory-synapses> ::= **weights-array**
<Us-layer> ::= <Neuron-plane>|<Neuron-plane><Us-layer>
<Uc-layer> ::= <Neuron-plane>|<Neuron-plane><Uc-layer>
<Ws-layer> ::= <Neuron-plane>|<Neuron-plane><Ws-layer>
<Wc-layer> ::= <Neuron-plane>|<Neuron-plane<>Wc-layer>
<Neuron-plane> ::= **neuron-array**

The new model is described formally as follows:

<Network> ::= <Module>|<Network>©<Module>

<Module> ::= <Forward-path>⊗<Backward-path>

<Forward-path> ::= <FLTM>©<FSTM>

<Backward-path> ::= <BLTM>©'<BSTM>
<FLTM> ::= **Back-propagation-net**

<FSTM> ::= <Us-layer>©<Uc-layer>
<BLTM> ::= **Back-propagation-net**

<BSTM> ::= <Ws-layer>©'<Wc-layer>
<Us-layer> ::= <Neuron-plane>|<Neuron-plane><Us-layer>
<Uc-layer> ::= <Neuron-plane>|<Neuron-plane><Uc-layer>

<Ws-layer> ::= <Neuron-plane>|<Neuron-plane><Ws-layer>
<Wc-layer> ::= <Neuron-plane>|<Neuron-plane><Wc-layer>
<Neuron-plane> ::= **neuron-array**

The difference between the formal descriptions of the original neocognitron and new model is that the new model replaces <Usv-plane><LTM> with <FLTM>, and <Wsv-plane><LTM> with <BLTM>. The new components of the FLTM and BLTM are back-propagation networks embedded in the neocognitron system. The inputs of a module in the new network are signals put on two terminals. One is on the input layer of the FLTM of the forward path. Another is on the Wc-layer of the backward path. The outputs of a module in the new network comprise both the neuron states of the Uc-layer of the forward path, and the neuron states of the Ws-layer of the backward path. An important modification of the new model is that the input signals come from more than one of the prior modules. It means that the different levels of the abstraction can be considered together at one time to extract features. For example, to recognize the letter "Q", one feature "\" is on feature level 1, and another feature "O" is on level 2. These two features on different levels can extract the third feature "Q" when a "\" occurs at the bottom right corner of the "O". Figures 1 and 2 show one module of the new model and the neocognitron.

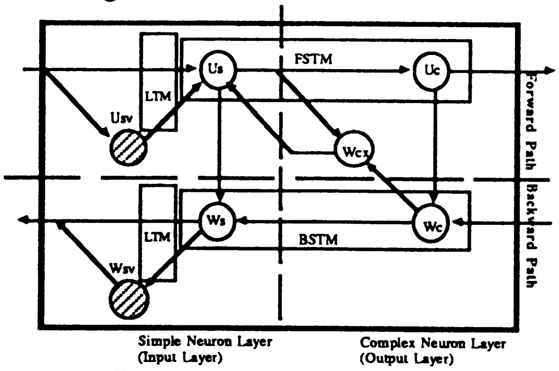


Figure 1. One Module of Neocognitron

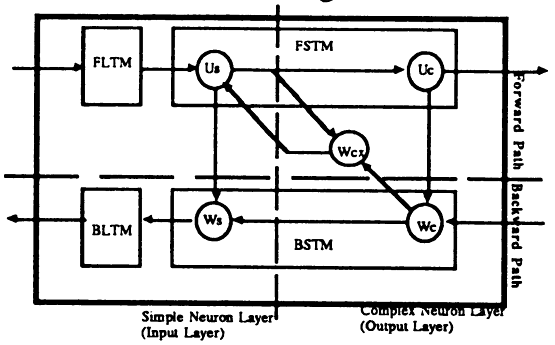


Figure 2. One module of the new model

3. Training Strategy

The training algorithm of the new model assigns values of weights in the FLTM and BLTM components. Notice that the FLTM and BLTM are both back-propagation networks, so the training algorithm of the model is similar to training back-propagation networks. The detailed structures of FLTM and BLTM will be discussed according to the back-propagation structure.

The input layer of the FLTM is a window, which will move around on the output planes of the prior modules. The hidden layer (or layers) of the FLTM will be designed differently than the input and output layers. In an example system, the hidden layer has 2/3 as many elements as the input layer has. The output layer of the FLTM is a vector, and the number of elements in the vector is as equal to the number of Us-planes in that module. Each element of the vector assigns values to Us neurons on the corresponding Us-plane. The training target always has only one non-zero element, which indicates the class to which a training pattern belongs.

The structure of the BLTM is an inverse back-propagation of the FLTM. The input layer of the BLTM is a vector connected by the Ws-planes. The hidden layer is as same as the FLTM. The output layer of the BLTM is a window to show the expected feature pattern of the input vector.

The training procedure is carried out module by module. On each module, the training of the FLTM and BLTM is executed concurrently. The input of FLTM is the target of BLTM, and the target of FLTM is the input of BLTM.

4. The Recognition Strategy

The recognition procedure of the new model is based on the neocognitron and ART systems. When a random pattern is placed on the input of the network, the pattern will be transferred module by module to the Uc-layer of the last module. In the last layer, the decision will be made by the network to either give a recognition result and stop, or send an expectation pattern back through the backward path.

In each module, the states of the Uc and Wc planes will be compared and matched. When these states are matched, the network will adjust the gain parameters according to the neocognitron strategy. Otherwise the input pattern mismatch will be considered a new pattern. The training procedure for this new pattern will start if the network has an empty plane for it. Actually, the new pattern will open a new category in the network. The criterion of matching decides the sensitivity of the network. In the example system, two planes are considered matched if 90% of their elements are identical.

5. Conclusion

This new model has the following advantages:

- enhanced training strategy comparing with the original neocognitron model
- on-line learning capability
- each plane corresponds to a class of patterns instead of one pattern
- have the same shifting tolerance and modular structure of the origianl neocognitron model
- better feature extraction ability since input signals come from different modules

The advances made here have opened up some exciting new avenues for research, and have enhanced the flexibility and power of the neocognitron neural network.

6. References

[1] Carpenter, G.A. and Grossberg, S. (1988), The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network, *IEEE Computer*, March, 1988, pp. 77-88

[2] Fukushima, K. (1988a), A Neural Network for Visual Pattern Recognition, *IEEE Computer*, March 1988, pp.65-75.

[3] Fukushima, K. (1988b), Neocognitron:A Hierarchical Neural Network Capable of Visual Pattern Recognition, *Neural Networks*,Vol.1, No.2, pp.119-130.

[4] Fukushima, K. (1986), A Neural Network Model for Selective Attention in Visual Pattern Recognition, *Biological Cybernetics*,Vol.55, pp.5-15.

[5] Li, Dapeng and Wee, William G. (1988) Feature extraction using Neural Networks, poster paper on *the first Annual Meeting of International Neural Network Society*, Boston, 1988

[6] Li, Dapeng and Wee, William G. (1989) Physiological foundation of Artificial Neural Networks in Pattern Recognition, *Proceedings of Beijing International Symposium for Young Computer Professionals*, Beijing, 1989

[7] Rumelhart, D.E., Hinton, G.E.,and Williams, R.J. (1986), Learning Internal Representations by Error Propagation, Chapter 8 of *Parallel Distributed Processing*, Vol.I, Cambridge,MA:MIT Press 1986

SEGMENT REVERSAL AND THE TRAVELING SALESMAN PROBLEM

Raymond Lister
Basser Department of Computer Science
University of Sydney NSW 2006
AUSTRALIA
ray@basser.cs.su.oz

Abstract

A neural network architecture for solving traveling salesman problems is presented, that encodes a simple form of the Lin and Kernighan rearrangement technique. It has produced high quality solutions for 30, 50, and 100 city problems. It has only $O(logN)$ interconnect.

Introduction

Hopfield and Tank (H&T) introduced the analog *N×N permutation matrix* for solving the Traveling Salesman Problem (TSP) with N cities (1985). We first introduce MCRotA, another matrix based approach. Then we modify it to form MCRevA, which implements the Lin and Kernighan technique.

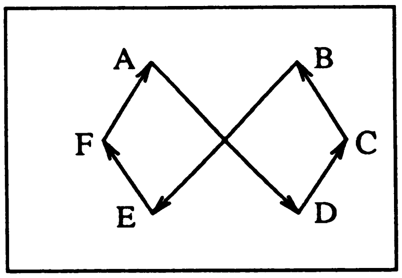
The Matrix Column Rotation Architecture (MCRotA)

MCRotA uses a discrete permutation matrix, initialized to any legal solution. The only type of change allowed, *column rotation*, preserves legality. Every unit that is off defines a unique column rotation. Let u_{ij} be such a unit, and let u_{ik} be the unit that is on in that row. If $k<j$, u_{ij} defines a column rotation to the left, in which the values of units in columns $k+1$ to j will "shift" to the column on their left, and the value of units in column k will be written into column j. If $k>j$, u_{ij} defines a column rotation to the right. Figure 2 illustrates a rotation on the path shown in Figure 1. A column rotation effectively moves one city to a new position in the path. In any permutation matrix representing a legal solution, there are N^2-N units that are off. These units represent all possible movements of city i to position j.

Since column rotation preserves legality, the energy function need only represent the distance between cities. A low energy arrangement of the matrix is found by applying rejection-less move simulated annealing (RSA) (Greene and Supowit, 1984). Consider a solution s in the solution space of a TSP with N cities. Traditional simulated annealing iteratively nominates a neighbour of s in the solution space, s', and replaces s with s' with probability p, a function of the difference in energy ΔE between s and s'. RSA considers the set of all neighbours of s, $S' = \{s'_i\}$, and makes a weighted random choice from S' to replace s, with the bias for each s_i given by its respective p_i. With MCRotA, $S' = \{s'_i: 1 \le i \le N^2-N\}$, and the column rotation defined by each inactive unit gives a unique s'_i.

MCRotA can be implemented efficiently with only $O(logN)$ connections per unit, without signal multiplexing. Only $O(logN)$ time is required for all units to calculate their respective p_i, negotiate which unit should switch on, and make the column rotation. (However, the time complexity of the cooling schedule has not been determined.) If units can communicate asynchronously, global synchronization is not required.

(a)

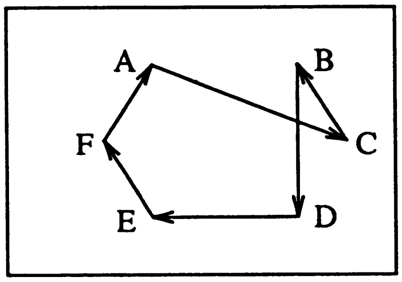


(b)

	1	2	3	4	5	6
A	1	0	0	0	0	0
B	0	0	0	1	0	0
C	0	0	1	0	0	0
D	0	1	0	0	0	0
E	0	0	0	0	1	0
F	0	0	0	0	0	1

Figure 1a-b. a A traveling salesman's path. **b** A (discrete) permutation matrix for that path.

(a)



(b)

	1	2	3	4	5	6
A	1	0	0	0	0	0
B	0	0	1	0	0	0
C	0	1	0	0	0	0
D	0	0	0	1	0	0
E	0	0	0	0	1	0
F	0	0	0	0	0	1

Figure 2a-b. a The path from Figure 1a after city "D" has been moved between cities "B" and "E". **b** The matrix for that path, with columns 2-4 of the matrix in Figure 1b rotated left.

MCRotA was evaluated on the same 30 city problem studied by H&T, on which they reported poor results. H&T also showed the best known solution for this problem, which was found using the Lin and Kernighan (L&K) algorithm. In 100 consecutive runs, MCRotA found that solution 96 times (cooling schedule: starting temperature = 5.0; cooling rate = 0.99; 30 changes per temperature; stop when all p_i truncate to zero, after about 1300 temperature iterations).

It may seem surprising that MCRotA should outperform H&T's approach, since it is conventional wisdom that analog matrix approaches are superior. One problem with H&T's approach is highly active or inactive rows and columns. Some workers have overcome this by modifying the energy function (Brandt *et al*, Szu, Van den Bout and Miller). In a sense, MCRotA implements the perfect penalty function.

The 100 runs of MCRotA gave 4 different solutions (see Figure 3). The path shown in Figure 3b crosses itself, which is suboptimal. To remove this crossover, MCRotA would need to move all the cities from one side of it to another. There would be no obvious gain until all cities had been moved. For N=30, the probability of this happening proved reasonable, but the probability decreases rapidly as N increases (see Figure 5a). The afore-mentioned analog approaches with modified energy functions were only demonstrated on small problems, and would probably suffer the same scaling problem.

The Matrix Column Reversal Architecture (MCRevA)

MCRevA is the same as MCRotA except that it reverses instead of rotates the columns. A column reversal is illustrated in Figure 4. Column reversal can remove any crossover in one step. It is the simplest type (k=2) of path rearrangement used by Lin and Kernighan (1973). MCRevA can also be implemented efficiently with only $O(logN)$ connections per unit, and requires only $O(logN)$ time to choose and make a reversal.

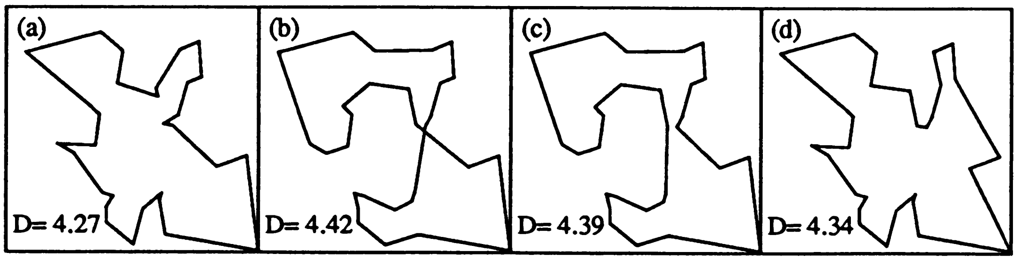


Figure 3a-d. The solutions found in 100 runs on a 30 city problem. **a** was found 96 times. This is the same solution found by the Lin and Kernighan algorithm. **d** was found twice.

Figure 5b shows a typical MCRevA solution on a 100 city problem. Table 1 shows the overall performance of both MCRotA and MCRevA on this problem, five 50 city problems, all from Durbin and Willshaw (D&W), and H&T's 30 city problem. MCRotA and MCRevA were both run 100 times (cooling schedule: starting temperature = 5.0; cooling rate = 0.9; changes per temperature = number of cities; stop when all p_i truncate to zero). The number in brackets is the number of times the best solution was found. Both approaches find consistently the best known solution to H&T's 30 city problem (found originally with the L&K algorithm, not H&T's network), but MCRevA is better than MCRotA on the larger problems. MCRevA compares favourably with the best solutions found by D&W in many trials using k=3 path rearrangement from the L&K technique (which would require $O(N^3)$ units).

Conclusion

MCRevA outperforms MCRotA (and more conventional analog matrix approaches) because it implements a domain specific heuristic (segment reversal). We conjecture that neural networks will not achieve very high performance on many other optimization problems unless they incorporate problem specific heuristics.

References

Brandt, R, Wang, Y, Laub, A, Mitra, S 'Alternative Networks for Solving the Traveling Salesman Problem and the List-Matching Problem' *IEEE International Conference on Neural Networks*, San Diego (1988), pp II-333 to II-340

Durbin, R and Willshaw, D 'An analogue approach to the travelling salesman problem using an elastic net method' *Nature* 326, pp 689-691 (16 April 1987)

Greene, J, and Supowit, K 'Simulated Annealing without Rejected Moves', *IEEE International Conference on Computer Design*, New York, 1984, pp 658-663.

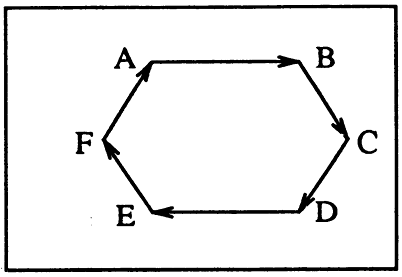
Hopfield, J and Tank, D '"Neural" Computation of Decisions in Optimization Problems' *Biol. Cybern.* 52, pp 141-152 (1985)

Lin, A and Kernighan, W 'An Effective Heuristic Algorithm for the Traveling-Salesman Problem' *Oper. Res.* 21, pp 498-516 (1973)

Szu, H 'Fast TSP Algorithm Based on Binary Neuron Output and Analog Neuron Input Using the Zero-Diagonal Interconnect Matrix and Necessary and Sufficient Constraints of the Permutation Matrix' *IEEE International Conference on Neural Networks*, San Diego (1988), pp II-259 to I-266

Van den Bout, D, and Miller, T 'A Traveling Salesman Objective Function That Works' *IEEE International Conference on Neural Networks*, San Diego (1988), pp II-299 to II-303

(a)



(b)

	1	2	3	4	5	6
A	1	0	0	0	0	0
B	0	1	0	0	0	0
C	0	0	1	0	0	0
D	0	0	0	1	0	0
E	0	0	0	0	1	0
F	0	0	0	0	0	1

Figure 4a-b. a The path from Figure 1a after segment DCB has been reversed. **b** The (discrete) permutation matrix for that path, with columns 2-4 of the matrix in Figure 1b reversed.

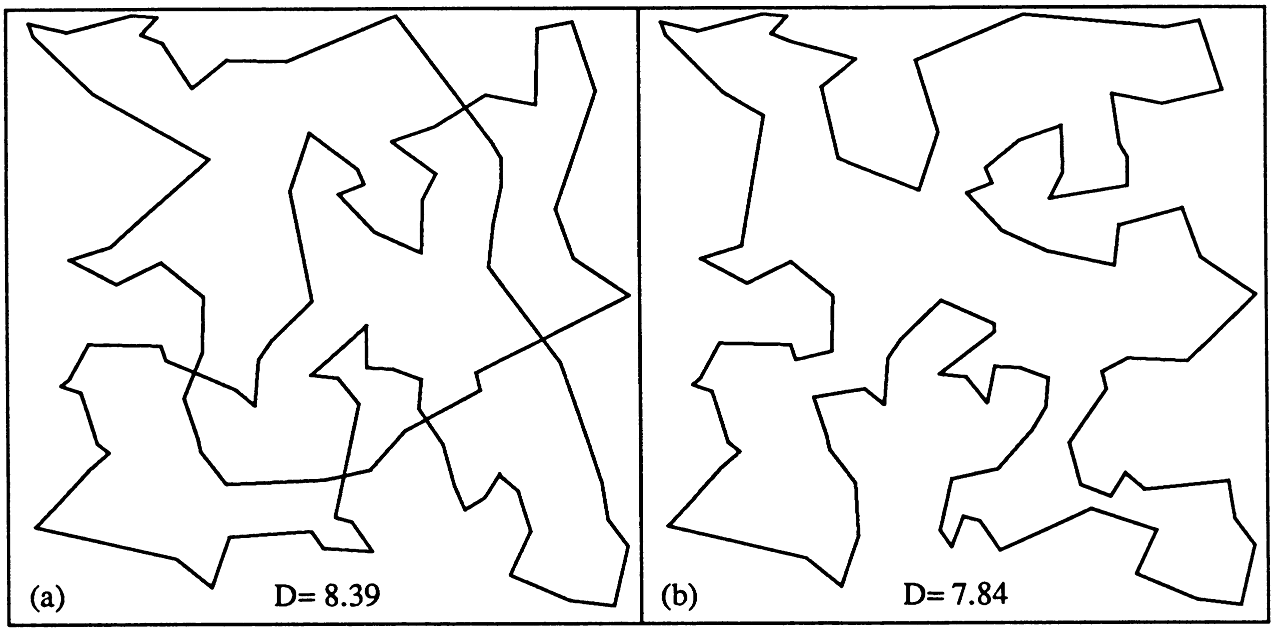


Figure 5a-b. a A typical run of MCRotA on Durbin and Willshaw's 100 city problem. **b** A typical run of MCRevA on the same problem.

authors	num. cities	author's best	MCRotA best	MCRotA average	MCRotA worst	MCRevA best	MCRevA average	MCRevA worst
H&T	30	4.26	4.268 (45)	4.404	4.849	4.268 (69)	4.286	4.434
D&W1	50	5.84	5.866 (1)	6.237	6.917	5.836 (16)	5.915	6.136
D&W2	50	5.99	5.995 (1)	6.443	6.895	5.995 (6)	6.106	6.396
D&W3	50	5.57	5.575 (5)	5.977	6.579	5.575 (15)	5.657	6.048
D&W4	50	5.70	5.698 (4)	5.967	6.476	5.698 (9)	5.808	6.078
D&W5	50	6.17	6.179 (2)	6.443	6.804	6.167 (4)	6.353	6.640
D&W6	100	7.70	7.830 (1)	8.392	8.989	7.706 (1)	7.844	8.100

Table 1. Performance of MCRotA and MCRevA over 100 runs on a variety of problems authored by Hopfield and Tank, and Durbin and Willshaw.

Stability and Temporal Pattern Recognition

*TERESA B. LUDERMIR**
Neural Systems Engineering Group
Imperial College, London SW7 2BT, England
email: JANET tbl%winge@sig.ee.ic.ac.uk

ABSTRACT

The aim of this paper is to discuss the influence of the stability property in the generalization of a neural net and consequently in the performance of the net to solve a specific task. The task we are working with is the recognition of temporal patterns and the model employed is an artificial neural net based on RAM as digital neurons[Aleksander, 79]. Some ways to control the stability of the net are presented. Experiments were done with different methods of controlling the stability and some of them are presented here.

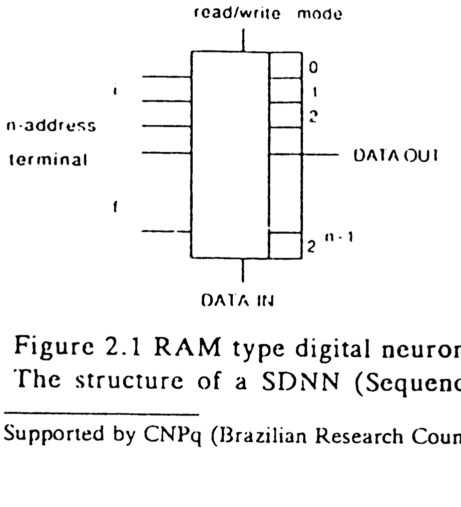
1. INTRODUCTION

There are different types of neural nets. The study of neural nets were largely originated in 1943 with the McCulloch and Pitts model of neuron [McCulloch-Pitts, 43]. They proposed a neuron model implemented by threshold logic gates, where variable input weights play a role analogous to that of synapses in natural neurons. The model used here is based on a different model called the RAM neuron model. The RAM model is based on the simple operations of a look-up table which is best implemented by random access memory (RAM) and where the knowledge is directly "stored" in the memory (the look-up tables) of the nodes during learning. Some advantages of this model are: (1) it is straightforward to implemented in hardware; (2) learning is not unreasonably slow and (3) error-correction requires only a global success signal.

The most important property of a pattern recognizer is generalization. Generalization is the ability to classify patterns others than those in the training set. RAM-nets having feedback connections between neurons have been successful with some temporal pattern recognition tasks [Ludermir, 89] but feedback machines are more sensitive to input errors than feedforward machines. However RAM-feedback nets are inherently stable adaptive structures [Fernandes, 85]. They are able to recover from input errors naturally and capable of recognizing input sequences independent of its initial state.

2. STABILITY AND GENERALIZATION PROPERTIES OF NETS WITH FEEDBACK

The type of net used in this work consists of a layer of identical RAM type digital neurons, where each of them has n-address terminals, i connected to an external matrix of binary elements and f connected to the output terminal of others neurons through clocked delay units ($n=i+f$). The RAM type digital neuron is represented in the figure 2.1.



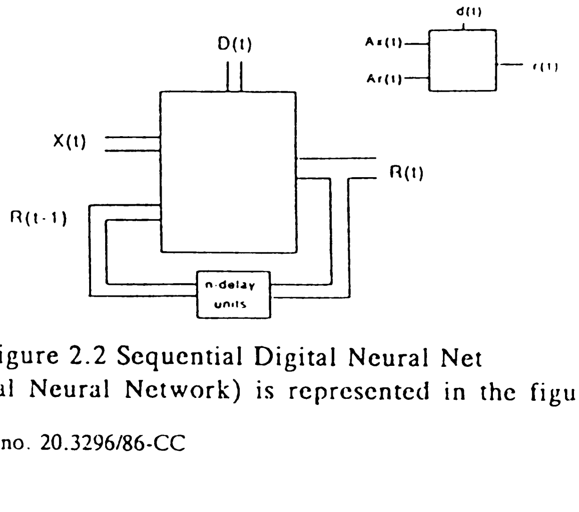


Figure 2.1 RAM type digital neuron Figure 2.2 Sequential Digital Neural Net

The structure of a SDNN (Sequence Digital Neural Network) is represented in the figure

Supported by CNPq (Brazilian Research Council) grant no. 20.3296/86-CC

2.2 where binary vectors x(t), r(t), r(t-1) and d(t) represent respectively at time t, the state of input matrix, the response, the delayed response and the 'desired' response of the neurons. The input and feedback connections are randomly generated. The SDNN is trained to anticipate its inputs, i.e. $d(t)=x(t+\alpha)$, such that $r(t)=x(t+\alpha)$ during test phase. During the training phase the net is fed with $\bar{x}\in A$ (where A is the training set) with one (or more) RAM(s) in the write mode and the memory position is changed $M_i[A_x(t),A_r(t)]=d_i(t)$ (A_x and A_r are input and feedback component respectively) for all RAMs in the write mode.

The stability property of the net is responsible for the increase of generalization. Thus it has direct influence on pattern discrimination and identification. Three types of misclassification can come from the generalization of the net. 1) rejection by doubt (intersection of two or more generalization sets); 2) unknown rejection (a pattern $\bar{x} \in L$ fall outside the generalization set); 3) error (a pattern fall within the generalization set of another category).

The stability of RAM-nets mainly depend on two parameters: a) neuron memory contents and b) feedback connection.

a) The stability can be controlled by distribution of 0's and 1's in the neuron memory. The greatest the difference between the number of zeros and ones in the neuron memory less will be the possibility of changes in the neuron output, in consequence the net will be more stable. There are two different ways of controlling the memory contents. 1) direct control: The distribution of zeros and ones is made randomly based on the difference L between zeros and ones and 2) adaptive control: The distribution of zeros and ones is made through training with any training strategy. These were first used by Fernandes in [Fernandes, 85].

b) Feedback connection influences the recover of a input error through time in a net. If we have a small feedback connection the error propagation through time is going to be reduced and the recover phase will be small. Thus the net is more stable.

3. INFLUENCE OF STABILITY IN GENERALIZATION AND IN PATTERN RECOGNITION

A classifier based on the probability $p(\bar{x}\in L_k/\bar{r})$ that the input sequence $\bar{x}\in L_k$ given that the response $\bar{r}$ occurred when $\bar{x}$ was fed into the net $0<=p(\bar{x}\in L_k/\bar{r})<=1$ was used. Although the ideal would be $p(\bar{x}\in L_k/\bar{r})=1$ for all symbols in the sequence $\bar{x}\in L_k$, this does not generally happen. What happens is that $p(\bar{x}\in L_k/\bar{r})$ changes randomly near 1 for sequences $\bar{x}$ in L_k and near 0 for sequences $\bar{x}$ not in L_k. In consequence, a measure is required which considers $p(\bar{x}\in L_k/\bar{r})$ for all input symbols in the same sequence $\bar{x}$. The measure that is being used is the continuous average

$$S_k(t)=a^{-1}\sum_{t=1}^{a} p(\,x\in L_k/r(t)),k=1,2\text{and}0\leq S_k<=1.$$

The measures we used to analyse the results are the size of state sets in each class of sequences fed into the net, the percentage of error recovery of the input sequence and sequence distinction. We do not wish neither that states sets of each class being large (for saving process time and memory) nor that the common states being in large quantity (for better distinction). The percentage of error recovery of input sequence has its importance in the states which occurs with a input sequence. Once the hamming distance between the responses of the sequence $\bar{x} \in L$ fed into the net in training and the responses of the sequence $\bar{x}' \in L$ fed into the net in testing is zero ($h(t)=h[r(t),r'(t)]=0$) all the responses will be the same.

Experiments were done to distinguish different geometric forms, such as triangles, squares and circles. The nets used were MNDS(32,2,2), MNDS(32,2,3), MNDS(32,2,4), MNDS(32,2,5) and MNDS(32,2,6). The distribution of 0's and 1's in the neuron memory were controlled by direct control and adaptive control as described in the last section. The input patterns $\bar{x}$ correspond to a sequence of eight tracking movements.

In the figure 3.1 below we show the percentage of error recovery of input sequences for experiments with five different nets. As we can see in the figure when the feedback connection increases the percentage of error recovery of input sequences decrease. With

MNDS(32,2,2) there is no difference between a direct control of stability with high value of L and adaptive control because the feedback connection is small and as consequence the net is very stable. With MNDS(32,2,3) the net with direct control with L=0, that is half of the memory contents is zero and half is one, is not able to recover from input error at all because the net was made unstable from the very small difference between the number of zeros and ones and also from the increase of the feedback connection. From MNDS(32,2,4) as the net is not very stable because of the high feedback connection only the adaptive control is successful in making the net to be able to recover from input errors.

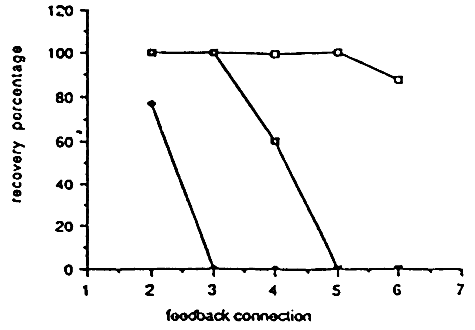


Figure 3.1 Recovery Percentage

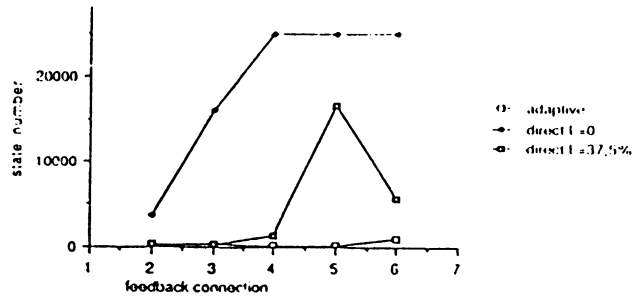


Figure 3.2 State Sets Size

It is important to observe that nets with high differences between the number of zeros and ones in memory of RAMs proceeding from direct control of stability not always result in a high input error recovery of the net. The reason for this is because this increase in the differences between zeros and ones were done randomly.

In the figure 3.2 we show the state quantity in each class of sequences for experiments with five different nets. The maximum number of states by class is the number of sequences fed into net times the number of symbols in each sequence: $500 \times 50 = 25.000$. With MNDS(32,2,2) when using adaptive control we have smaller states set, that is less computation is necessary to discriminate sequences in different classes, than when using direct control. With MNDS(32,2,3) the size of the state sets are closer with adaptive control and direct control with high value of L. From MNDS(32,2,4) we have very big state sets with direct control while with adaptive control we still have a reasonable size. The size of states set will have effect upon sequence discrimination as we will see in the next paragraph. Experiments in which the size of the state sets are big we have no discrimination among the sequences belonging to different classes.

Below we show in the figure 3.3 the discrimination capability of the net for experiments with five different nets. The discrimination capability of the net is illustrated by the difference between the values of $S_1(t)$ from sequences in the class we want to recognize and the values of $S_1(t)$ from sequences not in the class we want to recognize. When such values are negative it means that there was an intersection between the values and the number means the size of the intersection. With MNDS(32,2,2) and MNDS(32,2,3) were possible to distinguish the sequences with all ways of controlling the stability of the net because the nets are stable in all cases and the state sets are not very big.

It should be noticed that with MNDS(32,2,2) there was not to much difference in the input error recover between the two methods of stability control and the size of the states sets with adaptive control was smaller than with direct control meaning that the adaptive control is much efficient in sequence discrimination. However with MNDS(32,2,3) the size of the state sets are closer with adaptive control and direct control with high value of L, the input error recovery was similar and the discrimination power between the two ways of controlling the stability of the nets are also the same. From MNDS(32,2,4) we have values of $S_1(t)$ very close to each other mainly in the cases where the stability control was direct with $L=0$. With high value of feedback connection the net is not very good in discriminating sequences.

When we increase the feedback connection the net will remember of a input error for a longer time and we have a less stable structure. The percentage of input error recovery will deteriorate. We are going to have big state sets and consequently in some cases we are going to have difficult in temporal pattern recognition.

With unstable net the recognition of pattern is difficult also because small variations between the prototypes (patterns in training set) and pattern in the test set will result in a completely different response sequence $\bar{r}$. An unstable net will generate more unknown rejection whilst a very stable one will generate more rejection by doubt (net is not able to notice the differences between patterns) and error with the generalization.

With more stable nets we have more number of patterns not in the training set being recognized which implies in a bigger generalization. But we need to have limit in the size of the generalization set. Big generalization sets generates more mistakes with the generalization.

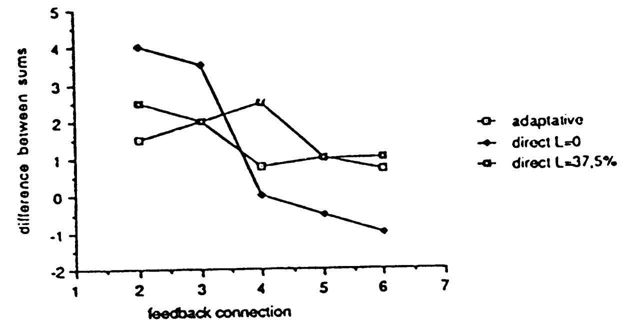


Figure 3.3 Sequence Discrimination

4. CONCLUSION

A study of the relation among the inherent stability property, the generalization and temporal pattern recognition was made. The main ideas behind this methodology are: (1) the use of a feedback RAM- net; (2) probabilistic classifier to temporal pattern recognition; (3) different methods of stability control and (4) the use of different parameters to analyse the results. Three parameters (size of state sets and their intersection, the percentage of error recovery of input sequence and sequence distinction) have been presented.

The main strengths of the method are that: (1) a response of a RAM-net with feedback carries information about the order of appearance of its input patterns; (2) the RAM-net is capable of recognizing patterns independently of its initial states even in the presence of input distortions and (3) the generalization emergent from these nets.

Many aspects of this methodology remain to be investigated since alternative training strategy were not explored. In addition probabilistic logic neuron [Aleksander, 88] could be adopted instead of RAM. The probabilistic logic neuron can avoid knowledge being overwritten in the training phase and introduces some non determinism into the system.

REFERENCES

[Aleksander, 79] Aleksander, I. & Stonham, T.J.: "A Guide to Pattern Recognition using Random Access Memories".IEE J Comp & Digital Tech 2(1), 29-40, 1979.

[Aleksander, 88] Aleksander, I.: "The logic of connectionist system" in R. Eckmiller, Chr. v.d. Malsburg, eds. Neural Computers. Berlin: Springer-Verlag, 189-197, 1988.

[Fernandes, 85] Fernandes, C.G.:"Stability Properties Inherent to Digital Neural Networks", COGNITIVA 85, Paris, 1985.

[Ludermir, 89] Ludermir, T.B.:"A Feedback RAM-Network for Temporal Pattern Recognition", Neural Systems Eng Report, Imperial College, Dept of Elec Eng 1989.

[McCulloch-Pitts, 43] McCulloch, W.S. & Pitts, W.:"A logical calculus of the ideas imminent in neural nets". Bull. Math. Biophys. 5, 1943.

AN ADAPTIVE STRATEGY TO DESIGN THE STRUCTURE OF FEEDFORWARD NEURAL NETS

B. Malakooti & Y. Zhou

Systems Engineering Department and Center for Automation and Intelligent Systems Research, Case Western Reserve University, Cleveland, Ohio 44106

INTRODUCTION

What is the minimum number of units (nodes) and the number of links (ANN topology) to represent a generalized feed-forward Artificial Neural Net (ANN) ? In this paper we develop a methodology that starts with one unit and adaptively increases its units until the minimum topology is achieved. Furthermore, we extend the model to adapt itself when the set of training patterns changes periodically, i.e., it can solve adaptive problems. This new ANN structure is hence called adaptive topology.

An ANN is a system of interconnected elements modeled after the human brain. It is specified by the net topology, node characteristics and weight training rules. There are several approaches in modeling and solving ANN [Rosenblatt, 1959; Minsky & Papert, 1988; Korhonen, 1984; Hopfield & Tang, 1985; Wang & Malakooti,1989].

Among the various topics, the topology adaptivity of an ANN is an important issue since fixed neural net topologies are not suitable for modeling the brain over time [Aoki, et al. 1988]. The topic of adaptive topology has not yet been addressed systematically in the literature. This paper provides a methodology for the topology adaptivity. The new methodology produces an adaptive strategy to expand an ANN while maintaining the previous information stored in the ANN. The goal of the adaptive topology is to obtain a suitable structure for an ANN representation and avoid local minima in training.

Almost any decision making and optimization problem is characterized by conflicting multiple objectives (criteria). For example, in selecting the best parameter for a machine, some conflicting objectives are production rate, cost of operation, and quality of products. In section 3, we demonstrate how MCDM problems can be formulated and solved by our adaptive methodology. Furthermore, we show how the problem is solved, when the decision maker's preferences of the set of training examples (patterns) are changed.

Our derivation is based on the semi-linear feedforward ANN [Rumelhart, 1986]. In section 2, we will discuss two topics: (1) Adaptive Strategy for topology selection; (2) One-dimensional Search for Learning Rate. Based on the two aspects, an algorithm for the adaptive strategy is developed in the section. In the third section, the Adaptive Strategy is applied to solve multiple criterion decision making (MCDM) problems under adaptive training sets. The conclusion is given in section 4.

2. ADAPTIVE STRATEGY

To present the Adaptive Strategy in this section, we first introduce the matrix notation for Feedforward Neural nets (FANN) . Then the theoretical basis for the Adaptive Strategy is established.

2-1. Matrix Notation for FANN

Let $X^T{}_p = [x_{p1} \dots x_{pn}]$, $Y^T{}_p = [y_{p1} \dots y_{pm}]$, $T^T{}_p = [t_{p1} \dots t_{pm}]$ and $e_p = T_p - Y_p$ be the input vector, output vector, target vector, and error vector, respectively, for a special training pattern p.

In the paper, the following evaluation function is used:

$$E = (1/2) \sum_{p=1}^{P} \sum_{b=1}^{m} (t_{pb} - y_{pb})^2 \qquad (2\text{-}1)$$

where P is the number of training patterns.

Definition 1 (Weight Matrix and Threshold Vector)

The Weight Matrix $W^{(ls)}$ and the Threshold Vector $H^{(l)}$ are defined as a $(n_l \times n_s)$ matrix and a n_l-tuple vector:

$$W^{(ls)} = [\, w^{(ls)}_{ji} \,] \qquad (2\text{-}2)$$
$$H^{(l)} = [\, h^{(l)}_1 \ldots h^{(l)}_{nl} \,] \qquad (2\text{-}3)$$
where $w^{(ls)}_{ji}$, ($j=1 \ldots m_l$, $i=1 \ldots n_s$) is the weight from the i^{th} node on layer s to the j^{th} node on layer l; $h^{(l)}_j$ ($j=1 \ldots m_l$) is the threshold inside the j^{th} node on layer l. $l \in \{1, 2, \ldots, r\}$; r is the number of layers in an ANN; and $s \in \{1, 2, \ldots, l\text{-}1\}$. Meanwhile, the parameter matrix W consists of all the Weight Matrices and the Threshold Vector.

Thus our matrix representation can be described here:

Assumption 1.The unit activation function is of the form f(net), where net is a real linear sum function .

Assumption 2. ($\partial f(\text{net})/\partial \text{net}$) $\neq 0$.

Forward Path: (Pattern Propagation)

Backward Path: (Back-propagation by using GDR)

2-1. Theory of Adaptive Strategy

As we know, one of the critical features in biological neural nets is adaptive. The following adaptive strategy can be used to design a FANN with the adaptive property. In this subsection, a sufficient condition for the optimal ANN representation is established, which is the theoretical basis for our adaptive strategy.

Definition2. (Stationary Matrix W_S)

The Stationary Matrix W_S in a FANN is a special parameter matrix W when $\Delta W_p^{(ls)} = 0$ and $\Delta H_p^{(l)} = 0$ for every l, s and p, i.e., no further change in W can be made during the training phase.

Definition3. (Input Matrix IN and Output Matrix OUT)

The elements of Input Matrix IN are defined as all possible inputs linked to the output units, and the Output Matrix OUT for the b^{th} output unit of a FANN is defined by the vector:

$$OUT^{(b)} = [\, OUT^{(b)}_{ij} \,]$$

with the element :

$$OUT^{(b)}_{ij} = \begin{cases} IN_{ij} & \text{, if the } ij^{th} \text{ node showed in IN is linked to the } b^{th} \text{ output node;} \\ 0 & \text{, otherwise.} \end{cases}$$

PROPOSITION 1: (sufficient condition for an optimal matrix)

If there exists a stationary matrix W_S in a FANN and the Output Matrix $OUT^{(b)}$ has rank P, then W_S is an optimal matrix for the b^{th} output unit of the FANN, i.e., $\Sigma_p\, (\, t_{pb} - y_{pb}\,)^2 = 0$.

Proof: (available in the full paper). #

Corollary. To avoid having a local minimizer in a FANN, at least P weights have to be linked to the output units. #

Based on the sufficient condition in the proposition, the Adaptive Strategy is described as follows:

ADAPTIVESTRATEGY: (for the fixed n and m)

(1). *InitialStep:*

Select the unit number on the first layer equal to (m-max(m,P-n-1)); besides these units, there are m output units. Connect all the (m-max(m,P-n-1)) units and n input units to the m output units so that the maximum number of links to the output units is obtained. Set r=2.

(2). *TrainingStep.*

Use the General Delta Rule (GDR) to train the FANN until a stationary matrix W_S is obtained. If $E(W_S) = 0$, Stop (we have an optimal ANN representation); otherwise, go to Adaptive Step (3).

(3). *AdaptiveStep.*

Add one more layer of at least one unit into the present FANN and connect all the units to the output units with zero weights if (a) the representation is not optimal or (b) the new patterns are available (i.e. P is changed). Then set r=r+1 and go to Training Step (2). #

Proposition 2.

If the original ANN has converged to the optimal condition, the ANN improved by the Adaptive Strategy can maintain the optimal condition.

Proof: (available in full paper)

2-3. IllustrativeExamples

In the section, three examples are given to show the performance of our Adaptive Strategy. The first example is taken from Rumelhart [1986]. The training patterns are XOR. The original net has trapped in a local minimizer. Now we add two weights from the input to the output unit without adding units. After 3000 presentations, we obtain the improved ANN. From the result, it is clear that the added weights made the improved ANN reach the optimal condition. The results are shown in Fig. 4

The second example compares the results of ANN with the optimal searching and without the optimal searching . The training patterns are still XOR. It is shown that the ANN with optimal searching can approach the optimal point more quickly (see Fig.5).

The third example shows that our ANN can be adaptive to the changes of training patterns. The original ANN has three units to represent the 30 training patterns. Later, the number of units is increased to five to represent the 60 training patterns. The results in Fig. 6 show that using the Adaptive Strategy rather than the regular method shortens the training time.

3. Using Adaptive Strategy in MCDM

In MCDM problems [see Chankong & Haimes, 1983], the decision maker's preference function is needed to represent the decision maker when there are many alternatives to be considered. However, the construction of the preference function is quite difficult. In this section, we will describe how to use ANN to obtain the function while more patterns are available.

The basic procedure in obtaining the preference function is to get a functional relationship between the desired output T and input vector $X = [x_1 \ldots x_m]$ based on the observation data $\{(T_1,X^1), (T_2,X^2), \ldots, (T_P, X^P)\}$. The error criterion is the same as eq.(2-2).

However, not all the training patterns can be obtained at the same time in the situation of MCDM, Therefore we can see the advantage of using adaptive strategy: after the patterns are increased (P increases), the Adaptive Strategy can cope with the new patterns.

4. Conclusion:

The Adaptive Strategy developed in the paper can be used in the situation when we are not sure how to construct the initial FANN or when we are facing the changing patterns. In both situations, the Adaptive Strategy can give an efficient ANN representation in terms of minimum structure, avoiding the local minima and finding the best solution. The sufficient condition provided in the paper gives a guideline to design such an ANN. The illustrative examples shew that the adaptive strategy algorithm is effective on both computational efforts and obtaining the best solution. The application example in MCDM indicates that the Adaptive Strategy is needed in the real world problems.

REFERENCE:

1. Aoki, Chiye, Siekevitz, & Philip. '*Plasticity in Brain Development*.' Scientific American, pp.56-64, 1988.
2. Chankong,V.& Haimes,Y.Y.'*Multiobjective decision making: Theory & methodology*' New York 1983.
3. Hopfield, J.J. and Tank, D.W., '*Neural Computation of Decision in Optimization Problems*', Biol. Cyber., Vol.52, no.3, pp1-25, July 1985

4. Korhonen, T., '*Self-organization and Associative Memory*', Springer Verlag,New York, 1984

5. Minsky, and Papert, S., '*Perceptrons*', (expanded edition). Cambridge: MIT Press, 1988

6. Rosenblatt, R., '*Principles of Neurodynamics*', New York, Spartan Books, 1959.

7. Rumelhart, D.E.and McClelland, J.L., '*Parallel Distributed Processing:Exploration in the Microstructure of Cognition, Vol.I and II*', MIT Press, 1986

8. Wang, J. & Malakooti, B. '*On Training of Artificial Neural Networks*', Proceedings of IJCNN. ii387-ii393, 1989.

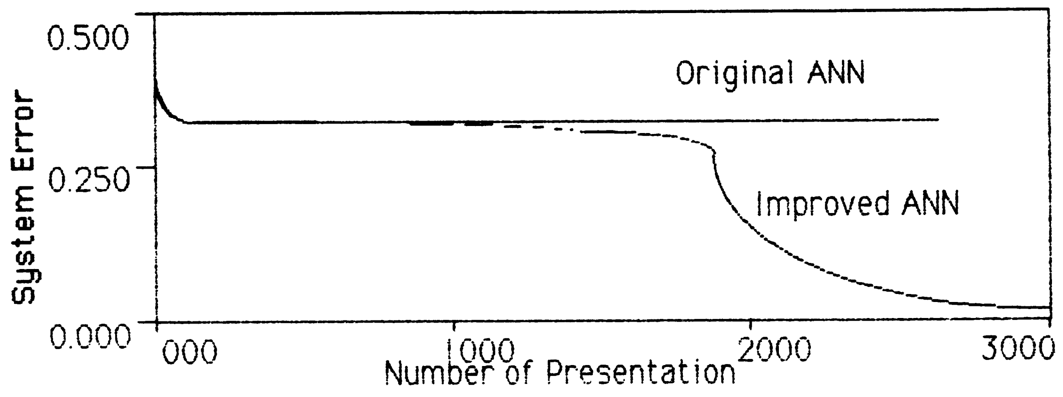


Figure 4. The Results by Adding the Connection to Output Unit

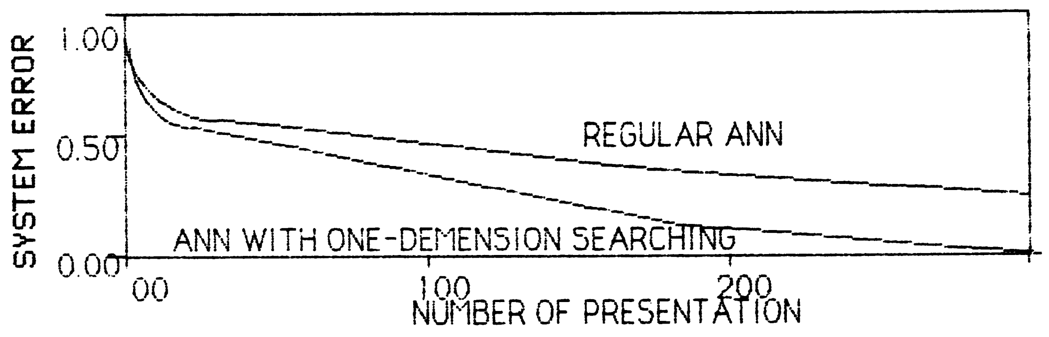


Figure 5. The Results from One-dimension Searching

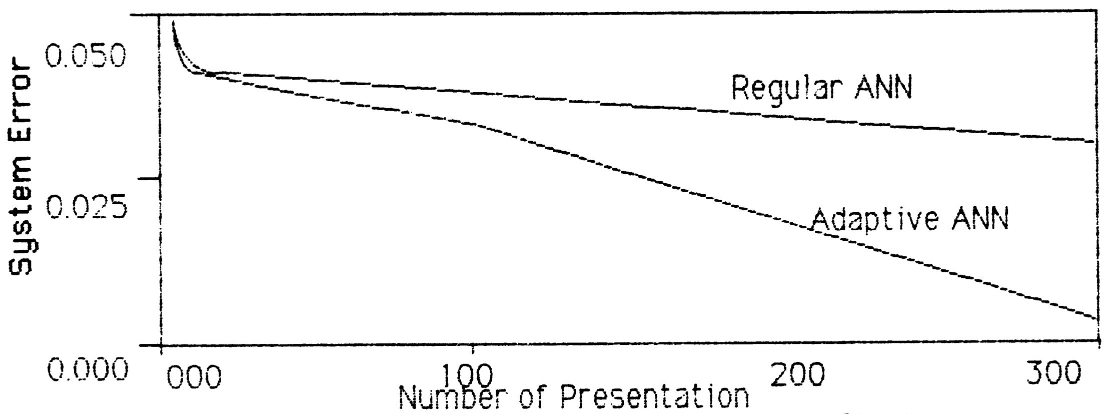


Figure 6. The Results by Using Adaptive Strategy

Multiple Descent Cost Algorithms for Standard Pattern Self-Organization

Yasuo Matsuyama

Department of Information Science, Ibaraki University,
Hitachi-city, 316 Japan (FAX: 294-37-2223)

Abstract Multiple descent cost algorithm for standard pattern self-organization is presented. This algorithm contains the author's variable region vector quantization which already generalizes the plain vector quantization. The multiple descent cost algorithm can be interpreted as a four-layer machine, where a lot of parallelism exists. Indications to further logical analysis are given.

I. Introduction A multiple descent cost algorithm is a composition of cost-decreasing mappings with different characteristics. These mappings cooperate to decrease the common cost. Therefore, the multiple descent cost algorithm is different from simple cascades of descent cost mappings. During the decrease of the cost, this algorithm self-organizes a set of standard patterns. The novel features are variable grouping of source data and various useful combinations of optimal and suboptimal mappings. Therefore, this paper extends the author's variable region vector quantization [MAT87, 88, 89] which already includes plain vector quantization as a special case. The multiple descent cost algorithm can be processed in a massively parallel way.

2. Preliminaries Let $\mathbf{x}_i$, $(i = 0, \ldots, T-1)$ be a finite set of vectors. The index i may stand for discrete time for time-series or scanned lattice of images. Each vector $\mathbf{x}_i$ is an M-dimensional tuple in $\mathbf{R}^M$. When $\mathbf{x}_i$, $(i = 0, \ldots, T-1)$ is used to design a standard pattern set, say $\mathcal{C}$, it is called training data. If the data is processed by using the designed standard pattern set $\mathcal{C}$, then it is called source data.

The data elements $\mathbf{x}_i, (i = 0, \ldots, T-1)$ are grouped into supervectors $\mathbf{v}_j, (j = 0, \ldots, J-1)$. Let the class for $\mathbf{v}_j$ be $\mathcal{G}$. Denote that class of such grouping patterns by $\mathcal{U}$. Then, $\mathbf{v}_i(u_m), (j = 0, \ldots, J-1)$ forms a partition. This grouping is done by looking at a standard pattern set to be self-organized: $\mathcal{C}[k_0, \ldots, k_{Q-1}] = \prod_{q=0}^{Q-1} \mathcal{C}^{(q)}[k_q]$, where $\mathcal{C}^{(q)}[k_q] = \{\mathbf{c}_0^{(q)}[k_0], \ldots, \mathbf{c}_{N_Q-1}^{(q)}[k_Q - 1]\}$, $\mathbf{c}_{n_q}^{(q)} \in \mathbf{R}^{K_q L_0}$, $(q = 0, \ldots, Q-1)$, $\sum_{q=0}^{Q-1} K_q = K$, and $\sum_{q=0}^{Q-1} k_q = m$.

The cost function defines similarity of two patterns:

$$D[k_0, \ldots, k_{Q-1}; m] = \sum_{j=0}^{J-1} \min_{\substack{0 \le n_q < N \\ 0 \le q < Q}} d_{\mathcal{G}}(\{\mathbf{v}_j(u_m)\}_{j=0}^{J-1}, \mathbf{w}(\prod_{q=0}^{Q-1} \mathbf{c}_{n_q}^{(q)}[k_q], \{\mathbf{v}_j(u_m)\}_{j=0}^{J-1})).$$

Here, $d_{\mathcal{G}}(\cdot, \cdot)$ is the cost function for the arguments in the class $\mathcal{G}$. The mapping $\mathbf{w} : \mathcal{G}_0 \times \mathcal{G} \mapsto \mathcal{G}$ warps a standard pattern to match to the form of $\mathbf{v}_j(u_m)$. This includes the nearest neighbor decision for generating the partitions:

$$\begin{aligned} A_{n_0, \ldots, n_{Q-1}}(u_m) = \{\{\mathbf{v}_j(u_m)\}_{j=0}^{J-1} \mid \quad & d_{\mathcal{G}}(\{\mathbf{v}_j(u_m)\}_{j=0}^{J-1}, \mathbf{w}(\prod_{q=0}^{Q-1} \mathbf{c}_{n_q}^{(q)}[k_q], \{\mathbf{v}_j(u_m)\}_{j=0}^{J-1})) \\ & \le d_{\mathcal{G}}(\{\mathbf{v}_j(u_m)\}_{j=0}^{J-1}, \mathbf{w}(\prod_{q=0}^{Q-1} \mathbf{c}_{l_q}^{(q)}[k_q], \{\mathbf{v}_j(u_m)\}_{j=0}^{J-1}), \\ & (n_0, \ldots, n_{Q-1}) \ne (l_0, \ldots, l_{Q-1})\}. \end{aligned}$$

The process of generating this partition is expressed by $\mu = \theta \circ \phi$. That is,

$$\mu(\{\mathbf{x}_i\}_{i=0}^{T-1}, \prod_{q=0}^{Q-1} \mathcal{C}^{(q)}[k_q]) = \left\{ \cdots \{\{A_{n_0,\ldots,n_{Q-1}}(u_m)\}_{n_0=0}^{N_0-1}\} \cdots \right\}_{n_{Q-1}=0}^{N_{Q-1}-1}.$$

θ is the nearest neighbor decision.

A descent cost grouping ϕ with respect to the standard pattern set $\prod_{q=0}^{Q-1} \mathcal{C}^{(q)}[k_q]$ is to find u'_m such that $D(\{\mathbf{x}_i\}_{i=0}^{T-1}, \prod_{q=0}^{Q-1} \mathcal{C}^{(q)}[k_q] | u'_m) \le D(\{\mathbf{x}_i\}_{i=0}^{T-1}, \prod_{q=0}^{Q-1} \mathcal{C}^{(q)}[k_q] | u_m)$. An important special case is the optimal grouping ϕ^* to obtain u^*_m which achieves the minimum cost. Another extremal case is the identity mapping ϕ^0. Denote the class of mappings for grouping by Φ. This is a finite set. Let $\tilde{\Phi}$ be a subset of Φ which approximates ϕ^*.

A descent cost partial update of the standard pattern set with respect to the p-th subpattern set $\mathcal{C}^{(p)}[k_p]$ is to find $\prod_{q=0}^{Q-1} \mathcal{C}^{(q)}[k'_q]$ with $D(\{\mathbf{x}_i\}_{i=0}^{T-1}, \prod_{q=0}^{Q-1} \mathcal{C}^{(q)}[k'_q] | u'_m) \le D(\{\mathbf{x}_i\}_{i=0}^{T-1}, \prod_{q=0}^{Q-1} \mathcal{C}^{(q)}[k_q] | u'_m)$ where $k'_q = k_q + \delta_{pq}$. Denote such a mapping for the update of the p-th subpattern set by

$$\psi_p(\prod_{q=0}^{Q-1} \mathcal{C}^{(q)}[k_q], u'_m) = \prod_{\substack{q=0 \\ \{p\,:\,\text{updated}\}}}^{Q-1} \mathcal{C}^{(q)}[k'_q].$$

From $A_{n_0,\ldots n_{Q-1}}(u'_m)$, we generate a coarser partition:

$$B_{n_p}(u'_m) = \bigcup_{\{\text{all except for } n_p\}} A_{n_0,\ldots,n_{Q-1}}(u'_m), \; (n_p = 0, \ldots, N_p - 1).$$

Using this partition $\{B_{n_p}(u'_m)\}_{n_p=0}^{N_p-1}$, an important special case to update the standard pattern set is obtained. That is, ψ^*_p is a mapping to find the generalized centroids of $B_{n_p}(u'_m), (n_p = 0, \ldots, N_p - 1)$. This gives $\mathcal{C}^{(p)}[k_p + 1] = \{\mathbf{c}_0^{(p)}[k_p + 1], \ldots, \mathbf{c}_{N_p-1}[k_p + 1]\}$. Denote the class of mappings for descent cost update of standard patterns by $\Psi_p, (p = 0, \ldots, Q-1)$. Let $\tilde{\Psi}_p$ be a finite subset of Ψ_p, whose elements are used to approximate ψ^*_p.

A mapping scheduler selects mappings from Φ, $\{\theta\}$ and $\Psi_q, (q = 0, \ldots, Q-1)$ so that the total cost is decreased. This selection may be either sequential or parallel. If the change of ordering of some part of the schedule does not affect the speed of the cost reduction nor the grouping pattern, then the scheduling can be executed in parallel with the same performance.

If the mapping scheduler selects the mappings autonomously without relying on outside intelligence, the entire algorithm is an unsupervised learning.

3. Multiple Descent Cost Algorithms The following is the basic design algorithm for the standard pattern set.

[Basic Design Algorithm]

Mapping Scheduler

The mapping scheduler knows the set of mappings Φ, Ψ_q, $q \in \{0, \ldots, Q-1\}$ and their finite subsets $\tilde{\Phi}$, $\tilde{\Psi}_q$, $q \in \{0, \ldots, Q-1\}$. The scheduler has either a fixed or adaptive mapping selection rule which is predetermined. In the adaptive case, the scheduling may depend on the cost, its decreasing speed, and the previous pattern set etc. as long as the independence of the outside intelligence is maintained. The mapping scheduler is settled to select every member of $\tilde{\Phi}$ and $\tilde{\Psi}_q$, $q \in \{0, \ldots, Q-1\}$ infinitely often. The scheduler has the following phases:

Initial State

The training set $\{\mathbf{x}_i\}_{i=0}^{T-1}$, a positive constant ϵ and the following initial states are given; standard pattern set $\prod_{q=0}^{Q-1} \mathcal{C}^{(q)}[0]$, grouping pattern u_0 and the initial value of the cost

$$D[k_0, \ldots, k_{Q-1}, m] = D[0, \ldots, 0, 0] = D[\text{old}] = \infty.$$

The infinity is replaced by a large number in numerical computation.
Mapping Selection for Grouping

The mapping scheduler picks up ϕ from $\tilde{\Phi}$ or from $\Phi\backslash\tilde{\Phi}$, and then apply $\theta \circ \phi$ to $(\prod_{q=0}^{(Q-1)} \mathcal{C}^{(q)}[k_q], u_m)$ in order to yield $u'_m = u_{m+1}$. Then, go to Stopping Check.
Stopping Check

If every element in $\tilde{\Phi}$ and $\tilde{\Psi}_q$, $q \in \{0,\dots,Q-1\}$ is selected since the previous Stopping Check, then let $D[\text{new}] = D[k_0,\dots,k_{Q-1}]$, which is the current cost. Then, compute and check the inequality

$$(D[\text{old}] - D[\text{new}])/D[\text{new}] < \epsilon.$$

If this inequality holds, then exit from the iteration, and adopt both $\prod_{q=0}^{Q-1} \mathcal{C}^{(q)}[k_q]$ and u_m to be the final $\prod_{q=0}^{Q-1} \mathcal{C}^{(q)}$ and u, respectively. If the inequality does not hold, then replace $D[\text{old}]$ by $D[\text{new}]$, and go to Mapping Selector for Standard Pattern Set. If there is still an unused element in $\tilde{\Phi}$ or in $\tilde{\Psi}_q$, $q \in \{0,\dots Q-1\}$ since the previous Stopping Check, then simply go to Mapping Selector for Standard Pattern Set.
Mapping Selector for Standard Pattern Set

A mapping ψ_q is selected from $\tilde{\Psi}_q$ or from $\Psi\backslash\tilde{\Psi}_q$, $q \in \{0,\dots Q-1\}$ according to the mapping scheduler's rule. Then, this ψ_q is applied to generate

$$\psi_p(\prod_{q=0}^{Q-1} \mathcal{C}^{(q)}[k_q], u'_m) = \prod_{\substack{q=0 \\ \{p:\text{updated}\}}}^{Q-1} \mathcal{C}^{(q)}[k'_q].$$

The index k'_p is increased by one to yield $k'_p = k_p + 1$ whenever ψ_p is applied. Then, go to Mapping Selector for Grouping.
This algorithm converges for any $\epsilon > 0$ after a finite number of stopping checks.

In the Basic Design Algorithm, a composition mapping $\theta \circ \phi$ is always inserted between subpattern updates of ψ_p and ψ_q so that the descent cost property is maintained. This is called concurrent subpattern update. Under specific circumstances, consecutive applications of ψ_q's still assure the descent cost property. A sufficient condition frequently met in applications is the blockwise additive cost. There is a further special case to the concurrent subpattern update:
[Design Algorithm for Concurrent Subpattern Update: Bang-Bang Switching by Cost Watcher]
Step 1

Let $\{\mathbf{x}_i\}_{i=0}^{T-1}$ be the training set which obeys to the blockwise additive cost and let $m = 0$. Let $\prod_{q=0}^{Q-1} \mathcal{C}^{(q)}[0]$ and u_0 be initial patterns. Let $D_\phi[\text{old}] = \infty$ and $D_\psi[\text{old}] < (1+\delta)D_\psi[\text{new}]$. Let $\epsilon > 0$, $\delta > 0$, and let two positive integers J and N be fixed design parameters.
Step 2

Check to see if

$$(D_\psi[\text{old}] - D_\psi[\text{new}])/D_\psi[\text{new}] < \delta$$

holds. If "no", then go to Step 4. If "yes," then update $D_\phi[\text{old}] \leftarrow D_\phi[\text{new}]$, and apply $\theta\circ\phi^*$ to $(\prod_{q=0}^{Q-1} \mathcal{C}^{(q)}[k_q], u_m)$ in order to get u'_m and $D_\phi[\text{new}]$. Then, update $D_\psi[\text{new}] \leftarrow D_\phi[\text{new}]$, and go to Step 3.
Step 3

Check to see if

$$(D_\phi[\text{old}] - D_\phi[\text{new}])/D_\phi[\text{new}] < \epsilon$$

holds. If "no," then go to Step 4. If "yes", then exit the loop and adopt $(\prod_{q=0}^{Q-1} C^{(q)}[k_q], u'_m)$ to be $(\prod_{q=0}^{Q-1} C^{(q)}, u)$.
Step 4
Update the cost $D_\psi[\text{old}] \leftarrow D_\psi[\text{new}]$. Then, apply $\prod_{q=0}^{Q-1} \psi_q^*$ to $(\prod_{q=0}^{Q-1} C^{(q)}[k_q], u_m)$ to get $\prod_{q=0}^{Q-1} C_{(q)}[k'_q]$ and $D_\psi[\text{new}]$. Then, let $m := m + Q$ and go back to Step 2.

For simplicity of the description, we used ϕ^* and ψ_q^*, $(q = 0, \ldots, Q-1)$. If the exact optimizations ϕ^* and ψ_q^*, $(q = 0, \ldots, Q-1)$ are computationally expensive, they are replaced by approximation sets:
[Design Algorithm for Bang-Bang Switching: Finite Set Approximation for ϕ^* and ψ_q^*]

(i) Instead of using ϕ^*, a finite set of descent cost grouping functions $\tilde{\Phi}$ can be found. Apply every member of $\tilde{\Phi}$. If necessary, repeat this process until the cost reduction is saturated.

(ii) Similarly to the above, find a finite set of descent cost pattern update functions $\tilde{\Psi}_q$, $(q = 0, \ldots, Q-1)$. Apply every member of $\tilde{\Psi}_q$, $(q = 0, \ldots, Q-1)$. If necessary, repeat that process until the cost reduction is saturated.

The bang-bang switching of finite set approximation is quite effective in raw data processing. The methods contain a lot of fine-grained data parallelism and coarse-grained parallelism for the mapping administration.

The self-organized standard pattern sets via above algorithms are used to process source data. This uses the basic pattern processing algorithm which is the version of the design algorithms without the standard pattern update.

4. Applications For applications, image processing is focused here. Time-series can also be processed by the multiple descent cost algorithms. Figure 1 illustrates the grouping of pixels. Here, the class $\mathcal{G}$ is a set of convex quadrilaterals. Round bullets are pixels recovered from the standard pattern set. Alteration of the quadrilateral shapes corresponds to the warping $\mathbf{w}$. When the multiple descent cost algorithm is applied on images, a quadrilateral mesh reflecting outlines is obtained. This can be used for further alteration of facial action [AIZ87]. The importance here is that the mesh pattern is self-organized.

5. Parallelism The multiple descent cost algorithms in this paper can be interpreted as four-layer networks with parallel computation. "Sequential update" [LUT89] is also obtained for our multiple descent cost algorithms. We expect neurocomputation to perform the data processing "From raw data to predicates and action" [MAT88, 89] or "Fast signal processing and logical analysis" [HOP89]. This is promoted by combination of "heterogeneous parallelisms." An example is found in [MAT88, 89].

Acknowledgements The author is grateful to Messrs. H. Nakajima, A. Matsuno of Hitachi Process Computer Engineering, Co. for their discussions. Graduate students Messrs. T. Moon, H. Wada and Y. Masuda receives thanks for their programming assistance.

References [MAT87] Matsuyama, Y., J. of IEICE, Japan, Vol. J70-A, No. 12, pp. 1830-1837 (1987): translation; Info. & Comm. Eng. of Japan, Vol. 71, No. 12, pp. 49-61 (1988). [MAT88] Neural Networks, Suppl. 1, pp. 36 (1988). [MAT89] IJCNN89, Vol. II, pp. 597. [LUT89] Luttrel, S.P., IJCNN89, Vol. II, pp. 495-498, (1989). [HOP89] Hopfield, J., IJCNN89 tutorial note 7 (1989). [AIZ87] GLOBECOM87, pp. 45-49 (1987).

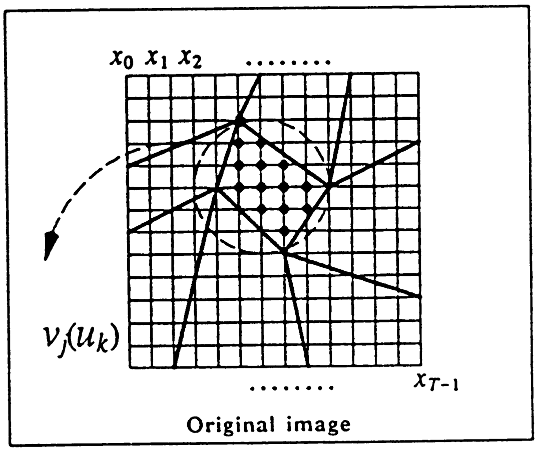


Fig. 1 Quadrilateral mesh pattern

Towards reducing the hardware complexity of feature detection-based models

Bassem Medawar and Andrew Noetzel
Polytechnic University
333 Jay St. Brooklyn, NY 11201

Abstract

A model for feature detection-based pattern recognition is presented. It attempts to improve on the hardware complexity of existing models. Traditionally, feature detection has been implemented with brute force duplication of template-based feature detectors, offering little scalability. This model eliminates the need to duplicate complex feature detectors using instead operators to transform patterns.

1. Introduction

It has been shown [1] that the brain uses feature detection, in its visual pattern recognition task. Many researches have attempted to capture the brain's pattern recognizing capability in abstract models. In their attempts, some have tried to remain faithful to the biological principles underlying the functioning and organization of the brain [2]. Others borrowed from the brain the most important principles and tried to cast them in any model that could be demonstrated to work [3,4]. The model in this paper follows the latter approach.

The work done by Fukushima will first be examined, representing earlier models of its class. A new model, which attempts to overcome their limitations, will then be described. Finally the paper will conclude with a description of future improvements to the model.

2. Earlier work

Models which loosely follow the brain's architecture, have done so based on the following elementary principles:

a) The neuron (as a threshold element) is the building block of those models.
b) The neuron's output can be viewed as a boolean corresponding to on and off activation states, or as a positive (bounded) real number representing the activation rate of the neuron.
c) The pattern recognizing network is layered.
d) Each layer contains feature detecting cells with increasing level of conceptual complexity, the higher the layer level.

Many models in the neural network literature applied those principles. Focus will be centered on Fukushima's model, because it is the most elaborate and has been shown to work with a (relatively) large retina of 128x128 pixels [5].

Fukushima's Neocognitron model [3,6] has layers with two levels each. The lower level is made up of groups of template matchers. The higher level neurons take their inputs from groups at the lower level that recognize the same object at different positions. The net effect is feature detectors tolerant of displacement and slight distortion.

Fukushima's neocognitron suffers the following problems. First, hardware is not amiable to large scale implementations: in one case [6], a simple 19x19 retina, 4 layer network implementation required over 40 thousand cells, excluding non-responding ones. Second, each learned template is duplicated in many positions after being trained in only one position: this is problematic for hardware implementations as well as being biologically implausible.

3. The model

This model [fig. 1] is based on the premise that instead of taking the feature detector to the pattern (multiplying the number of feature detectors), we bring the pattern to the feature detector (multiplying the pathways.) As the example of the neocognitron shows, duplicating complex feature detectors is costly, in terms of number of cells. What we hope to achieve is a reduction in the overall number of cells.

The retina is thus divided into several marginally overlapping receptive fields (RF's) of uniform size. Simple hard-wired operators provide a many-to-one mapping from the RF area to the feature detector. Those operators are divided into classes. For instance, on the lowest layer, the classes are displacement, scaling, and rotation. On higher layers, the classes include positional and set operators. Each class has its variations within each RF, depending on where the operator maps from, and the degree of the mapping. For example, the displacement class has variations which corresponds to the direction and the amount of the displacement.

On the next level, within a layer, feature detectors take their input from the output of the operators weighed by the optimal feature pattern. The output of those feature detectors represents the degree of success with which a particular operator maps into the optimal feature. From this large pool of feature detector outputs within a receptive field, the best variation and degree for each class is selected. Then, the optimum values across the layer from each RF are combined to choose the best class of operators. This choice represents the consensus as to which class of operators best maps into some feature.
The consensus is then fed back to lower levels, allowing each RF to reset its own image of the retina according to the new transformation. Notice that while the application of the operator class is enforced upon the layer, each RF implements the transformation in a way that generates optimal mapping.
After one class is selected within a layer, the class is then inhibited allowing another class to win in a second round. The process is then repeated until a threshold of desirable outcome is exceeded. The feature with best degree of success, can thus be said to have been recognized.

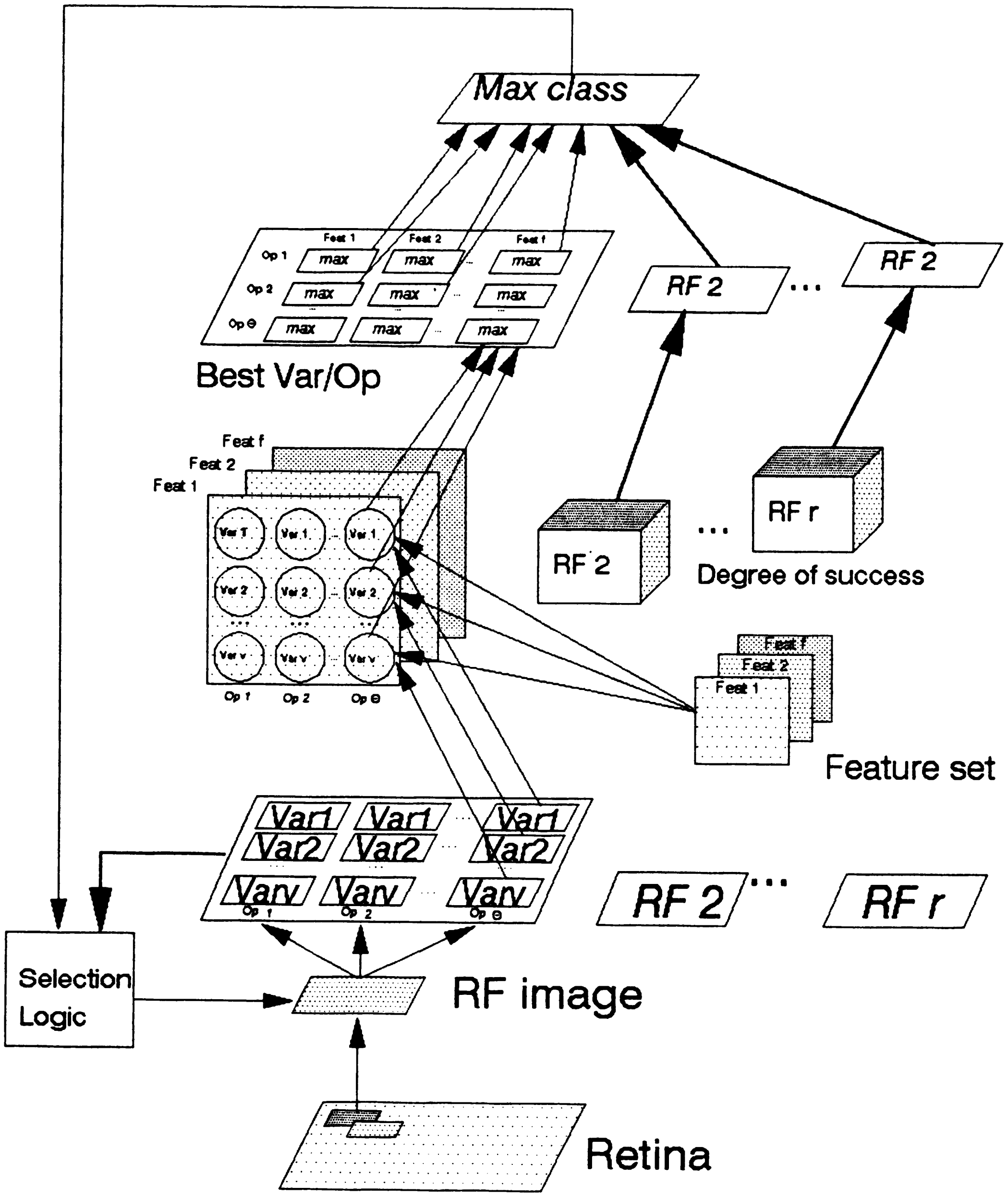


Figure 1. The model: One layer

Having recognized a feature for each RF on the first layer, the output of the first layer is fed into the second layer. A similar process of transformation and recognition is carried out in the second layer. Finally, on the topmost layer, a feature detector which conceptually represents an object is selected and the whole process terminates.

4. The model's weakness

Simulating this model necessitated additional hardware that was not originally envisioned. While its design premise is simple, the number of cells required to implement it is proportional to the number of RF's, the number of classes, the number of variations within a class, and the number of features within each layer.

The model does not lend itself to a nice and simple mathematical model to support it, and mathematical properties of the model are not practical to implement. For example, while it is mathematically sound to say that two features are different if one can not be generated from the other by applying any sequence of operators in any order. This property taken to the extreme is not practical to use in order to incorporate self-organization into the model. While that the model can be augmented with learning rules that change the weights on the feature detectors, it fails to address how a whole new class of operators can be learned.

The model has been shown in practice to fail under certain circumstances: the wrong sequence of transformations is applied leading to either faulty recognition or no recognition at all. This is a result of the lack of communication between neighboring RF's.

5. Conclusion and future work

A feature detection-based model was presented that was designed to address the limitations of previous models. The model was developed from an innovative idea. Although the model achieved its objective of lesser overall hardware complexity, it had few limitations of its own.

Currently, work is in progress on a new model. This model incorporates communication between neighboring RF, coupled with hill climbing techniques to pick the best transformation to apply within an RF image. In effect, this will result in a reduction in the number of pathways as only few transformations will be implemented at a time.

References

[1] Kuffler S. W., Nicholls J. G., and Martin A. R., **From Neuron to Brain**, 2nd Ed. Sinauer Associates Inc. Publishers, Sunderland, MA, 1984.

[2] Linsker R., Self-Organization in a Perceptual Network. **IEEE Computer**, March 1988, pp. 105-117.

[3] Fukushima K., and Miyake S., Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. **Pattern Recognition**, Vol. 15, No. 6, pp. 455-469, 1982.

[4] Widrow B., Adaline and Madaline - 1963. **IEEE 1st International Conference on Neural Networks**. Vol. 1, pp. 143-157.

[5] Menon M. M., and Heinemann K. G., Classification of patterns using a self-organizing neural network. **Neural Networks**, Vol. 1, pp. 201-215, 1988.

[6] Fukushima K., A neural network for visual pattern recognition. **IEEE Computer**, March 1988, pp. 65-74.

PARSIMONY IN NEURAL NETWORKS

William S. Meisel
Speech Systems Incorporated
18356 Oxnard Street
Tarzana, California 91356
(818)881-0885

INTRODUCTION

Decision trees and neural nets have been used as a form of piecewise-linear discriminant functions in pattern recognition [e.g., Breiman et al 1984, Meisel 1972]. Decision trees are sometimes assumed to have an advantage in that they can be executed very efficiently on a sequential computer. Neural networks are generally assumed to be designed for highly parallel systems, although currently most are simulated on sequential processors. Parameter estimation for decision networks tends to be sequential -- one node at a time -- as well, while parameter estimation for neural networks tends to be parallel -- all parameters estimated simultaneously. Information measures are often used in deriving decision trees [e.g., Quinlan 1986]; they are beginning to be applied to neural networks [Bishsel and Seitz 1989]. Since decision trees can be formulated as neural nets, neural net criteria and methods can be applied to decision trees. In this paper, we show that decision trees and fully parallel neural networks are the extremes of a spectrum of neural networks that differ in a systematic way.

A decision tree can be viewed as a special case of a "parsimonious" neural network, one that can be evaluated exactly for most cases with the calculation of only a fraction of the neural elements. A highly parsimonious network usually corresponds to a less-distributed recognition algorithm; in a parsimonious network, elements tend to "specialize" more, allowing a targeted use of samples, making possible some efficiencies in parameter estimation. The case for a distributed parallel system is strong [Rummelhart and McClelland 1986]; but where networks are simulated and where data and/or where computer time for network derivation are limited, parsimony can be advantageous.

This paper provides an algorithm for fully determining the output of a neural net without evaluating all of the elements in the net; it is applicable to two-layer neural nets. The average number of first-level neural elements which must be evaluated, expressed as a fraction of the first-level elements present, is a measure of the parsimony of the network. Parsimonious neural nets have been successfully applied in a commercial speech recognition system, the Phonetic Engine®, which evaluates neural nets for phoneme classification in real time on a single microprocessor [Meisel et al 1989].

NETWORK REPRESENTATION

In this note, we consider a two-layer neural network. The input is $\mathbf{x} = (x_1, x_2, ..., x_X)$. These variables are inputs to Y neural elements with outputs $y_1, y_2, ..., y_Y$:

$$y_i = F_i(\mathbf{w}\bullet\mathbf{x}') , \tag{1a}$$

$$\text{where } \mathbf{w}\bullet\mathbf{x}' = \sum_{i=1}^{X} w_{il}\, x_l + w_{i,\,X+1}, \text{ i.e., } \quad \mathbf{x}' = (x_1,x_2,...,x_X,1). \tag{1b}$$

An example of F_i is

$$F_i(x) = F\left(x;\beta_{i1},\beta_{i2}\right) = \begin{cases} 1 & \text{if} \quad x \geq \beta_{i1} \\ f(x) & \text{if} \quad \beta_{i1} < x \leq \beta_{i2} \\ -1 & \text{if} \quad x < \beta_{i2} \end{cases} \tag{2}$$

where f(x) is monotonically increasing and takes values in [-1, 1]. Ideally, f(x) is continuous, $f(\beta_{i1}) = -1$, and $f(\beta_{i2}) = 1$, making y a continuous function of x. However, $f(x) = 1$ and $\beta_{i1} = \beta_{i2} = 0$ is the important special case of a "hard-limiter." We assume in this paper only that F_i is chosen so that $|y_i| \leq 1$. The region (β_{i1}, β_{i2}) can be viewed as the region of uncertainty. We allow for the possibility that the uncertainty region will be different for different neural elements and not necessarily symmetrical. The output layer has Z elements with output $z_1, z_2, ..., z_Z$:

$$z_i = \begin{cases} 1 & \text{if } \mathbf{a}_i \bullet \mathbf{y} \geq \alpha_i \, \|\mathbf{a}_i\| \\ 0 & \text{otherwise} \end{cases} \tag{3a}$$

$$\text{where } \mathbf{a}_i = (a_{i1}, a_{i2}, ..., a_{iY}) \,, \quad \|\mathbf{a}_i\| \equiv \sum_{j=1}^{Y} |a_{ij}|, \quad \mathbf{a}_i \bullet \mathbf{y} = \sum_{j=1}^{Y} a_{ij} \, y_j \,. \tag{3b}$$

An additional assumption is $-1 \leq \alpha_i \leq 1$. (3c)

Equation (3c) does not constrain the generality of the decision function since $-\|\mathbf{a}_i\| \leq \mathbf{a}_i \bullet \mathbf{y} \leq \|\mathbf{a}_i\|$, given that $|y_i| \leq 1$.

PARSIMONIOUS NETS

As α_i approaches 1 with fixed a_{ij}, there will be fewer values of **y** which cause z_i to be 1 [Meisel 1968]. In fact, if $\alpha_i = 1$, z_i in equation (3) will be 1 if and only if $y_j = 1$ for all j where $a_{ij} = 1$ and $y_j = -1$ for all j where $a_{ij} = -1$. As a rule of thumb, the closer Min α_i is to 1, the more parsimonious the net.

The output values z_i can all be evaluated without necessarily evaluating all the neural elements y_j. The algorithm for doing so is based on observing that the output value for z_i may be determined without calculating all values of y_j. Suppose y_K is known. Then

$$\mathbf{a}_i \bullet \mathbf{y} \leq \sum_{j=1}^{K-1} |a_{ij}| + a_{iK} y_K + \sum_{j=K+1}^{Y} |a_{ij}| \equiv K_i \tag{4}$$

since $a_{ij} y_j$ can get no larger than $|a_{ij}|$.

If $K_i < \alpha_i \, \|\mathbf{a}_i\|$, then equation (3) produces $z_i = 0$, irrespective of the values of $y_1, ..., y_{K-1}, y_{K+1}, ..., y_Y$.

Similarly

$$\mathbf{a}_i \bullet \mathbf{y} \geq -\sum_{j=1}^{K-1} |a_{ij}| + a_{iK} y_K - \sum_{j=K+1}^{Y} |a_{ij}| \equiv M_i \,. \tag{5}$$

If $M_i \geq \alpha_i \, \|\mathbf{a}_i\|$, then $z_i = 1$. Thus it may be possible to calculate the full output of a two-layer neural net, formulated as in (1)-(3), without calculating all the intermediate neural elements y_j. It is convenient to re-write M_i and K_i as follows:

$$K_i = \|\mathbf{a}_i\| + a_{iK} y_K - |a_{iK}| \tag{6}$$

$$M_i = -\|\mathbf{a}_i\| + a_{iK} y_K + |a_{iK}| \tag{7}$$

We apply this idea iteratively, as follows:

<u>Initialization</u>

Initialize $M_i(k)$ and $K_i(k)$:

$$M_i(0) = -\|\mathbf{a}_i\| \,; \quad i = 1, ..., Z \tag{8a}$$

$$K_i(0) = \|\mathbf{a}_i\| \,; \quad i = 1, ..., Z \tag{8b}$$

Initialize sets F(k), T(k), and J(k):

$$F(0) = T(0) = \varnothing \tag{8c}$$

$$J(0) = \{1, 2, ..., Y\} \tag{8d}$$

<u>Iteration</u>

Start with k =1 and increment k by 1 until stopping rule is met.

For each k,

List the output nodes not yet fully evaluated:

$$R(k) = \{1,2,\ldots,Z\} - T(k-1) - F(k-1) \tag{9}$$

Find the neural element K not yet calculated which will affect the most z_i, that is, set K equal to the j for which there are the most non-zero values of a_{ij} for $i \in R(k)$. If more than one j satisfies this condition, arbitrarily choose the smallest.

Find those output elements with a value now fully determined, the set T for those determined as 1 and the set F for those determined as 0:

$$M_i(k) = M_i(k-1) + a_{iK}y_K + |a_{iK}| \ , \ i \in R(k) \tag{10}$$

$$K_i(k) = K_i(k-1) + a_{iK}y_K - |a_{iK}| \ , \ i \in R(k) \tag{11}$$

$$T(k) = T(k-1) \cup \{ i \mid M_i(k) \geq \alpha_i \|a_i\| , i \in R(k) \} \tag{12}$$

$$F(k) = F(k-1) \cup \{ i \mid K_i(k) < \alpha_i \|a_i\| , i \in R(k) \} \tag{13}$$

Update the set of first-level elements not yet evaluated:

$$J(k) = J(k-1) - \{ K \} \tag{14}$$

Stopping rule: If $F(k) \cup T(k) = \{1, 2, \ldots, Z\}$, stop. (15)

Result: $z_i = 0$ for $i \in F(k)$; $z_i = 1$ for $i \in T(k)$ (16)

This algorithm will not always result in evaluation of the minimal number of nodes possible for an arbitrary network. For example, suppose we have a net as follows: $z_1 = 1$ for $2y_1 + y_3 \geq 2$; $z_2 = 1$ for $2y_2 + y_3 \geq 2$. Then, z_1 is determined entirely by y_1; z_2 is determined entirely by y_2. The variable y_3, being in both equations, would be evaluated first by the algorithm, although it need not be evaluated at all. This is a pathologic case since the third element could be removed from the network without any effect, but it shows that the algorithm cannot deal with an arbitrary network optimally.

MEASURING PARSIMONY

Let us assume that N(x) is the minimal number of first-level elements which must be evaluated to fully determine the output of the network given x. The parsimony of a two-level neural net can be measured as the expected value of N(x) divided by the number of first-level elements. An example is given in the next section.

DECISION TREES AS PARSIMONIOUS NETS

Decision trees [e.g., Breiman et al, 1984; Quinlan 1986] are an important special case of neural networks in general and of parsimonious decision networks in particular. The two-level network of equations (1), (2) with the hard-limiter case, and (3) corresponds to a decision tree if parameters $\mathbf{a}_i$ are chosen properly and if $\alpha_i=1$ for all i. Figure 1 illustrates a binary decision tree and its equivalent as a decision network.

For any given tree structure, an equivalent network is constructed by associating a neural element with each decision node. That element is connected only to elements which are in the path to the equivalent terminal node. The weight for each connected element is 1 if the path was taken due to a "yes " answer, and -1 if taken due to a "no" answer. The threshold of the output element is $\|\mathbf{a}_i\|$. Then, if a given terminal node is reached by the path requiring that decision nodes 1, 2, and 3 are true and nodes 4 and 5 are false, that terminal node is represented by an output element:

$$z = \begin{cases} 1 & \text{if } y_1 + y_2 + y_3 - y_4 - y_5 \geq 5 \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

By the nature of this construction, only one z_i will be 1 at a time, corresponding exactly to the terminal node of the equivalent tree. For networks to be equivalent to decision trees , it is fully general to constrain a_{ij} to the values 1, 0, and −1.

In figure 1, if the question in decision node 1 (with weights w_1) is answered "no," then the equation for decision node 3 is never calculated. Similarly, if the algorithm of (8)-(16) is applied to the equivalent network for the same x, network element 3 will not be computed. More generally, in a symmetrical binary decision tree with 2^n terminal nodes (n-1 levels), only n decision nodes out of 2^n - 1 decision nodes are evaluated to make a decision. Thus, in a nine-level tree, only 9.8% of the nodes are evaluated. (Since in this case N(x) = n for all **x**, the parsimony is 9.8%.) If the equivalent network were evaluated fully, over 90% of the work would be redundant.

A classical decision tree making firm decisions can be most effectively calculated in the usual way; equations (8)-(16) add nothing to this case. The major relevance of this special case is the clear example it provides of the efficiencies possible.

FUZZY DECISION TREES

Decision trees can be allowed to make fuzzy rather than firm decisions at each node, (i.e., to use a non-zero uncertainty interval rather than a hard limiter); the interval (β_{i1},β_{i2}) for node i can be considered an uncertainty interval. We can use the algorithm: If for a sample **x**, $y_i = 1$, evaluate the right descendant node only; if $y_i = -1$, evaluate the left descendant node only; otherwise, evaluate both descendant nodes.

Decision trees are generally *constructed* node by node from the root node. When this is the case, we should avoid evaluating paths through the tree that are "ruled out" by an earlier node with a zero score for the path; ruled-out nodes will not have been estimated with appropriate samples. In general, it is possible for the algorithm (8)-(16) to choose a ruled-out path for a fuzzy tree. This problem can be avoided as follows: If $y_j = 0$, then $\mathbf{a}_i \bullet \mathbf{y}$ can get no larger than $\|\mathbf{a}_i\| - |a_{ij}|$. If we choose

$$\alpha_i > \frac{\|\mathbf{a}_i\| - \min\limits_{j \mid a_{ij} \neq 0} |a_{ij}|}{\|\mathbf{a}_i\|}, \qquad (18)$$

then no "ruled out" paths will be chosen. If $a_{ij} \in \{-1,0,1\}$, then of course Min $|a_{ij}|$ over non-zero a_{ij} is 1. Equation (18) is fully determined by the structure chosen for the second level of the net. In using neural net techniques to get decision tree parameters, the a_{ij} can be chosen to be a particular structure and held fixed while the $\mathbf{w}_i$ are optimized.

REFERENCES

Bichsel, M. and Seitz, P. (1989). Minimum class entropy: A maximum information approach to layered networks, *Neural Networks*, Vol. 2, pp. 133-141.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Belmont, CA: Wadsworth International Group.

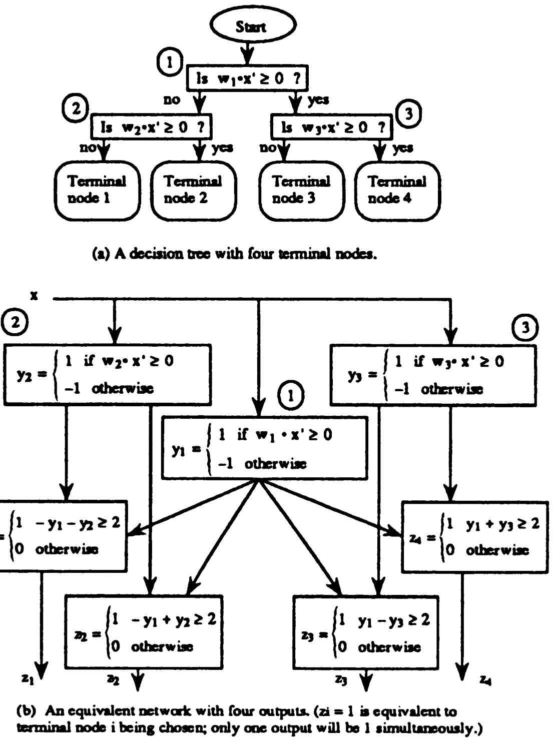
Meisel, W.S. (1968). Variable-Threshold Threshold Elements, *IEEE Trans. on Computers*, Vol. C-17, No.7, pp. 656-667.

Meisel, W.S. (1972). Piecewise Linear Discriminant Functions, Chapter 7 of *Computer-Oriented Approaches to Pattern Recognition*, Academic Press: New York.

Meisel, W.S., Fortunato, M.P., and Michalek, W.D. (1989). A Phonetically- based Speech Recognition System, *Speech Technology*, Apr/May 1989. pp. 44-48.

Quinlan, J. R. (1986). Induction of decision trees, *Machine Intelligence*, Vol.1, pp.81-106.

Rummelhart, D.E., and McClelland, J.L. (1986). *Parallel Distributed Processing, Vol. 1 & 2*. MIT Press: Cambridge, Mass.



Start
1
Is w₁•x' ≥ 0 ?
no
yes
2
Is w₂•x' ≥ 0 ?
3
Is w₃•x' ≥ 0 ?
Terminal node 1
Terminal node 2
Terminal node 3
Terminal node 4
x
y₂ = 1 if w₂•x' ≥ 0, -1 otherwise
y₃ = 1 if w₃•x' ≥ 0, -1 otherwise
y₁ = 1 if w₁•x' ≥ 0, -1 otherwise
z₁ = 1 -y₁ - y₂ ≥ 2, 0 otherwise
z₄ = 1 y₁ + y₃ ≥ 2, 0 otherwise
z₂ = 1 -y₁ + y₂ ≥ 2, 0 otherwise
z₃ = 1 y₁ - y₃ ≥ 2, 0 otherwise
z₁
z₂
z₃
z₄

(a) A decision tree with four terminal nodes.

(b) An equivalent network with four outputs. (zi = 1 is equivalent to terminal node i being chosen; only one output will be 1 simultaneously.)

Figure 1: A decision tree and an equivalent network.

AN OPTIMAL SELF-ORGANIZING PATTERN CLASSIFIER

A. MEKKAOUI, P. JESPERS
Microelectronics Laboratory
Place du Levent 3, B-1348 L.L.N. BELGIUM

1. INTRODUCTION:

Most applications of neural networks are classification problems in which an analog or binary input pattern should be assigned to one of m possible classes. Unfortunately for most of these classifiers it is very difficult to tailor their behavior when performing in a noisy non-controlled environment. It is also very difficult (if not impossible) to provide a rejection mechanism, which poses serious problems in real-world pattern recognition applications where it is preferable to reject totally unknown patterns rather than accepting poor matches.

This paper reports on the design of an optimal parallel pattern classifier whose ability to reject noise is controlled externally and in which a rejection mechanism is provided. The term optimal is used in the sense that classification is based on the maximum normalized correlation criterion, which is the same as the minimum Euclidean distance one for normalized inputs. A method to improve the system sensibility to noise is also proposed.

The work described herein is a part of a more global project dealing with the silicon implementation of a neural net based pattern recognition system. This explains our approach to solve certain problems at the implementation level rather than at the "equation level".

Throughout the paper $N_2(x)$ denotes the euclidean norm of vector **x**, bold characters indicate vectors and matrices. Note that the indices 1,2,...j,...,m used to denote the output cells do not necessarily reflect their physical positions, i.e., output cell j codes class j patterns but is not necessarily located at the jth physical position.

Because the proposed network is highly inspired from the ART architectures developed by Carpenter and Grossberg [1],[2] a brief introduction to the principle lying behind, is first presented in section 2.

2. ART SYSTEMS:

Figure 1. shows a block diagram of a typical ART system. The net accepts an n-input pattern and classifies it in one of the m possible classes. It should have developed during the learning process, in a form of weights, codes for the m classes. Each code can be considered as a representative prototype (a kind of mean) of a set of patterns forming a cluster.

Suppose that after a certain time the system has learned k patterns and sees a new stimulus at its inputs. Ideally we want the system to behave as follows: if the new input belongs to some already learned cluster the net must recognize it and adapt its weights accordingly, to take into consideration this new observation. If the present pattern is completely different from all the stored exemplars, a new code must be created for it. The weights are, then adapted to consider this input as the center of a new cluster. In the ART nets this is done by the following mechanism:
The input pattern is presented to a first layer where some computation takes place. The result of this computation is send via a bottom-up pathway to the second (output) layer where cell j computes its activation by:

$$a_j=\sum_{i=1}^{n} p_i b_{ij} \tag{1}$$

Where $p_1,\dots,p_i,\dots,p_n$ are the outputs of the input block, and $b_{1j},\dots,b_{kj},\dots,b_{nj}$ are the bottom-up weights to the output cell j. Its output is evaluated by:

$$\begin{aligned} o_j&=d \ \text{ if } a_j>a_k\,,\ \forall k\neq j \\ &=0 \text{ else.}\end{aligned} \tag{2}$$

Where d is some maximum value. This a winner-take-all system. Only the node with the greatest activation becomes maximally active and all the other are shut-off. The fact the jth output cell wins the competition can be interpreted as a hypothesis making that the actual input belongs to the jth class. Once the competition is complete top-down signals reaches the first layer where they interact with the input signals to provide the reset system with a meaningful information to enable it to confirm or disconfirm the choice made. If the mismatch between the bottom-up pattern and the top-down one exceeds a certain threshold value, depending on the vigilance parameter, the reset sub-system sends an inhibiting signal to the second layer to get the winning cell out of competition, so that a new cell would become active. The search of a good choice continues in the same manner. Whenever the reset sub-system confirms a choice, the search procedure stops and the weights are updated accordingly. The system is said to be in a resonant state (hence the name ART). A cell whose weights hold a learned pattern is said to be a coded cell. When the system uses up all previously coded cells without achieving resonance a new previously uncoded cell is chosen and no search is engaged. For a thorough description of ART systems the reader is referred to [1] and [2].

. In the next section a description of the proposed system is presented.

3. A SELF-ORGANIZING PATTERN CLASSIFIER:

Though inspired from the ART system, it is, in fact, totally different and much more simpler. The model consists of three sub-systems (layers): the input layer is composed of n cells each of which receives a component of the n dimensional real valued input vector (pattern). The output layer has m output lines, each one represents a given category (class) of patterns. In normal operation only one output line is active(ON). The j^{th} cell is active only when the pattern present in the input layer belongs to the j^{th} class. The reset sub-system plays the role of a watching agent. It sends an alarming signal whenever the wrong output cell becomes active. In figure 2 a complete 3-input 2-output network is depicted for illustration purposes.

The input layer has the sole role of normalizing the inputs and clipping noise spikes by passing the normalized components through a nonlinear function. Let i_i be the input to cell i and p_i its output. p_i is related to i_i by:

$$P_i = F(\frac{i_i}{N_2(i)}) \tag{3}$$

Where $i=(i_1,..,i_i,..i_n)$ and F is a nonlinearity expressed as:

$$\begin{aligned} F(x) &= x \quad \text{if } x \leq \theta \\ &= \theta \quad \text{if } x > \theta \end{aligned} \tag{4}$$

At the output layer each unit evaluates its activity by equation (1). The unit with maximum activation represents the class to which the actual input is likely to belong. The output of the cells in this layer is determined by a competitive process. See equation (2). The reset layer receives this information and compare (in some suitable metric) the input pattern with the pattern held by the winning output cell. A reset signal is issued whenever this comparison is negative, i.e, The difference between the input pattern and the stored code exceeds a certain threshold, depending on the value of the vigilance parameter, ρ. ρ is comprised between 0 and 1. The closer it is to 1 the more vigilant is the system; conversely when it closer to 0 poor matches can be accepted without issuing a reset signal.

For the model we propose here, the reset signal is issued only in the case where the present stimulus is seen for the first time and no good match is found. This mean that any previously learned pattern will access its code directly without any search. This is due to input normalization and appropriate learning equations, as explained below.

Bottom-up weights (b_{ij}'s) are updated, in case where the p input belongs to the jth class, according to:

$$b_{ij}=(1-\alpha)b_{ij} + \alpha p_i \tag{5}$$

With $0<<\alpha<1$. Equation (5) indicates that the weights to the j^{th} output cell tend to replicate the patterns of class j. The parameter α indicates how much past experience is to be taken into account in updating weights. If it is closer to 0 more importance is given to experience, on the other hand when it is close to 1 more emphasis is put on the present observation.

When a pattern of class j is present at the input, the inequality:

$$a_j=\sum_{i=1}^{n} p_i b_{ij} > a_k=\sum_{i=1}^{n} p_i b_{ik} \tag{6}$$

is always verified for any k, k^{th} cell being already coded, because the set bij holds class j patterns and is maximally correlated only with those patterns by virtue of the normalized correlation. But we are not guaranteed that :

$$a_j=\sum_{i=1}^{n} p_i b_{ij} > a_u=\sum_{i=1}^{n} p_i b_{iu} \tag{7}$$

where the b_{iu}'s represent the weights to any uncoded output cell which still maintain their initial values and hold no pattern. The inequality can be assured by imposing proper initial values for the bottom-up weights. This can be achieved by setting them as indicated by (8):

$$b_{ik} < \frac{1}{\sqrt{n}} \qquad i=1,\ldots,n \text{ and } k=1,\ldots,m \tag{8}$$

n being the dimensionality of the input pattern as usual. Equation (8) guarantees that $N_2(\mathbf{b_u})<1$ which in turn makes (6) valid for coded and uncoded cells. $\mathbf{b_u}=(b_{1u},\ldots b_{iu},\ldots b_{nu})$.

The top-down weights t_{ij} serve for a comparison purpose and are local to the reset sub-system. All tij's are initialized to 0 and are updated, whenever a choice is confirmed, according to equation (5).

In order to assure that no reset occur when a cell is selected for a first time to code a new class, a test is provided at the implementation level to see if the corresponding tij's were ever updated. This is accomplished by having a flag register, f_j, associated with each set of t_{ij}'s: an "ON" f_j indicates that the t_{ij}'s were updated at least once. A rejection case occurs when all the f_j's are ON along with the reset signal. (This scheme is not shown in fig. 2.)

4. A VARIABLE VIGILANCE CLASSIFIER:

A serious limitation of most neural classifiers is their sensibility to noise, specially in cases where the difference between the closest prototypes is less than the noise it is wished to reject [3]. If this level is low the system confuses between the closest, but different, patterns. On the other hand if it is high the net tends to consider as different two noisy versions of the same pattern.

We propose here a method to cope with this difficulty. We assume that the system would operate in two phases: a learning phase and a normal phase. It is also assumed that we dispose of a set of sample prototypes.
In the learning phase the vigilance parameter is set so high that different codes are generated for the closest prototypes. At the limit one can set this parameter to the maximum. Then the prototypes are presented to the net which would generate a code for each prototype. No more than one pass is required for the learning phase. During the normal mode the net is used as a classifier where the vigilance parameter is set to achieve any desired level of noise tolerance. Entering the normal mode, plasticity (adaptativity) is not shut-off. The weights b_{ij} and t_{ij} are updated after each recognition of a j^{th} class pattern. This enables the system to keep pace with its environment.

In the jargon of "neural" associative memories the system just described looks like a memory with m fixed points. Each one having a basin of attraction whose size can be controlled at will.

5. CONCLUSION:

We have proposed an unsupervised (self-organizing) artificial neural network whose function is to optimally classify binary and analog valued patterns.

When successfully implemented as an analog integrated circuit(s) this system would be well suited for real time pattern classification and vector quantization applications.

REFERENCES:

[1]G. Carpenter, S. Grossberg, "A Massively Parallel Architecture for Self-Organizing Neural Pattern recognition Machine", Computer Vision Graphics and Image Processing, vol. 37, 1987, pp. 54-115.
[2]G. Carpenter, S.Grossberg, "ART2: Self-Organization of Stable Category Recognition Codes for Analog Input Patterns", Applied Optics, Dec. 1, 1987, pp. 4919-4930.
[3]R. Lipmann, "An introduction to computing with neural nets", IEEE ASSP Magazine, April 1987, pp 4-22.

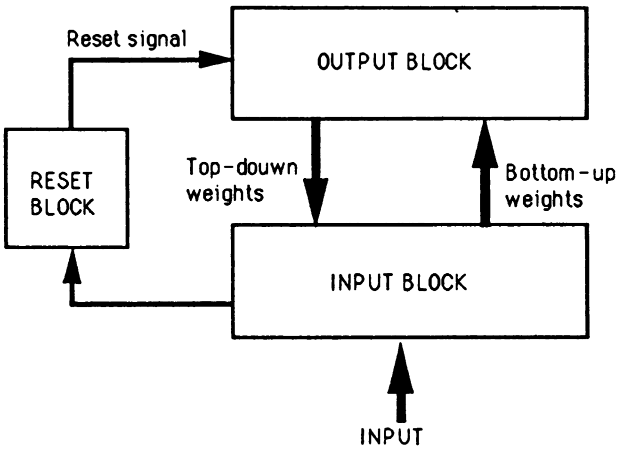


Fig. 1. A block diagram of a typical ART system

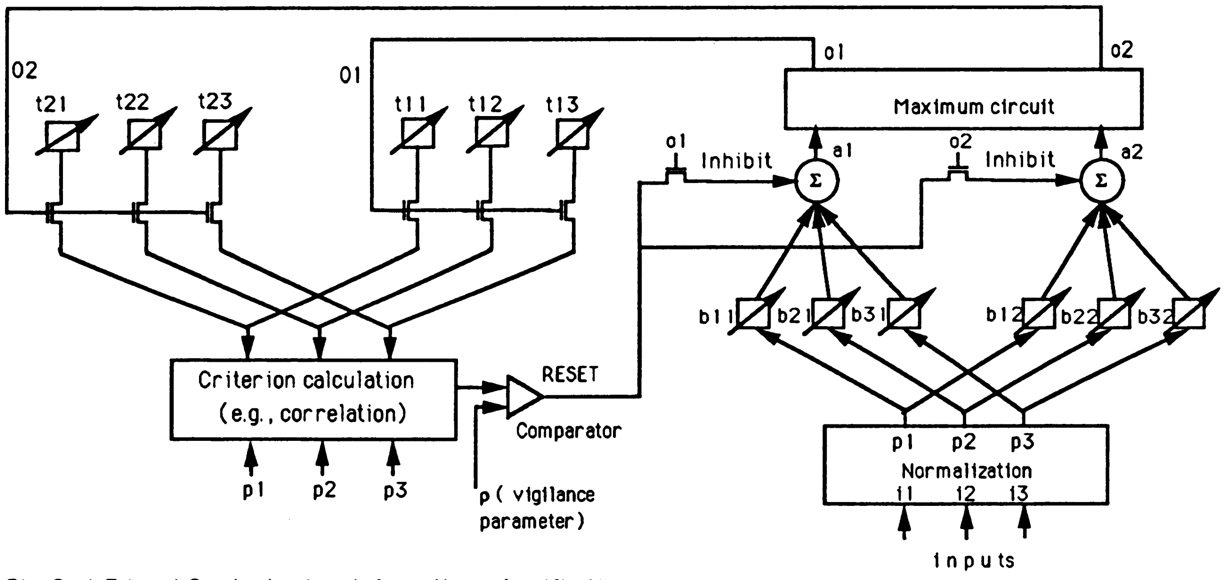


Fig. 2. A 3 input 2 output network for pattern classification . Generalization to n inputs m outputs should be obvious.

Y W X An adjustable analog multiplying coefficient. y=wx. In case where x is not shown it is assumed to be unity.

B C A A switch. A is connected to B only if C is active (ON)

Probability-based Neural Networks

John H. Murphy

Westinghouse Science & Technology Center
1310 Beulah Road
Pittsburgh, PA 15235

Abstract

This paper introduces a new class of neural networks based on the integration of three technologies: artificial intelligence, neural networks and probability theory. This approach has realized the Adaptive Probabilistic Reasoning System (APRS) — a probability-based neural network for adaptive knowledge processing. This approach combines probability theory and optimization methods and forms the theoretical foundation for unifying expert system and neural network technologies. APRS is based on the Probabilistic Reasoning System (PRS). PRS is a mathematical formulation of an inference engine for expert systems. Because of its structure, the knowledge can be located and interpreted in the resultant probability-based neural networks.

1. Introduction

To meet the military's need for a high-speed, mathematically-sound reasoning engine that could be applied to both expert systems and neural networks, we developed a new class of processing elements which explicitly process integers instead of floating point numbers. These processing elements were specifically developed for use in the PRS to produce expert system inference engines which can be verified and validated. Accordingly, these processing elements work with probabilities. To exploit this technology, the probabilities are converted into integers by scaling and truncating. The resulting processing elements can operate at very high speeds.

The probability-based neural network: is rigorously based on classical probability theory; propagates probabilities rather than unstructured real numbers; possesses an inference engine which can be both validated and verified; has columns of binary decision trees instead of layers of processing elements; maintains its knowledge in a localized and interpretable (nonholistic) form; uses high speed algorithms.

Our interest in parallel computer architectures is focused on solving high-speed decision analysis problems. To accomplish this task, the architecture must be able to handle a mixture of parallel and serial processing. Conventional von Neumann-type serial processing computers are based on an instruction-flow architecture. PRS utilizes a data-flow architecture which is ideal for a parallel processing system. This multiprocessor architecture accommodates serial processing by reusing available computational resources.

The PRS decision processing element performs arithmetic on probabilities. When used in this manner, the inputs and outputs are constrained to lie in the closed interval from zero to one. The simplest form of this device multiplies a pair of linearly transformed input signals.

The current focus is on developing and applying computer systems which adapt to their environment. Westinghouse's program has concentrated on adaptive knowledge processing systems. An adaptive knowledge processing system interfaces with its environment — receives information from it, and transmits information to it. The system contains within itself a body of knowledge — "the knowledge base." Knowledge is represented as a set of "IF...THEN" production rules. These rules govern how information from the environment is to be processed — what conclusions are to be drawn from it; what actions are to be taken. An expert system performing deductive reasoning is a simple knowledge processing system. An adaptive knowledge processing system must also do inductive reasoning. Specifically, it must use the information from the environment to refine and modify its knowledge base. It must continuously modify its set of rules to improve its performance and better conform to its changing environment. An adaptive knowledge processing system learns.

There are two distinct kinds of structures for processing knowledge — expert systems and neural networks. An expert system uses a knowledge based approach to solving problems. Its knowledge is basically static. It works either to determine the consequences of given input conditions, or it works to determine what input conditions are required to produce a specified output. Thus, an expert system is strictly deterministic — its function is strictly deductive. The knowledge base of an expert system is constructed by the very slow and expensive process of knowledge engineering.[1,2] In contrast, a neural network solves decision analysis problems in an entirely different manner. It begins with a large network of massively interconnected processing elements. The processing elements are very simple. Knowledge is imbedded into the network by adjusting the weights applied to the signals between pairs of processing elements. This knowledge is basically dynamic.

There is one characteristic of conventional neural networks that is undesirable — their solutions are inherently holistic. In conventional neural networks the knowledge is not localized, as it is in expert systems, but generally dispersed throughout the network. Holistic or gestalt knowledge is not reducible, in any obvious way, to a simple set of production rules. Dispersion of the knowledge makes functional interpretation, verification, and validation very difficult. This means that conventional neural networks are not readily translated into expert system structures. This makes them unsatisfactory for most practical applications.

A dilemma exists. Expert systems and neural networks are two apparently incompatible structures for knowledge processing. They each offer a partial solution to the problem of constructing a truly adaptive knowledge processing system, but only a partial solution. On the one hand, expert systems are comprehensible but hard to train; and neural networks can be trained but are hard to comprehend. Can this dilemma be resolved? Can the two structures be combined into an integrated system which can be both trained and interpreted?

An adaptable system is needed which reveals its internal operations. If the following scenario were feasible, we would have such a system: first, train a neural network to derive its holistic solution; second, convert the holistic solution into a set of production rules for a knowledge base; and third, use this knowledge base to implement an expert system. Unfortunately, the above scenario is unfeasible because the conversion step — going from a holistic solution to a set of interpretable production rules — is particularly onerous and difficult to execute.

2. The Probability-based Neural Network Paradigm

The goal is to develop an adaptive knowledge processing system which is logically consistent and does not require close supervision by a perceptive individual. This implies that the central feature of such a system must be an inference engine which handles probabilities. PRS is the ideal candidate; it is an entirely consistent realization of an expert system inference engine which can be verified and validated. It is mathematically consistent, it exhibits rigorous logical performance, and it will not exhibit spurious, unanticipated or untrustworthy behavior.

The development of PRS was motivated by the need to realize real-time control systems — a task for which state-of-the-art expert systems were unsuited. Those systems lacked a calculus for carrying out "uncertainty" propagation that was based on sound mathematical principles. They use *ad hoc* propositions that are arbitrary and inconsistent. PRS is based on probabilities — a major departure from the principles of artificial intelligence.

In creating the PRS, we had to overcome the problem generated by interdependence among the variables. This problem is handled in conventional expert systems by resorting to strong and unrealistic assumptions. In the PRS, these assumptions are avoided by using a new, explicit and precise measure of independence — the *I-factor*

$$I(A, B \mid C) = \frac{P\{A \mid B\,C\}}{P\{A \mid C\}} \tag{1}$$

where P is a conditional probability measure and A, B, and C are events; — and its counterpart the *K-factor*

$$K(A, B, C) = \frac{I(A, B \mid C)}{I(A, B)} \tag{2}$$

which is a symmetric function of its arguments.

In PRS, there are two crucial formulas for manipulating probabilities — one for sequential updating

$$P\{H \mid E'\} = \Big(I(H, E' \mid E)\cdot P\{H \mid E\} \;-\; I(H, E' \mid \bar{E})\cdot P\{H \mid \bar{E}\} \Big) \cdot P\{E \mid E'\} + I(H, E' \mid \bar{E})\cdot P\{H \mid \bar{E}\} \tag{3}$$

and one for parallel updating

$$P\{H \mid E'F'\} = K(H, E', F')\,\frac{P\{H \mid E'\}\,P\{H \mid F'\}}{P\{H\}} \tag{4}$$

where H, E, E', F, and F' are events. These formulas are general forms of the inference formulas used in PROSPECTOR, a classic expert system.[3]

The processing elements assume constant conditional probabilities and are woven into a network as follows. Start by considering a rule-based expert system to be a collection of decision trees. Each of the decision trees is composed of a collection of processing elements which carry out the sequential and parallel updating on the measures of belief.

Recall that in implementing any expert system, one must face the problem of the knowledge acquisition bottleneck. Currently, the programming of knowledge into expert systems is performed manually at a great cost. To accomplish our goal, automatic knowledge acquisition was identified as an potentially low cost alternative.

A wide variety of possibilities were considered for automating knowledge acquisition within expert system technology. Among the numerous techniques for inductive reasoning considered were: various statistical techniques, such as CART (classification and regression tree analysis);[3] artificial intelligence techniques like the ID3 and C4 algorithms of Quinlan;[5] and neural network techniques. We concluded that the neural network technology might be useful because such networks could be made compatible with expert system technology.

Our development brings together two technologies — expert systems and neural networks. The expert system design is used as the basis for the neural network. We then train the network. When trained, all the interconnections are probabilistic IMPLIES and hence expressible as production rules. The result is a system which completely bypasses the knowledge acquisition bottleneck and, at the same time, avoids a holistic solution. The new network is easily trained — like most neural networks; and is based on production rules — like most expert systems. Since the product performed by the probabilistic AND is inherently nonlinear, any of a number of optimization techniques might suffice as learning algorithms.

Once the neural network is trained, we reap the benefits of the foresight in starting with an expert system skeleton — the trained network is at the same time a working expert system. We can extract a set of production rules from this probability-based neural network, and we can interpret these production rules. This capacity for interpretation is an inherent property of expert systems and hence of any probability-based neural network.

For a typical neural network, there are three classes of algorithms to discuss:

- linkage algorithms — these algorithms determine the strength of the connection between processing elements.
- processing element algorithms — these algorithms tell how to combine information from two or more linkages.
- learning algorithms — these algorithms provide the network with the ability to adapt to changes in the environment.

In this paper, we examine the linkage and processing element algorithms of a new class of neural networks — the probability-based neural network.

The linkage algorithm determines the strength of the connection between the processing elements. Conventional neural networks perform a simple weighting of the signals on the linkages. Probability-based neural networks perform a more complex function — the implication transformation found in expert systems.

In conventional neural networks, the linkage algorithm determines the linkage output signal y by multiplying the linkage input signal x by a weight A.

$$y = A \cdot x \qquad (5)$$

The linkage weight A can be either positive or negative depending on whether the connection between the processing elements is excitatory or inhibitory, respectively. This algorithm allows one to reason with weighted information — a classical method of decision making.

Probability-based neural networks utilize a simple implication function or linear transformation. The algorithm for simple implications determines the linkage output signal y by multiplying the linkage input signal x by a weight A and adding an offset B.

$$y = A \cdot x + B \qquad (6)$$

The linkage offset is necessary to work with probabilities as implied by equation (3). Without it, $B = 0$, this algorithm reduces to the conventional neural network algorithm, equation (5).

The processing element algorithm describes the method of combining signals from two or more linkages. Conventional neural networks perform a threshold or sigmoid function on the sum of the incoming signals. In the probability-based neural network, a product is performed.

Early neural nets, such as Perceptron[6] and MADALINE,[7] determined the output signal O by summing the incoming signals y_i and performing a unit step function U if this sum is above a threshold value Θ.

$$O = U(\Sigma y_i - \Theta) \qquad (7)$$

Later neural nets, such as backpropagation, and the Boltzmann and Cauchy machines,[8] use the differentiable sigmoid function because it allows them to use classical optimization techniques for learning. The output signal O is obtained by applying a sigmoid function to the incoming signals y_i,

$$O = \{ 1 - e^{-(\Sigma y_i - \Theta)/T} \}^{-1} \qquad (8)$$

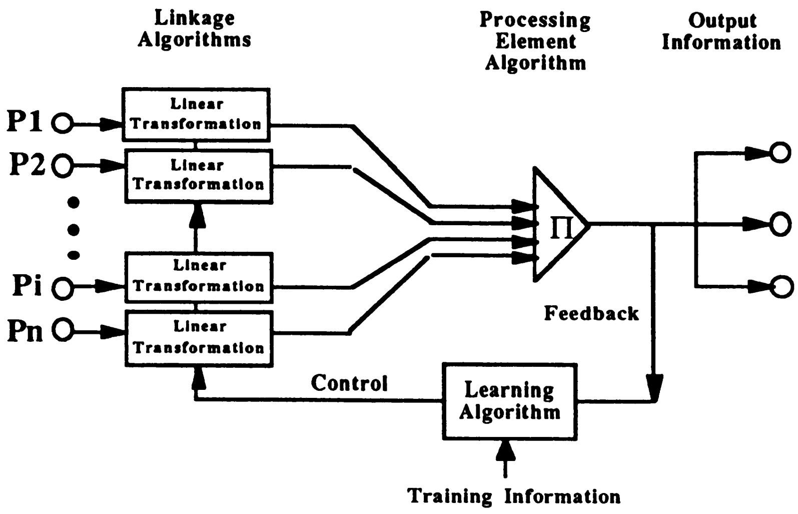
where the threshold value is Θ and the normalization factor is T.

The probability-based neural networks use a processing element algorithm which determines the output signal O by multiplying together the incoming signals y_i as implied by equation (4).

$$O = \Pi y_i \qquad (9)$$

This is ideal — it is nonlinear; it is simple to compute; it is differentiable. The introduction of the linear offset in determining the interconnection strength allows one to validate and verify its performance. This approach to building a neural network is remotely related to the classic sigma-pi units in that multiplication is utilized.[8]

Figure 1 — Adaptive Probabilistic Reasoning System Paradigm



3. Conclusion

In summary, the probability-based neural network, APRS, integrates neural network and expert system technologies. It is a new approach to neural networks which can be validated and verified and used for decision making in autonomous operations. It is therefore a ideal candidate for potential military applications. Its paradigm is simple as illustrated in Figure 1. Its attributes include:

- The linkage algorithm is a linear transformation.
- The processing element algorithm is a product function.
- The probability-based neural network is nonholistic, since this network has an expert system architecture — i.e. made up of collections of binary decision trees.
- The probability-based neural network has a knowledge base, since it is fundamentally based on the Probabilistic Reasoning System — an expert system.
- The probability-based neural network stores knowledge as symbolic rules which allows a trained network to be interpreted and understood, and thereby verified and validated.

4. References

1. E. A. Feigenbaum, "The art of artificial intelligence: themes and case studies in knowledge engineering," *Proceedings IJCAI–83*, 1158–1169 (1977).

2. E. A. Feigenbaum, "Expert systems in the 1980s," in *State of the art report on machine intelligence*, A. Bond, ed. Pergamon-Infotech, Maidenhead (1981).

3. Richard O. Duda, Peter E. Hart, Nils J. Nilsson, "Subjective Bayesian methods for rule-based inference systems," *Proc. of 1976 National Computer Conf. AFIPS*, 1075-1082 (1976).

4. Leo Breiman, Jerome H. Friedman, Richard A. Olshen and Charles J. Stone,*Classification and Regression Trees*, Wadsworth International Group, Belmont CA (1984).

5. J. R. Quinlan, "Inductive knowledge acquisition: a case study," in *Applications of Expert Systems* J.R. Quinlan, ed. pp157-73. Addison-Wesley, NY (1987).

6. Frank Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Spartan Books, Washington D.C. (1962).

7. B. Widrow and F. W. Smith, "Pattern Recognizing Control Systems," *Computer Information Sciences (COINS) Symposium* (1963).

8. D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition Volume 1: Foundations*, MIT Press, Cambridge, MA (1986).

Fault-Tolerance of Optimization Networks: Treating Faults as Additional Constraints*

Peter W. Protzel and Michael K. Arras

Institute for Computer Applications in Science and Engineering
Mail Stop 132C, NASA Langley Research Center
Hampton, VA 23665

Abstract Although it is widely accepted that artificial as well as biological neural networks exhibit a certain degree of fault-tolerance, the underlying mechanism and the relation between the fault-tolerance and the functional characteristics are still poorly understood. This paper investigates this relationship by looking at two examples of networks that solve optimization problems, the Assignment Problem and the Traveling Salesman Problem. While the principal structure of both networks is very similar, the difference of the performed tasks and the redundancy of the problem representation in one case leads to distinctive fault-tolerance characteristics. The reasons for the performance degradation in the presence of faults are discussed, and numerical results for large-scale fault-injection experiments are presented. It can be shown that the simulated faults act like additional constraints for the problem, but do not impair the network dynamics in converging to the best solution possible under these constraints

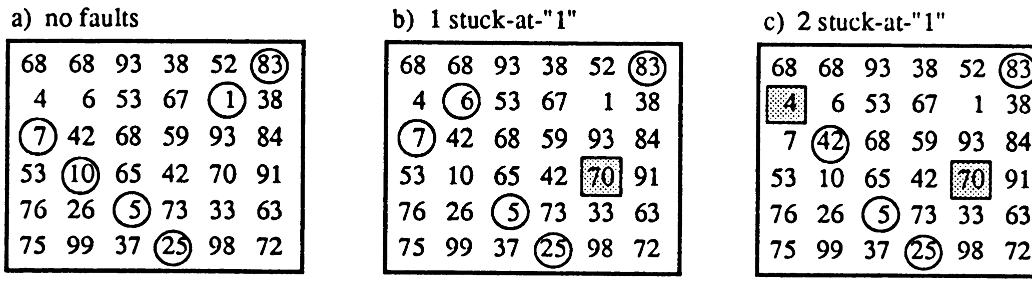
1. Fault-Tolerance and Problem Representation

Fault-Tolerance is a qualitative, general term defined as the ability of a system to perform its function according to the specification in spite of the presence of faults in its subsystems. This definition is very unspecific and a system that is said to be fault-tolerant does not necessarily tolerate any number of faults of any kind in any of its subsystems. A specific way to quantify fault-tolerance is to determine the *performance degradation* in the presence of certain faults in certain subsystems, given that some measure of performance exists. In this paper, we compare the performance degradation for two examples of Artificial Neural Networks (ANNs) that 'solve' optimization problems, the Assignment Problem (AP) and the Traveling Salesman Problem (TSP). The main characteristics of both model problems are briefly summarized below. Our goal is to understand qualitatively how a certain fault affects the dynamics and the 'result' of the network, and to assess the performance degradation quantitatively in a statistically relevant manner.

The TSP is to find a closed tour of minimal tour length for a certain number of cities n. The distance between any pair of cities is given as input data. We use a modification of Hopfield's and Tank's (H&T's) [2] original solution method (Brandt et al., 1988 [1]) that produces consistently valid tours, which is seen as a prerequisite for a fault-tolerance evaluation. For a problem with n cities to be represented by the ANN, n^2 'neurons' or units are arranged in form of a quadratic matrix with the output of a unit V_{Xi} representing the 'hypothesis' that city X is in position i of the tour [2]. A solution represents a valid tour if the ANN converges to a permutation matrix with only one unit $V_{Xi}=1$ in each row and column. This problem representation is redundant in the sense that the same tour can be mapped onto the network in 2n different ways because the starting point and the direction are free variables.

The Assignment Problem (AP) or list matching problem is to find the one-to-one assignment or match between the elements of two lists that has minimal cost given the cost for each individual pairing [1]. The problem is again represented by an ANN with n^2 units in matrix-form with the only difference to the TSP that the output of a unit V_{Xi} now represents the hypothesis that element X of the first list should be assigned to element i of the second list. To produce a valid solution with an one-to-one match the ANN again has to converge to a permutation matrix. Thus, with identical constraints, the only difference between the two ANNs is a quadratic cost function for the TSP mapping onto additional interconnections versus a linear cost function for the AP mapping onto the external

* This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-18605 while the authors were in residence at ICASE.



a) no faults

68	68	93	38	52	83
4	6	53	67	1	38
7	42	68	59	93	84
53	10	65	42	70	91
76	26	5	73	33	63
75	99	37	25	98	72

b) 1 stuck-at-"1"

68	68	93	38	52	83
4	6	53	67	1	38
7	42	68	59	93	84
53	10	65	42	70	91
76	26	5	73	33	63
75	99	37	25	98	72

c) 2 stuck-at-"1"

68	68	93	38	52	83
4	6	53	67	1	38
7	42	68	59	93	84
53	10	65	42	70	91
76	26	5	73	33	63
75	99	37	25	98	72

Figure 1. Cost matrix for an exemplary Assignment Problem with 6 elements and different solutions of an ANN (circled elements) solving that problem: global optimum (a) and performance degradation (b, c) after injection of stuck-at-"1" faults (shaded squares). Note that (b, c) are still optimal solutions under the constraints imposed by the faults.

current. Additionally, the problem representation for the AP is not redundant, that is, a particular solution has a unique network representation.

Two different types of faults are used in our simulations that correspond to the highest failure rates of a hardware implementation: a *stuck-at-"0"* or *stuck-at-"1"* fault occurs when the output of a unit (operational amplifier) is permanently pulled to the ground potential or the highest potential, respectively. The fault-locations were randomly selected with one important exception: *Two simultaneous faults in the same row or column are prohibited.* This is because two stuck-at-"1" faults in the same row or column would preclude a valid solution. Figure 1 illustrates the behavior of an ANN solving a simple AP with 6 elements. The values shown in the quadratic matrix represent the cost-matrix and the circled numbers correspond to the solution of the ANN, which is the global optimum in Figure 1a. In Figure 1b one stuck-at-"1" fault is 'injected' at the position marked with the shaded square, and this fault determines that particular assignment of cost 70 to be part of the overall solution. It can be seen that the network still converges to a valid solution with a performance degradation corresponding to the higher, overall cost of the solution. Figure 1c shows a similar behavior of the ANN after injecting two stuck-at-"1" faults.

Interestingly, by analyzing the solutions in Figure 1b and c we found that *the network still converged to the best solution possible under the constraints imposed by the faults.* Thus, the injected faults act as additional constraints and do not affect the 'ability' of the network to find the (redefined) global optimum. Stuck-at-"0" faults preclude a particular solution and have no effect at all on the AP unless the fault location coincides with an active unit that is part of the solution. In that case we could observe the same phenomenon, that is, the network treats the fault as an additional constraint and converges to the 'next-best' solution.

An ANN solving the TSP shows a similar performance degradation under the presence of faults, but there are two important differences. First, the redundant problem representation reduces the effect of a fault, because a single stuck-at-"1" in position (X,i), for example, determines only that city X has to end up in position i of the tour, which does not yet limit the solution space. Only when two stuck-at-"1" faults appear in adjacent columns (e.g. in positions (X,i) and (Y,i+1)), then this determines a link between city X and Y and acts as a new constraint for the remaining tour. The other difference is that the fault-free performance of an ANN solving the TSP is lower and usually suboptimal. Thus, we have seen cases where two stuck-at-"1" faults in adjacent columns determine to include a link between two cities into the tour that was part of the optimal solution not found by the fault-free network. This actually means that a *performance increase after fault-injection* is possible under certain conditions.

2. Large-Scale Fault-Injection Experiments

One problem in assessing the performance of optimization networks is the considerable performance variation for different problem instances and sizes, and, for the TSP, also the dependency on the random initialization. In order to obtain statistically relevant results it is necessary to simulate a sufficient number of different problem instances for different sizes and different random initializations. Another problem is the lack of an appropriate performance measure that is independent of a particular problem and of the problem size, and relates the solution of the ANN to reference values of interest. Therefore, we defined a solution quality q as a performance measure that relates an ANN-solution of cost c for a particular problem instance to the average cost value c_{ave} of a sufficient number of random trials and to the optimal value c_{opt} such that $q=(c_{ave}-c)/(c_{ave}-c_{opt})$ [4]. Thus, q is a normalized

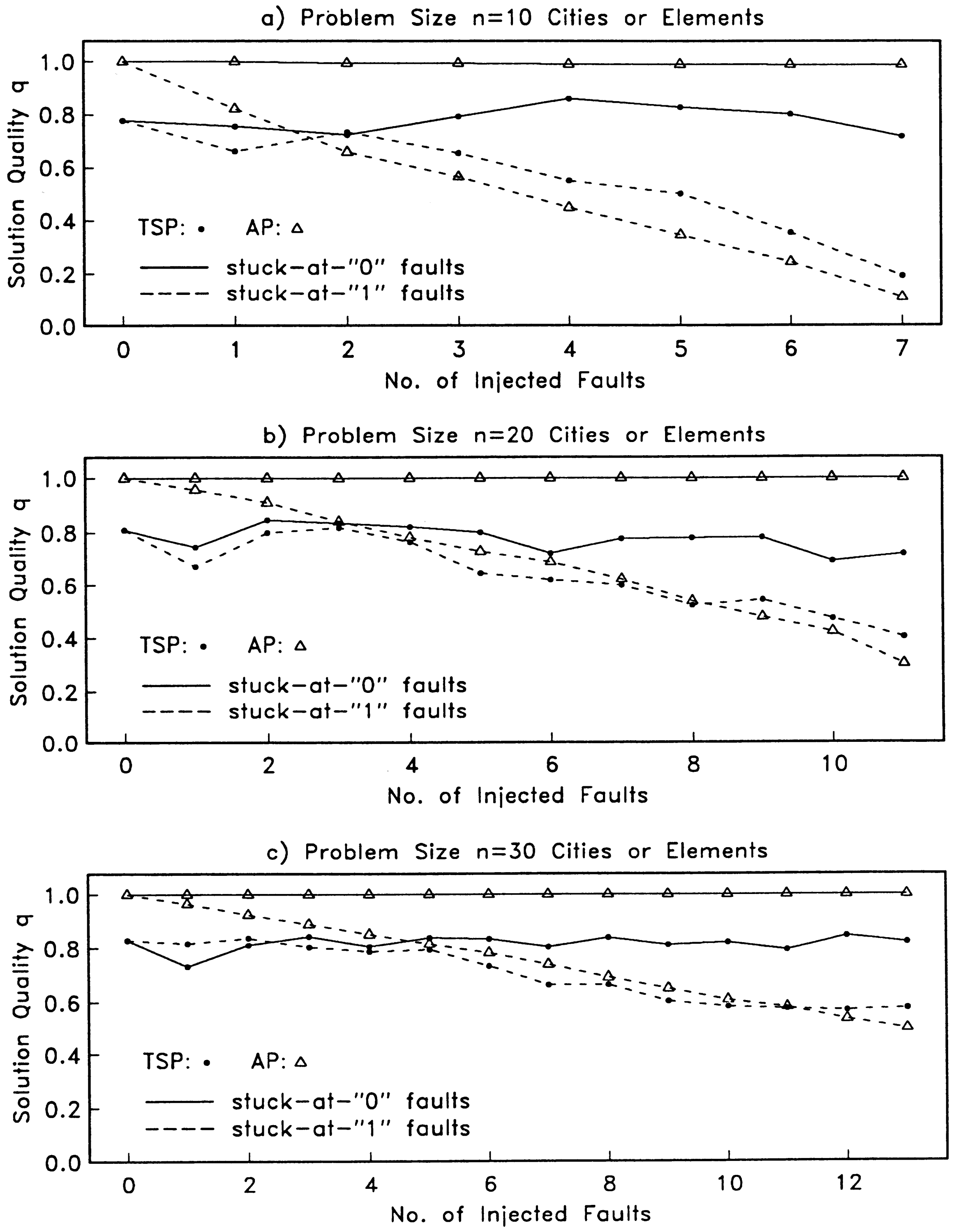


Figure 2. Performance degradation of an ANN solving the Traveling Salesman Problem (TSP) and the Assignment Problem (AP) after fault-injection for different problem sizes. The values are averages over 10 different problem instances for each size with additionally 10 different random initializations each for the TSP.

factor with a value q=1 if a given solution c is optimal ($c=c_{opt}$) and a value q=0 if the answer has the quality of an average random tour ($c=c_{ave}$).

The definition of q requires the knowledge of the optimal solution values c_{opt}, which can be obtained for the AP by using a conventional textbook algorithm [5]. Since the TSP is an NP-complete problem, values for the global optimum are generally unknown, but the Lin-Kernigham algorithm [3] provides excellent reference values. A possible event $c<c_{opt}$ can be recognized by a value q>1. A test-set for the TSP included 10 different, randomly generated city-distribution for each problem size n=10, 20, and 30. Values for c_{ave} were obtained by generating 10^5 random tours for each city-distribution. A similar test-set with 10 different, random cost matrices for n=10, 20, and 30 were generated for the AP. The fault-locations were also randomly selected with the same locations for stuck-at-"1" and stuck-at-"0" faults.

The numerical results in Figure 2 confirm our conjecture that stuck-at-"0" faults have practically no effect on the AP performance while stuck-at-"1" faults result in an almost linear performance degradation with respect to our performance measure. The redundancy of the problem representation for the TSP is reflected in a relatively slower performance decrease as the number of faults increase. With the number of stuck-at-"1" faults approaching the number of cities or elements, the performance for both the TSP and the AP approaches zero, which corresponds to the random average (Figure 2a). This is because the randomly selected fault-locations eventually predetermine a random tour. A more detailed interpretation of the results and a complete description of all the necessary information to recreate the results can be found in a forthcoming technical report.

3. Conclusion

None of our simulations failed to converge to a valid tour because of one or more injected faults. The only 'total failure mode' for both examples seems to be the occurrence of two simultaneous stuck-at-"1" faults in the same row or column preventing a valid tour, which was explicitly excluded in our experiment. The most interesting result is the observation that the faults act like additional constraints and do not seem to impair the dynamics of the network. Preliminary results indicate that the *conditional performance* of the network in the presence of faults might not change at all, that is, that the network still finds the optimum under the additional constraints imposed by the faults, or at least a good but suboptimal answer of the same average quality q as in the fault-free case. This also implies that it is possible to predict the fault-tolerance for particular problem instances. For example, the performance degradation for the AP in Figure 1 becomes predictable by analyzing the cost-matrix under the assumption that the network will always be able to find the best possible solution under the constraints of faults in certain locations. Since the fault-tolerance is often seen as one of the key benefits of an ANN-hardware implementation, the predictability will become a crucial criterion for real-world applications with provable reliability requirements.

References

[1] Brandt, R. D., Wang, Y., Laub, A. J., and Mitra, S. K. Alternative networks for solving the traveling salesman problem and the list-matching problem. In *Proceedings of the IEEE International Conference on Neural Networks, San Diego, CA* (July 1988), pp. II-333-340.

[2] Hopfield, J. J., and Tank, D. W. "Neural" computation of decisions in optimization problems. *Biological Cybernetics 52* (1985), 141-152.

[3] Lin, S., and Kernigham, B. W. An effective heuristic algorithm for the traveling salesman problem. *Operations Research 21* (1973), 498-516.

[4] Protzel, P., Palumbo, D., and Arras, M. Fault-tolerance of a neural network solving the traveling salesman problem. ICASE Report No. 89-12 / NASA Contractor Report 181798, ICASE / NASA Langley Research Center, Feb 1989.

[5] Syslo, M. M., Deo, N., and Kowalik, J. S. *Discrete Optimization Algorithms.* Prentice Hall, Inc., Englewood Cliffs, NJ, 1983.

Sampling Learning, Recall, and Filtering in Stable, Adaptive Neural Systems with Graded Response

Bernd Schürmann
Siemens AG, Corporate Research and Development
Otto-Hahn-Ring 6, D-8000 München 83

I. Introduction

Stable, adaptive artificial neural systems possess remarkable properties which are subject of this contribution. First, the underlying model is discussed. Next, macroscopic measures for the global dynamic behavior of neural nets are introduced, followed by a discussion of learning and recall. Subsequently, typical results of a computer experiment are presented, followed by their theoretical explanation.

II. Theoretical Foundations

The starting point is a fully connected one-layer network with q nodes. The activity z_k of node k, $k = 1,..,q$, is described by [1,2]:

$$dz_k/dt \equiv \dot{z}_k(t) = -C_k z_k + \Sigma_s w_{sk}(t)\, S[z_s(t)] + K_k \tag{2.1}$$

with the real valued function $z_k \varepsilon (-\infty,\infty)$. The change of z_k with time arises from (i) a decay term with constant C_k, (ii) an internal input term with weights w_{sk} of the units connected with node k and output signals $S(z_s)$ of these units, and (iii) an external input to node k consisting of the component K_k of a pattern vector **K** assumed to be constant with respect to the relevant time scales of the network. It is assumed that K scales with the number of nodes, i.e.

$$K_k = q \cdot \mathrm{sgn}(K_k)\ . \tag{2.2}$$

The pattern vector **K** may contain noise, i.e. **K** may consist of a systematic and a stochastic part:

$$\mathbf{K} = \mathbf{K}_{syst} + \mathbf{K}_{stoch}\ . \tag{2.3}$$

Therefore, eq.(2.1) in general is a stochastic differential equation. For the signal function we use

$$S(z) = \tanh(z/T)\ . \tag{2.4}$$

The system is asymptotically stable, if [3]

$$\mu_k w_{sk} = \mu_s w_{ks}\ , \tag{2.5}$$

$$\dot{w}_{sk} = -\lambda_{sk} w_{sk} + S(z_s)\, S(z_k), \quad s < k\ . \tag{2.6}$$

The quantities μ_k, μ_s, and λ_{sk} are real and positive. The change of w_{sk} with time arises from (i) a decay term with constant λ_{sk}, and (ii) a Hebbian learning term. The weights w_{sk} are calculated from (2.6) for $s < k$ and subsequently from (2.5) for $s > k$.

The expression

$$L(t) = \sum_{k=1}^{q} \mu_k [S(z_k)(C_k z_k - K_k) - C_k \textstyle\int^{z_k} S(\zeta_k)\, d\zeta_k] - \sum_{s,k} \mu_k w_{sk} S(z_s)\, S(z_k)$$

$$+ (1/2) \sum_{s,k} \mu_k \lambda_{sk} w^2_{sk}, \quad s<k \tag{2.7}$$

is a Lyapunov function for the system [3], i.e.

$$\dot{L} = (\text{grad}_z L) \cdot \dot{\mathbf{z}} + (\text{grad}_w L) \cdot \dot{\mathbf{w}} < 0 . \qquad (2.8)$$

Eq.(2.8) is satisfied, if

$$\text{grad}_z L = - \alpha \dot{\mathbf{z}} , \quad \text{grad}_w L = - \beta \dot{\mathbf{w}} , \qquad (2.9)$$

with proportionality constants α and $\beta > 0$. The network can be structured into several layers.

III. Macroscopic Observables

Large sized neural nets call for observables which quantify the current global state of the net. The Lyapunov function is such a measure, as well as the direction cosine, originally introduced for 2-state systems [4]. For models with graded response the direction cosine is modified to

$$a^{(v)}(t) = \Sigma_k \; S[K_k^{(v)}] \, S[z_k(t)] / \{[\Sigma \, S^2 (K_k^{(v)})]^{\frac{1}{2}} [\Sigma \, S^2 (z_k(t))]^{\frac{1}{2}}\} . \qquad (3.1)$$

It describes the overlap of the v-th pattern vector $\mathbf{K}^{(v)}$ with the current state vector $\mathbf{z}(t)$ of the system.

IV. Learning and Recall

The network should store an as large as possible number of patterns to be retrieved. Success or failure depends on the learning method. A suitable procedure is "sampling learning": Samples of pattern distributions with centers $\mathbf{K}_{syst}^{(v)}$, $v = 1,..,m$ are offered to the net, with samples from the v-th distribution for a time δt_v^{α} where $\alpha = 1,2,..,c$, and c = number of presentation cycles. The learning time is given by

$$t_{learn} = \Sigma_{\alpha} \Sigma_v \; \delta t_v^{\alpha} . \qquad (4.1)$$

The intervals δt_v^{α} are determined by use of (3.1). As soon as $a^{(v)}(t) = 1$, δt_v^{α} is fixed. By means of (2.9) one obtains

$$< \dot{\mathbf{w}} > = - (1/\beta) < \text{grad}_w L > . \qquad (4.2)$$

The brackets stand for averaging over the pattern centers. It is assumed that the noisy background in the patterns has been averaged out during the times δt_v^{α} . Learning is finished when

$$< \dot{\mathbf{w}} > = 0, \quad \text{or} < L > = \text{const.} \qquad (4.3)$$

The second relation holds if the fluctuations $<(L-<L>)^2>$ are sufficiently small which is tacitly assumed. From (4.3), c can be fixed. Thus finally t_{learn} is fixed.

In the recall phase, the weights w_{sl} remain constant. The recall ability depends (i) on the initial value $z(0)$ of the activation state vector, and (ii) on the storage capacity $r = m/q$. It is appropriately characterized by the direction cosine (3.1).

V. A Computer Experiment

We exemplify the behavior of stable adaptive neural nets by pattern pair recognition. The details of the simulation in short hand form are: 2-layer net; 7x7 nodes per layer; weights nonzero only between layers (cf. [5]). Input patterns: 7x7 matrices, displayed in pixel array. Clean patterns (pattern centers $\mathbf{K}_{syst}^{(v)}$) : $z_k \leq -52$ = "white" and $z_k \geq 52$ = "black". Noisy patterns (pattern samples $\mathbf{K}^{(v)}$): add random numbers ε [-60,60] to the clean patterns with activities -52 and 52, respectively. Graphic display in 6 shades of grey by dividing interval [-60,60] into 6 parts. Choice of constants: $C_k < \lambda_{sk}$ causes node activities to change faster than weights ("short term memory" vs. "long term memory"); $\mu_k \neq 1$ introduces multiple time scales into system [3]), in particular $\mu_k \leq 1$ enhances time scale differences between nodes and weights. We choose $C_k = 2.0$, $\lambda_{sk} = 1.0$, and $\mu_k \varepsilon [0.5,1.0]$. Eqs.(2.1) and (2.5) are discretized with time step 0.1.

10 pattern distributions are offered to the system. It should store and retrieve the corresponding clean patterns $K_{syst}^{(\nu)}$. Note that the storage capacity exceeds the Hopfield limit of $r \approx 0.15$ considering that the net is not fully connected. In Figs.1-4 some results of the simulation are displayed. Figs.1-3 are based on a simulation with a highly nonlinear signal function ($T = 0.02$ in (2.4)). Fig.1 shows snapshots of the learning phase. After having learned the pair (V,X), at each time step samples from a new pattern distribution are presented which possesses (7,2) as pattern center. The net stores the clean patterns though it "sees" only noisy versions: In addition to storing, it filters. Fig.2 shows an example from the recall phase. The net sees at each time step the same noisy pattern and relaxes to the clean state. Fig.3 exhibits a more difficult task: Essentially only one pattern half is presented as input, and the net should associate the other half. Here, it becomes even more clear that recall and filtering in the learning phase are completely different processes: For recall the only essential point is the correspondence of the stored patterns to minima of the Lyapunov function. Fig.4 illustrates the behavior of a linearized network ($T = 20.0$ in (2.4)). Figs.4.a and 4.b show a snapshot of the learning phase. It is seen that the filtering ability persists in the linear case. The recall ability, however, is completely lost, as Figs.4.c and 4.d clearly show.

These results can be understood as follows. First to the filtering property. We linearize the signal function (2.4), i.e. $S(z) \approx z/T$. Insertion in (2.1), using $|K_k| = q$ (cf. (2.2)), and $|(1/T)\Sigma_s w_{sk} z_s| \ll |K_k|$, (2.1) becomes

$$\dot{z}_k \approx - C_k z_k + K_k. \tag{5.1}$$

Eq.(5.1) is of Langevin type with a "friction" term $- C_k z_k$ and a "stochastic force" K_k. Averaging over the pattern distribution with center K_k^{syst}, (5.1) becomes

$$\dot{\bar{z}}_k \approx - C_k \bar{z}_k + K_k^{syst} \tag{5.2}$$

where the bar denotes averaging. For large times,

$$\bar{z}_k \approx K_k^{syst}/C_k, \tag{5.3}$$

and hence the activations are proportional to the clean patterns. The filtering ability persists in the nonlinear case.

In the recall phase,

$$\dot{z}_k = - C_k z_k + \Sigma_s w_{sk}^{learn} S(z_s) + K_k, \tag{5.4}$$

where the w_{sk}^{learn} are the weights obtained at the end of the learning phase. The pattern K here is a <u>constant</u> noisy pattern whose clean version is to be recognized by the net. For large times,

$$z_k(\infty) = C_k^{-1}\{\Sigma_s w_{sk}^{learn} S[z_s(\infty)] + K_k\}. \tag{5.5}$$

If the signal function S is sufficiently nonlinear, the net will behave as an associative memory. In contrast, in the linear case

$$z_k(\infty) \approx K_k/C_k, \tag{5.6}$$

i.e. the net will, up to a proportionality constant, only reproduce the constant noisy pattern offered and not relax to the clean pattern. Precisely this behavior is illustrated in Figs.4.c and 4.d. Thus in contrast to <u>filtering</u>, the nonlinearity of the net is indispensable for <u>storing</u> in the learning phase and hence for <u>recall</u> as well.

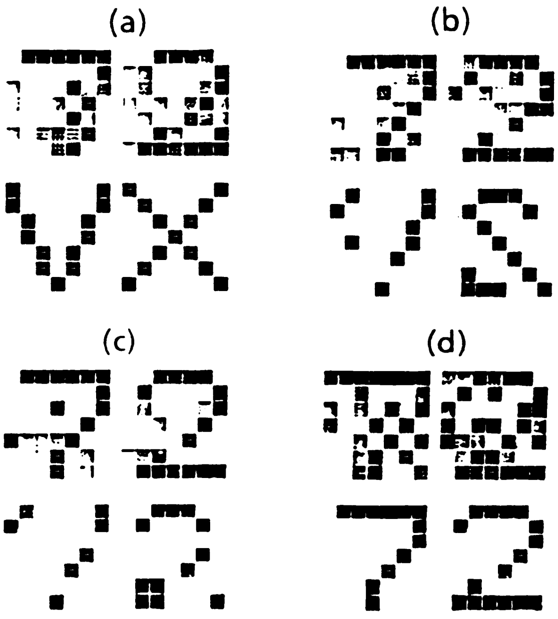


Fig.1. Nonlinear Model.
Learning phase.

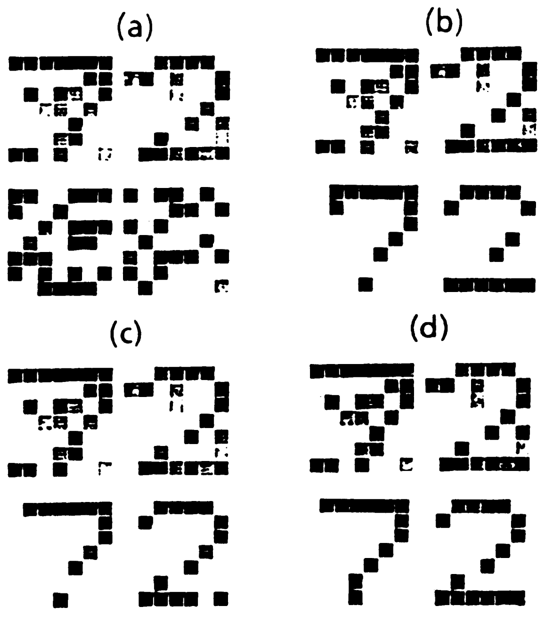


Fig.2. Nonlinear Model.
Recall phase.

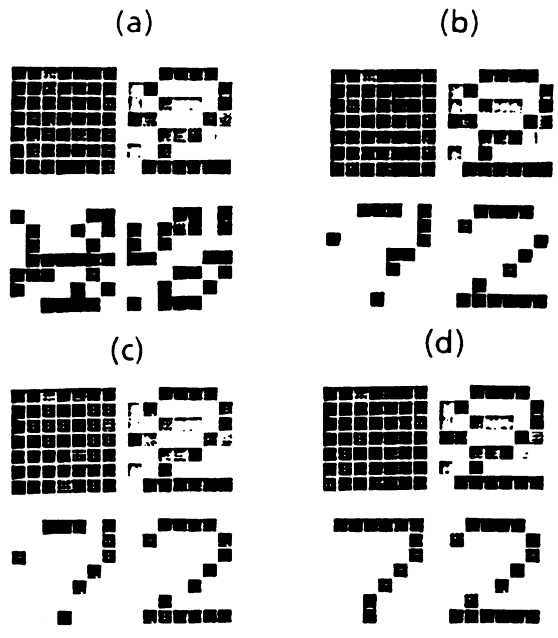


Fig.3. Nonlinear Model.
Recall phase.

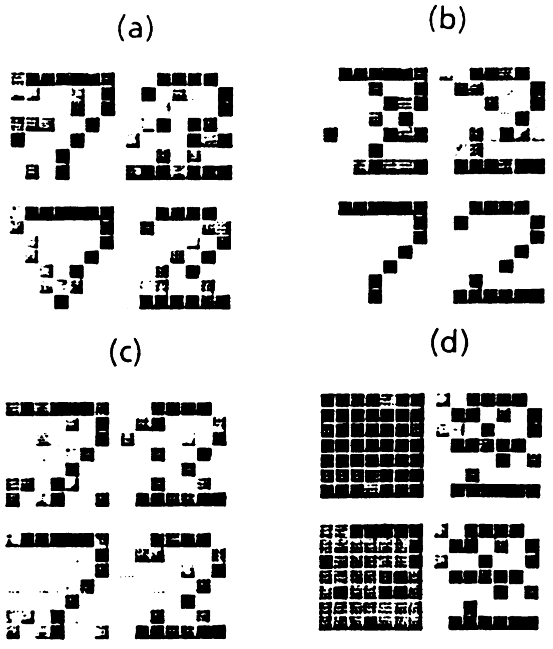


Fig.4. Linear model.
(a),(b): Learning phase. (c),(d): Recall phase.

References

[1]. J. J. Hopfield, Proc. Nat. Acad. Sci. USA 81, 3088 (1984).
[2]. S. Grossberg, Neural Networks 1, 65 (1988).
[3]. B. Schürmann, IJCNN Int. Joint Conf. on Neural Networks, Vol. I, Washington D. C., 91 (1989); to appear in Phys. Rev. A, September 1989.
[4]. S.-I. Amari und K. Maginu, Neural Networks 1, 63 (1988).
[5]. B. Kosko, Applied Optics 26, 4947 (1987).

Self-learning Simulated Annealing

Enrique Carlos Segura - Bruno Cernuschi Frías
Centro de Ingenierías de la Computación y del Procesamiento de la Información
Facultad de Ingeniería - Universidad de Buenos Aires
Mailing Address: Casilla 8 - Sucursal 12(B), (1412) Buenos Aires - Argentina

1. Introduction

In this work, a new "cooling schedule" for the Simulated Annealing (SA) is proposed, introducing a variable parametrization of the search process.

The SA, commonly applied to the minimization of functions with many local minima (over a continuous space or a discrete one of great cardinality), is based on the analogy between that problem and the models of statistical mechanics that study the behaviour of systems of particles with many degrees of freedom at thermal equilibrium, for which, according to the model, the probability of being in the state s is given by the Gibbs distribution:

$$\pi(s) = \exp\left\{-\frac{\varepsilon(s)}{kT}\right\}$$

where ε is the energy associated to that state, and k is the so called Boltzmann constant.

The SA works like thus: starting in a state x, a new state x_p is generated; if it reduces the value of the function, it is accepted as the new state of the system, if not, it will be accepted with probability:

$$\frac{\pi(x_p)}{\pi(x)} = \exp\left\{-\frac{\Delta f}{T}\right\}$$

now being T the control parameter of the process ([1],[2]).

Most of the classical cooling laws, proposed on the basis of theoric results of convergence, include knowledge about the particular problem to be solved, but fail in introducing information about the search process; other proposals, which use as control parameted an estimate of the distance of the current value of the objective function to the minimum, solve partially the problem of getting information about the process, but fall in lack of knowledge about the problem. We are interested in combining the advantages of both groups, managing information about the problem and, at the same time, about the evolution of the search.

2. The Algorithm

Basically, the classic SA works like thus:

```
- Choose $x_0 \in X$;
  $x = x_0$;
  Update ($T$);
- While (certain stop condition is not true)
      {
      Choose $x_p \in N(x)$    /* a neighbor of $x$ */
      Generate a random number $\alpha \sim \mathcal{U}[0,1]$
```

```
/* from the uniform distribution in [0, 1] */
If ($(f(x_p) \leq f(x)$ or $\alpha \leq \exp\{-(f(x_p) - f(x))/T\})$
$x = x_p$;
Update ($T$);
}
```

where $f : X \to \mathbf{R}_+$ is the function to be minimized (maximized), X may be continuous (a subset of $\mathbf{R}^n$) or discret (as in the case of combinatorial problems, e.g. the Travelling Salesman Problem), and $N(x)$ is the set of neighbors of x in X, defined, in the continuous case, as

$$N_r(x) = \{y \in X : \|x - y\| = r\}$$

and, in the discret case, as the set of states that may be generated from x by means of some defined change (for instance, in the TSP, interchanging two cities, or removing a city from a place and inserting it in another place).

The update function determines the way in which the control parameter T (the "temperature") is updated at each iteration of the process. The classical cooling laws ([3],[4]) are exponentially slow. The rule we will obtain here does not depend on the time (i.e. the number of steps); we will see, however, that it contains, in a certain way, the same type of information about the problem as the laws mentioned above.

The two main problems of the "classical SA" are: a) The tendency to get stuck during long times in local minima with deep "valleys": it is the consequence of an early descent of the temperature (freezing) or of a too little value for the initial parameters; b) The tendency to "wander" across the state space: it occurs when the lowering of temperature is too slow or when the initial one is to high.

In order to avoid both problems, we must bind the probability of accepting a candidate when it does not improve the solution, i.e.:

$$\exp\{-(f(x_p) - f(x))/T\} \in (a(x), b(x)) \subset [0, 1] \tag{2.1}$$

In order to determine the way in which $a(x)$ and $b(x)$ depend on x or, more precisely, on $f(x)$, we must consider that they have to be reduced as f goes to zero, but avoiding a behaviour like showed in Fig. 1. We propose:

$$a(x) = \alpha[1 - e^{-(f(x) - f^*_{\min})}] \qquad b(x) = \beta[1 - e^{-f(x) - f^*_{\min})}]$$

where α y β will be input parameters and $f^*_{\min}$ is an estimate of $f_{\min}$, the absolute minimum. This brings the behaviour showed in Fig. 2.

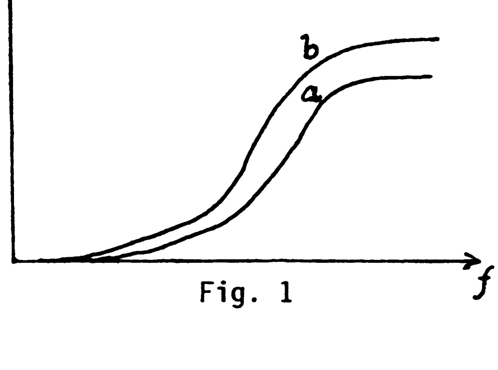


Fig. 1

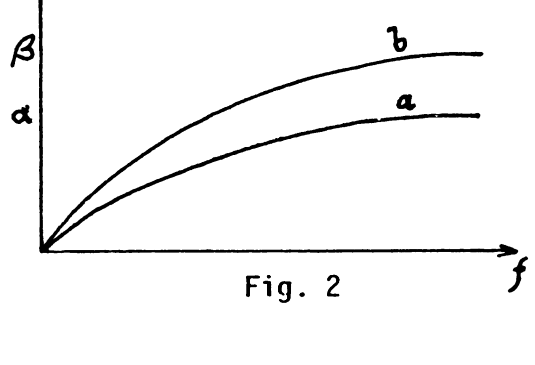


Fig. 2

In order to estimate $f_{\min}$, we must introduce a new parameter, a certain coefficient $h \in (0,1)$ which will be applied to the current value of f.

We may rewrite (2.1) as:

$$T(x)\,\ln a(x) < f(x) - f(x_p) < T(x)\,\ln b(x)$$

Suppose we know a Lipschitz constant for f, defined, in the discrete case, as:

$$L = \max_{x \in X} \left\{ \max_{y \in N(x)} |f(x) - f(y)| \right\}$$

and, in the continuous one, as:

$$|f(x) - f(y)| < L\|x - y\| \qquad \forall x, y \in X$$

It holds that:

$$f(x) - f(x_p) > -Lr$$

(in the discrete case, $r = 1$). Then, the lower bound in (2.1) holds if:

$$T(x) > \frac{-Lr}{\ln a(x)} \tag{2.2}$$

With regard to the upper bound, it will be impossible, in general, to ensure it, since $|f(x_p) - f(x)|$ may be arbitrarily small. Nevertheless, it would be desirable to approach as much as possible the situation of Fig. 3.

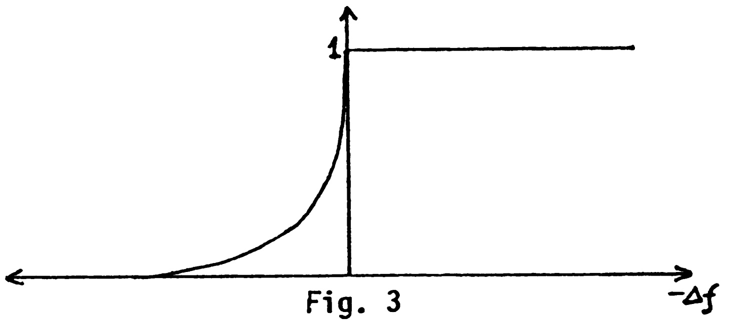


Fig. 3

that is, among the candidates that do not improve the solution, only those with small increases are accepted with a high probability. This implies that in (2.2) we must choose the equality:

$$T(x) = \frac{-Lr}{\ln a(x)}$$

From all this, the update function would work thus:

```
Update
    {
    If ($f(x) \leq f^*_{\min}$)
    $f^*_{\min} = h \cdot f(x)$;
    $T(x) = \frac{-Lr}{\ln a(x)}$;
    }
```

3. Experimental results

a. Our first application was to the minimization of the continuous function given by:

$$f(x,y) = cx^2 + dy^2 - e\cos(\gamma x) - f\cos(\delta y) + e + f$$

with $c = 1$, $d = 2$, $e = 0.3$, $f = 0.4$, $\gamma = 3\pi$, $\delta = 4\pi$ (taken from [5]).

This function has an only absolute minimum $f(0,0) = 0$.

Using the Mean Value Theorem it is easy to find a Lipschitz constant $L = 35$ for the square $|x| \leq 5$, $|y| \leq 5$. Although this may seem very restrictive (we ought to be sure that the minimum is within that square), we will see how that value is still too high for the optimal behaviour of the algorithm. In fact, we begun running the program with $L = 35$, $r = 0.1$, $\alpha = 0.6$ and $x_0 = (-3,5)$, i.e., a point on the edge of the square in which the validity of L was ensured. This did not work well: the search was wandering, purely random; there was not freezing. Then we decided to try with smaller values for L. With $L = 5$, the cooling was smooth and fast; at 800 iterations, we reached the value $f = 1{,}6 \times 10^{-4}$ in $(x,y) = (.0033; -.00037)$ and the descent continued.

Then we tried to solve the same problem with the classic SA, with a cooling schedule of the form

$$T(n) = \frac{c}{\log(1+n)} \tag{3.1}$$

Trying with different values for c, the result was the same in a certain moment, the search got stuck in a local minimun.

Finally, we applied the law

$$T(f) = \exp\left(\frac{-\Delta f}{f}\right) \tag{3.2}$$

i.e., generalized SA (as proposed in [5]). There was no problem of getting stuck in local minima, but now we observed the opposite problem: there was not freezing, so the only way in which the search was relatively successful was keeping memory at each iteration of the minimum value obtained for the function.

Then we changed slightly the function by adding a constant to it, so the minimum was now $f(0,0) = 5$. We applied our method with the same values as in the former case and $h = 0.99$. The results were satisfactory, although not as good as in the first case; the cooling was indeed "in stages"; therefore, periodically it was observed some wandering. However, the current value for f never exceeded the minimum found in more than the difference between that minimum and the absolute one. A value of $f = 5{,}0014$ in $(x,y) = (.0053; .0056)$ was obtained at 900 iterations.

The application of the other two cooling laws ((3.1) and (3.2)) produced behaviours similar to those in the first case.

b. Then we applied our algorithm to a discret problem: the TSP. This is a classical NP-complete problem, which consists of the following: given a set of N cities, how to visit every city once and only once, returning to the starting point and minimizing the total tour length. It is a problem of over- exponential complexity and no deterministic algorithm is known that solves it in polynomial time; so it is an ideal candidate for an stochastic optimization method and, in fact, it has been the classical problem for testing the performance of SA.

We took a map with 100 cities arranged in a quadrangular lattice with side length equal to the unity, considering the Euclidean distances between the cities. The optimum tour has clearly length 100. Here the performance was best than with both (3.1) and (3.2), but the difference was less significative. While a 50000 iterations the value of the function with (3.1) was 110,45 and with (3.2) it was 109,93, with the "self learning" version the value reached at the same number of iterations (with $h = 0.7,\ L = 1,\ \alpha = 0.5$) was $f = 108.52$.

4. Conclusions.

From the simulations we deduce that the algorithm really makes a sinthesis of two sources of knowledge: the inherent properties of the problem and the evolution of the searching process.

Although the practical results are best than those obtained with other cooling laws, we must not some divergencies between theoretical prescriptions and practical facts, specially concerning the parameter L. In 3.a the theoretic value found for L was $L \simeq 35$; but the value with which the algorithm worked was 5; in 3.b, the theoretic value was $L \simeq 26$, while the optimal performance was obtained with $L = 1$. This problem is analogous to that of the schedules of the type 3.1 the theoric value of c that ensures the convergence to the optimum is unapplicable because it makes the search exponentially slow. The most familiar case is the curve proposed in [2]

$$T(n) \geq \frac{rL}{\log(n + n_0 + 1)} \tag{4.1}$$

where L is the Lipschitz constant, r is a measure related to the radius of the graph (in the discrete case) and n_0 is a parameter which controls the initial temperature. The condition (4.1) warrants the strong ergodicity of the Markov Process associated to the SA; but setting the temperature at that value turns the process virtually a SA at constant temperature and produces, therefore, a purely random search. Just like the problem pointed out in 3.

References.

[1] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A.H. and Teller, E., Equation of state calculations by fast computing machines. J. Chem. Phys., Vol.21, N° 6, 1087-1091, 1953.

[2] Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P., Optimization by Simulated Annealing. Science, Vol.220, 671-680, 1983.

[3] Geman, S. and Geman, D., Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of Images. IEEE PAMI 6, N° 6, 721-741, 1984.

[4] Mitra, D., Romeo, F. and Sangiovanni-Vincentelli, A., Convergence and finite-time behaviour of Simulated Annealing. Adv. Appl. Prob. 18, 747-771, 1986.

[5] Bohachevsky, I., Johnson, M. and Stein, M., Generalized Simulated Annealing for function optimization, Technometrics, Vol.28, N° 3, 209-217, 1986.

ASSOCIATIVE MEMORY SYSTEMS

Patrick K. Simpson

General Dynamics
Electronics Division
P.O. Box 85310, MZ 7202-K
San Diego, CA 92138

INTRODUCTION

Binary and bipolar valued correlation (Hopfield) associative memories have an inherent storage capacity limitation that logarithmically decreases as the dimensionality increases (cf. McEleice, et al., 1987; Amari & Maginu, 1988). We present a refinement of earlier work (Simpson, 1989) that circumvents this storage capacity limitation by employing a system of associative memories that partitions the ensemble of data patterns into orthogonal pattern sets and stores the resultant sets into individual memories. By employing this encoding scheme we can drastically improve the capacity performance of each individual memory from logarithmically decreasing to linearly increasing with increasing dimensionality.

The motivation for this work was the storage capacity bottleneck. There are several VLSI and optical implementations of the Hopfield associative memories that are not being used because their storage capacity is too low. Our system presents an alternative in that we can now use current associative memory implementations as components of a slightly more complex system that will provide far greater storage capacity.

ASSOCIATIVE MEMORY SYSTEM ENCODING AND RECALL

The encoding algorithm performs the partitioning and encoding of the ensemble of data patterns in a single pass through the data. The recall algorithm operates in a nearest neighbor fashion by selecting the best response from the ensemble of memories.

<u>Variables</u>:

Assume an ensemble of m data patterns $\underline{A} = \{A_1, A_2, ., A_m\}$ where each A_k is an n-dimensional bipolar valued row vector $A_k = (a_{k1}, a_{k2}, ., a_{kn})$. $\underline{A}$ can be partitioned into M orthogonal sets $\underline{A} = \{\underline{\alpha}_1, \underline{\alpha}_2, ., \underline{\alpha}_M\}$, where M will be no larger than $2^n/n$. Each orthogonal set $\underline{\alpha}_h = \{\alpha_{h1}, \alpha_{h2}, ., \alpha_{h\beta}\}$ is encoded into its own associative memory W_h, where β is the number of orthogonal patterns in $\underline{\alpha}_h$. The variables used during the encoding and recall process are the following.

A_k	kth pattern, $A_k \in \{-1,+1\}^n$
m	number of patterns
W_h	hth associative memory (n-by-n matrix)
M	number of associative memories
Γ	flag signalling proper/improper encoding of Ak

E^*	orthogonal energy (see appendix for definition)
X	temporary n-element pattern vector
T	temporary n-by-n associative memory matrix
A	input pattern presented to each Wh during recall
X_h	stable pattern recalled from Wh given A
μ	index of the memory with the best response

<u>Functions</u>:

The recall operation for each associative memory is represented as a function. This operation, described in detail by Hopfield (1982) and McEleice, et al. (1987), is guaranteed to be stable for any arbitrary number of patterns in a finite number of iterations. In general, we assume a synchronous update procedure the employs a bipolar step function. This operation is defined as a function by the following:

recall(W,A)	associative memory recall function that is passed an n-by-n associative memory matrix and a vector $A \in \{-1,+1\}^n$ and returns the resonant pattern X. See Hopfield (1982) for details.

<u>Encoding Algorithm</u>:

The following algorithm begins with one n-by-n associative memory W_1. As more memories are needed they will be added therefore the ensemble of associative memories $\underline{W} = \{W_1, W_2, ., W_M\}$ will grow as needed. The entire partitioning and encoding process takes place in a single pass through $\underline{A}$.

```
$M = 1, k = 1$

while $k \leq m$ do

    $T = A_k^T A_k, \Gamma = FALSE, h = 1$

    /* Store $A_k$ in first $W_h$ that has orthogonal energy */

    while $h \leq M$ and $\Gamma = FALSE$ do

        $\Gamma = TRUE, W_h = W_h + T$

        if $|A_k W_h A_k^T - E^*| > 0$ then

            $W_h = W_h - T, \Gamma = FALSE, h = h+1$

        else

            $h = M+1$

        endif

    endwhile
```

```
        /* If $\Gamma$ = FALSE then store $A_k$ in a new assoc. mem. */
        if $\Gamma$ = FALSE then
            M = M+1, $W_M$ = T
        endif
        k = k+1
    endwhile
```

Recall Algorithm:

The recall operation requires selecting the appropriate memory from which the response pattern will be drawn. Knowing that each memory is constructed orthogonally will be used to determine which associative memory is most appropriately selected for the response. The following recall algorithm will select the best response pattern, X_μ, given an arbitrary input pattern, A, in a single pass through the associative memories.

```
    $\mu = 1$
    for h = 1 to M do
        $X_h$ = recall($W_h$,A)
        if $(|AW_\mu A^T - X_\mu W_\mu X_\mu^T| + |X_\mu W_\mu X_\mu - E^*|)$
         $< (|AW_h A^T - X_h W_h X_h^T| + |X_h W_h X_h - E^*|)$ then $\mu = h$
    endfor
```

CONCLUSIONS

By utilizing an ensemble of associative memories we are able to store far more patterns in each associative memory which means much more information is being stored per connection. As an example, using the criteria of McEleice, et al., a single associative memory could perfectly store 11 randomly selected 100-dimension bipolar patterns -- yielding a storage density of $11/100^2 = 0.0011$ patterns per connection. On the other hand, an associative memory system could store as many as 100 patterns per associative memory -- yielding a storage density of $100/100^2 = 0.01$ patterns per connection. This is an improvement in efficiency by a factor of almost 10 for only moderately large patterns. As the dimensionality increases, the improvement is even more dramatic. In addition, an associative memory system does not require complete retraining when a new pattern must be incorporated into the existing knowledge base. New patterns can be encoded immediately and efficiently -- a feature several existing neural networks lack.

APPENDIX: ORTHOGONAL ENERGY

When all the patterns stored in an associative memory are members of an orthogonal set then there is a simple expression that describes the energy of the system.

Assume that the dimensionality of the patterns being stored is even. The correlation association associative memory is constructed using the equation

$$W = \sum_{k=1}^{m} A_k^T A_k, \qquad \text{(A-1)}$$

where each A_k is an n-dimensional bipolar valued row vector $A_k \in \{-1,+1\}^n$. The energy of this system is defined as

$$E \equiv -AWA^T, \qquad \text{(A-2)}$$

where $A \in \{A_k: k = 1, 2, ..., m\}$. We can rewrite equation A-2 as

$$E = -A \left(\sum_{k=1}^{m} A_k^T A_k \right) A^T$$

$$= - \sum_{k=1}^{m} AA_k^T A_k A^T$$

$$= - AA_k^T A_k A^T - \sum_{j<>k}^{m} AA_j^T A_j A^T, \qquad \text{(A-3)}$$

which yields the orthogonal energy relationship

$$E^* = -n^2. \qquad \text{(A-4)}$$

As might be expected, the energy of a system with all patterns stored orthogonally is only dependent on the number of dimensions in the system and not the number of patterns stored.

REFERENCES

Amari, S-I. & Maginu, K. (1988). Statistical neurodynamics of associative memory, Neural Networks, 1, 63-74.

Hopfield, J. (1982). Neural networks and physical systems with emergent collection computational abilities, Proceedings of the National Academy of Sciences, 79, 2554-2558.

McEleice, R., Posner, E., Rodemich, E. & Venkatesh, S. (1987). THe capacity of the Hopfield model, IEEE Transactions on Information Theory, IT-33, 461-482.

Simpson, P. (1989). Bidirectional associative memory systems, Heuristics, Vol. 1, no. 2., pp. 50-59.

EQUILIBRIUM UNIQUENESS AND GLOBAL EXPONENTIAL STABILITY OF A NEURAL NETWORK FOR OPTIMIZATION APPLICATIONS

S. I. Sudharsanan and **M. K. Sundareshan**
Department of Electrical and Computer Engineering
University of Arizona
Tucson, AZ 85721

1. INTRODUCTION:

In the recent past, several innovative results have appeared in the literature that demonstrate the use of neural networks in providing a computational architecture of choice for handling complex optimization problems. A number of very versatile networks have been reported to perform optimization in various applications.

The key to the successful application of a neural network to solve optimization problems lies in its convergence properties. Analysis of the qualitative properties of Hopfield-type networks (and their generalizations), in particular their stability properties, has been investigated in a number of articles [1-3]. Most of these results, however, are developed in the context of using these networks as associative memories. There exist some very fundamental differences in the types of qualitative properties that one would be interested in examining in order to serve as guidelines for tailoring the neural network which is intended for eventual use either as an associative memory or for optimization purposes. In particular, for problems of designing a neural network to serve as an associative memory, the analysis of the recall abilities of a set of stored vectors in the state space is of importance and consequently the synthesis question is one of adjusting the network parameters such that the network has multiple equilibrium points corresponding to the desired vectors to be recalled from the memory. Also, the stability conditions of interest are only of a local nature that ensure asymptotic convergence to an equilibrium point when the network is started from an initial state in the vicinity of that equilibrium point.

On the other hand, for neural networks that are intended to be used for solving optimization problems, construction of the network with a unique equilibrium point that corresponds to the global optimum of the objective function is highly desirable, to prevent convergence to the local minima which may generally be far from the global optimum conditions. In this case, conditions for the global asymptotic stability of the network that ensure convergence of the solutions starting from any initial state to the unique equilibrium point , and also an estimate of the exponential convergence rate, are of particular importance. The synthesis question of interest then is one of selecting network parameters such that the network possesses a unique equilibrium point and an exponential convergence with a specified rate to this point (i.e. exponential stability with a prescribed degree [4]) is ensured. In this paper, we shall report the results of a qualitative analysis aimed at obtaining such conditions for a neural network that can be programmed to solve optimization problems.

2. NEURAL NETWORK MODEL:

We consider a modified Hopfield-type neural network structure whose application to solving least squares estimation problems has been demonstrated recently [5]. For an N - neuron network, the dynamics are described by

$$\frac{du_i}{dt} = -\frac{1}{\alpha_i}u_i + \sum_{j=1}^{N} t_{ij}g(u_j) + b_i \quad , i = 1, 2, \cdots, N \tag{1}$$

where u_i is the input to neuron i, $t_{ij} = t_{ji}$ is the synaptic interconnection strength between the i-th and the j-th neurons, and b_i is the external bias input. Further, $\alpha_i > 0$ is an RC-time constant and $g(u_i)$ is a monotonically nondecreasing function specifying the input-output characteristic of each neuron in the form

$$v_i = g(u_i) = \begin{cases} \beta u_i, & |u_i| \leq B \\ \beta B, & u_i > B \\ -\beta B, & u_i < -B \end{cases} \tag{2}$$

where β and B are finite positive constants. The equilibrium points of the dynamical equations (1) and (2) correspond to the minima of an associated energy function

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(t_{ij} - \frac{\delta_{ij}}{\alpha_i \beta})v_i v_j - \sum_{i=1}^{N} b_i v_i \tag{3}$$

where δ_{ij} is the Kronecker delta. E in (3) indicates the usefulness of the network described by (1) and (2) for quadratic optimization problems.

It should be mentioned that networks of this type have been shown to be realizable by analog electronic circuits with $g(u_i)$ being realized by an amplifier. This model differs from the network proposed by Hopfield in $g(.)$ being a piecewise linear function rather than being a strictly increasing sigmoidal function.

3. QUALITATIVE ANALYSIS:

Of fundamental importance in the synthesis of a network for optimization applications is the establishment of conditions for characterizing the equilibrium conditions and for guaranteeing convergence to a steady-state equilibrium following a transient regime. Specifically, we are interested in the following questions: Are the solution trajectories starting at different initial states remain bounded for all times? How can we guarantee the existence of equilibrium points at certain desired operational ranges? What are the structural conditions that ensure an unique equilibrium point? What types of relations between the network parameters ensure exponential convergence to an equilibrium condition irrespective of the choice of initial conditions to start the network? Answers to these questions will be summarized in this section in the form of theorems. Proofs of these will be omitted due to page restrictions.

Theorem 1: For the network described by (1) and (2), a solution for any specified initial condition (i.e. $u_i(t_0)$) exists and is unique.

Theorem 2: The solution of (1) starting at any initial state $u_i(t_0)$ is bounded for all $t \geq t_0$.

It is evident that the locations of the equilibrium points in $\Re^N$ and their characteristics are determined by the interconnection pattern of the neural net (i.e. by the parameters t_{ij}) as well as by the parameters α_i and the constants β and B defining the neuron input-output characteristic. For the sake of simplicity in stating the following results which provide a characterization of the equilibrium points of the network defined by (1) and (2), consider the compact set $\mathcal{S} \subset \Re^N$ defined by

$$\mathcal{S} = \{\mathbf{u} : -B \leq u_i \leq B, \ i = 1, 2, \cdots, N \}, \tag{4}$$

where $\mathbf{u} \in \Re^N \ni \mathbf{u} = [u_1, u_2, \ldots, u_N]^T$. We can now state a theorem which confines the equilibrium points of (1) to a desired operational range corresponding to the linear region of the input-output function $g(.)$ with slope $\beta > 0$.

Theorem 3 : Let the following condition hold:

$$B\left[1 - \alpha_i\beta\sum_{j=1}^{N}|t_{ij}|\right] - \alpha_i|b_i| \geq 0 \;\; \forall\, i = 1,2,\cdots,N\;. \tag{5}$$

Then the equilibrium points of (1) lie on or inside $\mathcal{S}$.

As mentioned earlier, for neural networks intended to be used for solving optimization problems, establishment of conditions that ensure an unique equilibrium point that corresponds to the global optimum of the objective function is highly useful. This is due to the fact that if the network is programmed to be asymptotically stable, then starting from any feasible initial state, the network will converge to the same stationary point solution in the solution space. The following result enunciates the conditions for an unique equilibrium point for the network under consideration.

Theorem 4: If the equilibrium points of the network defined by (1) and (2) are in $\mathcal{S}$ and the matrix $W \in \Re^{N\times N}$ given by

$$W = A - \beta T$$

$A = \text{diag}\left[\frac{1}{\alpha_1}, \frac{1}{\alpha_2}, \cdots, \frac{1}{\alpha_N}\right]$ and $T = [t_{ij}]_{i,j=1,2,\cdots,N}$, is nonsingular, then the network has a unique equilibrium point.

Having obtained the conditions that ensure the existence of a unique equilibrium point in a desired operational range of the network, what remains is to tailor the parameters such that exponential convergence to this equilibrium point is guaranteed from any feasible initial condition. Towards this end, we will give the following definition and an exponential stability result.

Definition: The equilibrium point $x^* = 0$ of the dynamical system

$$\dot{x}(t) = f(x(t)), f(0) = 0,$$

where $x(.) : \Re \rightarrow \Re^N$ and $f : \Re^N \rightarrow \Re^N$, is *exponentially stable with degree* η if every trajectory starting at any feasible initial state $x(t_0) = x_0 \in \Re^N$ satisfies the condition

$$||x(t)|| \leq \pi||x_0||\exp(-\eta(t - t_0)) \quad \forall\, t \geq t_0 \tag{6}$$

where π and η are positive constants independent of the initial conditions (t_0, x_0) and $||.||$ is the $\mathcal{L}_2$-norm.

Theorem 5: Let $\tilde{\eta}$ be a number selected in the range $0 \leq \tilde{\eta} \leq 1/\bar{\alpha}$ where $\bar{\alpha} = \max_i \alpha_i$. Then the equilibrium point u_i^* of the neural network described by (1) and (2) is exponentially stable with degree $\tilde{\eta}$ if the vector $h(.) : \Re^N \rightarrow \Re^N$ defined by

$$\begin{aligned} h(\xi) &= [h_1(\xi), h_2(\xi), \cdots, h_N(\xi)]^T \\ \xi &= [\xi_1, \xi_2, \cdots, \xi_N]^T,\; \xi_i \in \Re \;\; \forall\;\; i = 1,2,\cdots,N \\ h_i(\xi) &= \sum_{j=1}^{N} t_{ij}\tilde{g}(\xi_j) = \sum_{j=1}^{N} t_{ij}[g(\xi_j + u_j^*) - g(u_j^*)] \end{aligned} \tag{7}$$

can be factored in the form

$$h(\xi) = [U(\xi) - S(\xi)]P\xi \tag{8}$$

where $P \in \Re^{N\times N} \ni P = \mathrm{diag}[p_{11}, p_{22}, \cdots, p_{NN}]$ with $p_{ii} = 0.5\alpha_i/(1-\tilde{\eta}\alpha_i), U : \Re^N \to \Re^{N\times N}$ is an arbitrary skew-symmetric matrix, $S : \Re^N \to \Re^{N\times N}$ is an arbitrary symmetric matrix that satisfies the inequality

$$\xi^T[I + 2PS(\xi)P]\xi \geq 0 \quad \forall\, \xi \in \Re^N\,, \tag{9}$$

I being the $N \times N$ identity matrix.

4. DISCUSSION OF RESULTS:

(A) The various results presented in this section serve as useful guidelines for synthesizing a neural network with desired properties. The conditions enunciated, although only sufficient, are not overly restrictive and could be used in a number of ways. In particular, they can be used to develop systematic construction procedures for selecting the network parameters to yield a globally exponentially stable network whose unique equilibrium point corresponds to the global minimum of an appropriately formulated objective function in an optimization problem. An illustration of this approach to construct explicit algorithms for programming the neural network to solve two estimation problems (viz. parameter estimation by a recursive least squares procedure and maximum *a posteriori* state estimation in a dynamical system with noise corrupted outputs) is given in [5}.

(B) It should be noted that the present result for exponential stability is valid irrespective of the size of the network (number of neurons) and holds so long as the interconnection pattern satisfies the required condition.

REFERENCES

[1] M. A. Cohen and S. Grossberg, "Absolute stability of global pattern formation and parallel memory storage by competitive neural networks," *IEEE Trans. Syst. Man, Cybern.*, vol. SMC-13, pp. 815-826, 1983.

[2] J.-H. Li, A. N. Michel and W. Porod, "Qualitative analysis and synthesis of a class of neural networks," *IEEE Trans. Circuits Syst..*, vol. CAS-35, pp 976-986, 1988.

[3] A. N. Michel, J. A. Farrel and W. Porod, "Qualitative analysis of neural networks," *IEEE Trans. Circuits Syst.*, vol. CAS-36, pp 229-243, 1989.

[4] W. Hahn, *Stability of Motion*, New York: Springer Verlag, 1968.

[5] S. I. Sudharsanan and M. K. Sundareshan, "Neural network computational algorithms for least squares estimation problems," Presented at *International Joint Conference on Neural Networks*, Washington D.C., June 1989.

CONNECTIONIST FINITE STATE MACHINES. Claude Touzet and Norbert Giambiasi . L.E.R.I., Parc Scientifique Georges Besse, F-30000 NIMES FRANCE

Sequential problems like speech understanding or speaking, robotic control, trajectory recognition, begin to be explored by increasing number of researchers in neural networks.

Lot of previous connectionist approaches have treated problems of sequential nature by a spatio-temporal transformation (for example, the use of a window of 7 characters in NETtalk to handle the context). Others approaches really implement a notion of state, in terms of automata's state (1), but in most cases, the state can not be easily manipulate or observe (2, 3).

In this paper, using the definition of a finite state machine, i.e. the quintuple : Input, State, Output, state function, output function (4), we are investigating neural systems exibiting finite state machines behavior.

First, we present the formal definition and description of a neural network seen as a synchronous sequential machine. In addition, in many practical situations, the synchronizing clock pulses are not available, we present the basic model of neural network for asynchronous sequential state machines too.

Based on multi-layered neural nets and backpropagation learning algorithm, we then introduce :

(a) A neural network model of the state machine : the connectionist state machine.
(b) A neural network model of the synchronous sequential machine : we have designed a specific neural memory device, counterpart of a bistable in the digital domain, used in the feedback loops.
(c) A neural network model of the asynchronous sequential machine.
(d) Implementions of Mealy and Moore connectionist machines.

In the third part, experiments on sequences storage and recall with connectionist state machine are described. The model of a connectionist state machine ables to store a sequence of patterns shows two sets of inputs : inputs of the pattern and inputs of the state and two sets of ouputs : outputs of the response and outputs of the next state. Recurrent connections allow the state to be update (i.e. the next state becomes the state for the next pattern presentation). Using SACREN*, a neural network event driven simulator, simulations show that this model can remenber multiple sequences.

In conclusion, we illustrated that based on this approach further developments may give reasonning capabilities to neural networks.

References

[1] T. Kohonen, "Self-Organization and Associative Memory", 2nd Edition, Springer Series in Information Sciences, Vol 8, Springer Verlag, Berlin 1987, ch. 1, pp 16-21.
[2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," in "Parallel Distributed Processing : Exploration in the Microstructure of Cognition," D. E. Rumelhart, J. L. Mac Clelland and the PDP Research Group, Vol 1 : Foundations, Cambridge : MIT Press, 1986, ch. 8, pp 318-362.
[3] S. I. Gallant, "A Neural Network Model for Sequential Task," First INNS Meeting, Boston, 1988.
[4] Koavi, " Switching and Finite Automata Theory," McGraw-Hill, 1978, part 3.

*Developed under ANVAR contract n° A8801006JAL.

Phase Space Diagrams: Towards a Useful Characterization of Network Behaviour.

Ronan Waldron
Department of Computer Science
Trinity College, Dublin
Ireland

Abstract

A taxonomy of neural networks, as dynamical systems, is introduced. The approach is a synthesis of current Liapunov methods, and more general dynamical system analysis methods. The combination allows for the application of an extensive battery of mathematical techniques to the theoretical study of network dynamics, while at the same time ensuring that the results are as accessible as possible.

1 Introduction

Neural network activity is conventionally displayed by selecting a hopefully relevant instant in time, and then plotting a two dimensional projection of the network structure. The particular values of the nodal activities are often plotted as black or white squares, of varying areas. Several of these diagrams will give a 'complete' description of the network's history. The central problem with this approach is the difficulty of visualising, and hence understanding, the network's dynamics. Given that a neural network is essentially a dynamical system, a comprehensive view of its dynamics, for all values of exogenous[1] variables is desirable.

Our central concern in this paper will be with the provision of a broader perspective on the study of network dynamics. The approach will be from the normal time series and *configuration* representations of network behaviour, to a *phase space* representation, and so a taxonomy of network structures. It is felt that this pictorial classification of networks will make the design of architectures much more accessible to the uninitiated, and so will bring the common-place understanding of neural networks that much closer.

2 Time Series, and Configuration Space.

A network's configuration at any one time may be identified by an n-tuple of numbers—the activities of the nodes in the network at that time. These n-tuples may be viewed as the coordinates of a point in n-dimensional Euclidean space. This space is the configuration space of the network. A typical system will have the dynamics encoded as a set of differential equations. For example, see the simple network below:

$$\frac{dx_i(t)}{dt} = -A_i x_i + \sum_{j=1}^{N} B_{ij} z_{ij} \tag{1}$$

$$\frac{dz_{ij}(t)}{dt} = -D_{ij} z_{ij} + E_{ij} x_i \tag{2}$$

[1]Exogenous variables, also referred to as constitutive parameters, are those values, usually constant within a simulation, that define the particular system.

B_{ij} and E_{ij} are signal functions. The set $(x_i(t), z_{ij}(t))$ defines the network configuration at any one time. The configuration space may be viewed as a vector space, and the point denoted by $\vec{x}$. To capture the temporal behaviour of the network, *as a whole*, the gradient of the vector may be plotted as an arrow in the vector space. The gradient is given just by Equations 1 and 2. The need for carrying out a simulation for each set of initial conditions is done away with; the network's dynamics may be understood by looking at the diagram as a whole.

The utility of this mechanism for large networks, with some thousands of endogenous[2] variables, appears to be limited by the dimensionality of the space. This is a problem intrinsic to the study of many dimensional phenomena. There are many different approaches to reducing the complexity at hand. The first such approach we consider is the Liapunov method (see [2] for a detailed review).

The Liapunov method is a method for the global analysis of network dynamics. The function provides a single variable that describes the behaviour of the system as a whole. The principle of minimization of the Liapunov function enables us to see the global dynamics of the network. The problem of the high dimensionality of the space is now circumvented. We may plot the Liapunov function against time, as a simple two-dimensional plot, and so watch the global behaviour. In this case, the system has been greatly simplified, and can readily approach the problem of the global behaviour of the network. For example, the Liapunov method for content addressible memory networks allows the visualisation of the network as a flow, into a set of vortices, or attractors. The vortices define a set of equivalence classes on the space of initial conditions. To ensure error-free memory is equivalent to ensuring disjoint equivalence classes. To ensure rapid access, is to ensure strong attractors, or equivalently, deep vortices.

For an illustration of the two different kinds of information conveyed, see the diagrams. Figure 1 shows the timeslice of the network after some thirty cycles. The convergence of the synaptic weights can clearly be seen, as can the contrast enhancement. The same network is displayed in Figure 2, but this time a small section of the trajectory through configuration space is displayed.

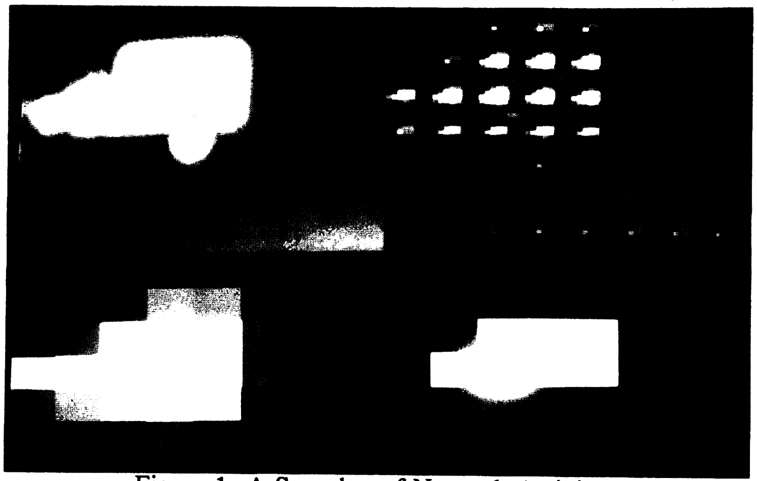


Figure 1: A Snapshot of Network Activity.

[2]Endogenous variables are variables internal to the system. As such they would include all the nodal activities, and synaptic weights.

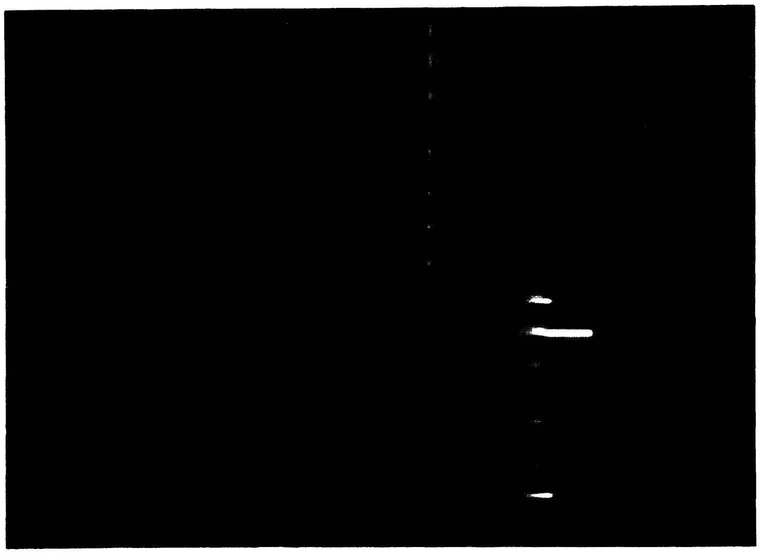


Figure 2: A Segment of a Trajectory in Configuration Space.

3 Network Dynamics

There are several ways of approaching the problem of understanding network dynamics. The simplest is to simulate a discrete system of equations that either constitute the network in question or that approximate the continuous dynamical system under investigation. Models of real neural systems are constrained to be continuous, and hence our central concern in this paper is with these. To understand the complex dynamics of a large system, it is inefficient to take the initial conditions as the exogenous variables, as then each particular system within a class requires many simulations, one for each set of initial conditions. A configuration space diagram of the system overcomes this problem, as it simultaneously displays the dynamics of the network for all initial conditions.

The configuration space approach is intrinsically based on the *temporal* evolution of the system in question. To model the behaviour as a function of time requires the solution of the system of equations that link the activities. This is invariably impossible, once non-linear mechanisms have been introduced. Numerical simulation has a limited generality. The same problem appears in the solution of even simple physical dynamical systems. To attain a picture of the global behaviour of the system, a common approach is to transfer the problem to *phase space*[3]. Phase space is a multi-dimensional space, with the coordinate axes defined by the set of all variables on which the future state of the system depends. For example, in the above case $(x_i(t), E_{ij}(t), z_{ij}(t), B_{ij}(t))$ would be sufficient. This set gives the entire dynamics of the system, as if it were a physical dynamical system. Time does not appear explicitly in this formalism. It is the *topology* of the space that enables us to describe the motion. If we identify the Liapunov function, for example, as the linkage relation between the endogenous variables, then we may solve the system for the orbits.

The obvious question is what is the utility of transferring to the phase space? The variables do not represent the actual state of the network, and so simplifications creep in. The signal functions may be non-linear, but this drops out of the system of equations, once they are rewritten for phase space. The powerful aspect of this sort of representation is the ability to study complex continuous dynamical systems. To understand the continuous dynamics fully, we must understand the properties of the phase space as a continuous medium. This can be achieved by the standard topological approach of mathematical analysis. We will then have the computer simulation, giving the layman a clear picture of the dynamics, and a

[3]For an intuitive introduction to phase spaces, see [1].

detailed, mathematical gaurantee of the validity of the picture. This overcomes the ever present problem with simulating networks—there are always points, or degrees of accuracy, which have been ignored possibly with catastrophic results.

4 Phase Space As a Taxonomic Mechanism

A central problem in neural network research, and in its application, is that of the design of a particular network for a given application. At present, the bulk of the work has to be done by an expert, who is familiar with every aspect of a diverse range of paradigms. The difficulty of training a programmer, for example, to this level of expertise makes neural network applications as yet uncompetitive. To facilitate the design problem, a taxonomy of the various paradigms is suggested.

The basic idea follows from the work described above. The constitutive parameters, referred to above as exogenous variables, also define a multi-dimensional Euclidean configuration space. The reason for introducing the somewhat more obscure terminology for the two classes of variables was to highlight just this parallel. A particular paradigm may be encoded into a Cartesian product of the exogenous configuration space, and the endogenuous phase space. This diagrammatic representation is accessible to the layman, comprehensive, and verifiable mathematically.

If the network's central features may be captured in a single function, or in a general topological property of the phase space, then the necessity for the representation of the network at that level of detail is done away with. The illustration above of the visualization of content addressible memory as a flow towards a set of vortices hopefully serves to indicate the power of this method.

5 Conclusion

Much rigorous mathematical treatment of neural network dynamics has been done. The inaccessibility of the results to the general audience limits the utility of this approach. No one will dispute the merit of having a mathematical proof that a network does in fact do what it seems to do. It is felt that the introduction of phase space diagrams will make this kind of analysis much more accessible.

If a rigorous taxonomy is possible, by means of the phase space representation of the system, then the design of complex networks may be achieved by the synthesis of a selection of the mechanisms, characterised in the diagrams.

The utility of a small number of functions which characterise the dynamical behaviour of a network is evident. It is felt that the topological properties of the phase space are sufficiently succinct to encapsulate much information in a compact fashion. More work remains to be done on characterising the topological aspects of various network models. However, as a mechanism for understanding the complex dynamics of the networks, it is felt that it will prove to be invaluable.

References

[1] I. Stewart. *Does God Play Dice? The mathematics of chaos.*. Basil Blackwell, Oxford, England, 1989.

[2] S. Grossberg. Non-Linear Neural Networks: Principles, Mechanisms and Architectures. *Neural Networks*, 1:17–61, 1988.

Disproof of Two Conjectures on Capacity of Hopfield Associative Memories

Xin Wang

Department of Mathematics, University of Southern California

Los Angeles, CA 90089

Abstract. In this paper, we disprove the following two conjectures: for a Hopfield associative memory of n neurons, (1) it is impossible to have a number of memory patterns between $2n+1$ and $2^n - 1$ [7]; and (2) the maximal number of learning patterns which can be memorized as the *only* memory patterns is cn^2 for some constant $c > 1$ [11].

I. Introduction

The neural network we consider is a system of n interconnected neurons with possible states 1 and -1, connection weight matrix W and zero threshold. The network updates its states *asynchronously* (*i.e.*, only one neuron i is chosen randomly with a equal probability to update its states at each time), according to the following rule: if x is a current state of the network and x_i^+ denotes the next state of neuron i, then

$$\begin{aligned} x_i^+ &= \begin{cases} +1 & \text{if } \sum_j W_{ij}x_j > 0 \\ x_i & \text{if } \sum_j W_{ij}x_j = 0 \\ -1 & \text{if } \sum_j W_{ij}x_j < 0 \end{cases} \\ &= sgn(Wx)_i \end{aligned}$$

Such a network is considered as a model of associative memory in the following sense. We use the n-hypercube $\mathcal{B}^n = \{-1,1\}^n$ as a pattern space with each vector being a pattern. A pattern v is called a *memory pattern* if v is an equilibrium state of the network, *i.e.,* $v = sgn(Wv)$. A memory pattern v is said to be recalled from a pattern x if, starting at x, the network will be led to (equilibrium state) v. We will denote the neural network $AM(W)$ as the associative memory with weight matrix W, the set of all equilibrium states $M(W)$ as the set of all possible memory patterns of $AM(W)$, and the attraction domain $A(v)$ of a memory pattern v as the association set of v, *i.e.*, the set of all possible patterns from which v can be recalled (see [3,5]).

One goal of using a neural network as a model of associative memory is that, given a set of m *learning* patterns $v_1, ..., v_m$ $(1 \leq m \leq 2^n)$, to construct weights W such that the network has all $v_1, ..., v_m$ as memory patterns and each association set $A(v_i)$ is as "large" as possible.

One way to construct the weight matrix was proposed by Hopfield [9] and other people. That is, using the Hebb rule,

$$W = \sum_{j=1}^{m} v_j v_j^T - mI = VV^\top - mI_n$$

(where I_n is the $n \times n$ identity matrix) for learning patterns $V = [v_1, ..., v_m]$ [1]. (We will also denote $AM(V)$ for $AM(W)$, and $M(V)$ for $M(W)$, provided V and W have the above relation.)

[1]Since order of v_j's in V does not affect the formed weight matrix W, we can put V in a vector form.

A very basic question of such a Hopfield memory is about its memorizing capacity. One probabilistic capacity is the number K such that any set of less than K learning patterns will be memory patterns with probability approximately 1. The capacity K of a Hopfield memory of n neurons was shown empirically to be $0.15n$ [9] and was proved probabilistically to be of order $n/log(n)$ [10]. And an upper bound of K was proved to be $2n$ for random learning patterns [2] and to be cn^2 (for some constant c) for correlated learning patterns [6].

Based on these results, people started to make conjectures on non-probabilistical capacity m – there exists some set of m learning patterns which are memory patterns. There are two conjectures: for a Hopfield memory of n neurons, (1) it is impossible for m to be between $2n+1$ and $2^n - 1$ [7]; and (2) the maximal m such that some set of m learning patterns V are only memory patterns without introducing any *extraneous* memory patterns ($V = M(V)$) is less than cn^2 for some constant c greater than 1 [11].

In this paper, we will first study some properties of the Hopfield memory and then show that for some n, the capacity m can be between $2n+1$ and $2^n - 1$, and further number m of learning patterns as the only memory patterns without introducing any extraneous memory patterns can be of *exponent* order of n. Therefore, we disprove the above conjectures in general.

The followings are some mathematical notations and definitions used later in this paper ([1,3,4,5]).

For any $x, x' \in \mathcal{B}^n$, we will denote by $-x$ the complementary pattern of x, by $\delta(x, x')$ the Hamming distance of x and x', *i.e.*, the number of components where x and x' differ. Two patterns x and x' are orthogonal if their inner product $x^\top x' = \Sigma_i x_i x_i' = 0$. If x and x' are orthogonal, then $\delta(x, x') = n/2$. Therefore, if there are some orthogonal patterns in $\mathcal{B}^n$, then n must be even.

An *isometry* on $\mathcal{B}^n$ is a map $\tau : \mathcal{B}^n \to \mathcal{B}^n$ preserving the Hamming distance, *i.e.*, $\delta(\tau x, \tau x') = \delta(x, x')$ for any $x, x' \in \mathcal{B}^n$.

For a set of patterns $v_1, ..., v_m$, we denote V as both the matrix $V = [v_1, ..., v_m]$ and the set $V = \{v_j | j = 1, ..., m\}$. We will use I_m and 0_m to denote the $m \times m$ identity and zero matrices, respectively.

A memory pattern v is said to be a k-attractor if its neighborhood ball with radius k (in the Hamming distance) is in its association set $A(v)$. In other words, v is a k-attractor if and only if v can be recalled from any pattern x with $\delta(v, x) \leq k$.

II. Some Properties of the Model.

Lemma 1. If $B_n = [v_1, \ldots, v_{2^n}]$ is an $n \times 2^n$ matrix whose columns enumerate all vectors in $\mathcal{B}^n$, then the outer-product of B_n is

$$B_n B_n^\top = 2^n I_n.$$

Lemma 2. Suppose that learning patterns $V = [v_1, ..., v_m]$ are orthogonal. Then any patterns which can expressed as linear combinations of V are memory patterns of $AM(W)$, $W = VV^\top - mI$.

The following lemma which is used in proof of the next lemma is due to [5]. See also [3].

Lemma 3. A necessary and sufficient condition for v to be a k-attractor in a $AM(W)$ is given as follows: $\forall i, \forall j_1, ..., j_k$ (with $j_1, ..., j_k$ all different)

$$v_i \left((Wv)_i - 2\sum_{l=1}^{k} W_{ij_l} v_{j_l}\right) > 0.$$

Lemma 4. Suppose $m|n$ (n is a multiple of m), and

$$W = t\begin{bmatrix} 0_m & I_m & \dots & I_m \\ I_m & 0_m & \ddots & I_m \\ \vdots & & & \vdots \\ I_m & I_m & \dots & 0_m \end{bmatrix},$$

for some $t > 0$. Then for any pattern v, v is a memory pattern in $AM(W)$, if and only if

$$v = \begin{bmatrix} u \\ u \\ \vdots \\ u \end{bmatrix}, \quad \text{for some} \quad u \in \mathcal{B}^m.$$

Hence (i). the set of all memory patterns of $AM(W)$ is $M(V) = \{[u^\top, ..., u^\top]^\top \,|\, u \in \mathcal{B}_m\}$ of in the matrix form,

$$M(W) = \begin{bmatrix} \mathcal{B}_m \\ \mathcal{B}_m \\ \vdots \\ \mathcal{B}_m \end{bmatrix};$$

(ii). the number of memory patterns $|M(V)|$ is 2^m;

(iii). every memory pattern is an exact k-attractor where $k = \lfloor n/2m - 1/2 \rfloor$.

Theorem 1. (Isometric Preservation of k-attractors). Let τ be any isometry on $\mathcal{B}^n$. And let $V = [v_1, ..., v_m]$ are learning patterns and $V' = \tau(V) = [\tau v_1, ..., \tau v_m]$ be the image of V under the isometry τ. Consider two $AM(W)$ and $AM(W')$, where $W = VV^\top - mI$ and $W' = V'V'^\top - mI$. Then, an pattern v is a k-attractor in the $AM(W)$ if and only if $\tau(v)$ is a k-attractor in the $AM(W')$.

Corollary 1. If V and V' are two isometric sets of m learning patterns. Then the numbers of memory patterns of the $AM(V)$ and $AM(V')$ are equal, *i.e.*, $|M(V)| = |M(V')|$.

Theorem 2. If m is such that the Hadamard matrix [2] $H_m = [u_1, ..., u_m](u_j \in \mathcal{B}_m)$ exists and $m|n$, then the $AM(V)$, where

$$V = \begin{bmatrix} H_m \\ \vdots \\ H_m \end{bmatrix} = \left.\begin{bmatrix} u_1 & \cdots & u_m \\ \vdots & \dots & \vdots \\ u_1 & \cdots & u_m \end{bmatrix}\right\} n/m \quad \left(\text{hence } W = m\begin{bmatrix} 0_m & I_m & \dots & I_m \\ I_m & 0_m & \dots & I_m \\ \vdots & & & \vdots \\ I_m & I_m & \dots & 0_m \end{bmatrix}\right),$$

has 2^m memory patterns, and each memory pattern is exactly an $\lfloor n/2m - 1/2 \rfloor$-attractor.

III. Disproof of the Conjectures.

There is an inductive way to construct Hadamard matrices H_m[8,3], when $m = 2^q$ for $q = 1, 2, ...$:

$$H_2 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \qquad H_{2^q} = \begin{bmatrix} H_{2^{q-1}} & H_{2^{q-1}} \\ H_{2^{q-1}} & -H_{2^{q-1}} \end{bmatrix}$$

[2] A Hadamard matrix H_m is an orthogonal $m \times m$ matrix with entries 1 or -1. For construction and properties of H_m, see [8].

Theorem 3. Let

$$V = \begin{bmatrix} H_{2^q} \\ \vdots \\ H_{2^q} \end{bmatrix}$$

and $V' = M(V)$ be the set of all memory patterns of the $AM(V)$. Then V' are the only memory patterns in $AM(V')$ without any extranenous memory patterns, *i.e.*, $V' = M(V')$. And moreover V' are $\lfloor n/2m - 1/2 \rfloor$-attractors. [3]

Corollary 2. For some n it is possible for a Hopfield memory of n neurons to have a number of learning patterns of up to exponent order of n as the only memory patterns without introducing any extraneous memory patterns.

Proof. If we take $n = 2^{q+1}$, $m = 2^q = n/p$ for some $p > 1$ and V as in the theorem, then $M(V)$ are the $2^m = 2^{n/p}$ learning patterns which are the only memory patterns ($\lfloor p/2 - 1/2 \rfloor$-attractors) in the Hopfield memory $AM(M(V))$ without any extraneous memory patterns.

Acknowledgement. The author thanks Dr. E.K. Blum for his valuable guidance to this research.

References

[1] P. Baldi. "Group Actions and Learning for a Family of Automata". *J. Computer and System Sciences.* Vol. 36, 1988, 1-15.

[2] P. Baldi, S. Venkatesh. "Number of Stable Points for Spin Glass and Neural Networks of Higher Orders". *Phys. Rev. Lett.* Vol. 58, 1987, 913-915.

[3] E.K. Blum. "Mathematical Aspects of Out-Product Asynchronous Content-Addressable Memories". U.S.C. Sept. 1988.

[4] E.K. Blum, X. Wang. "Mathematical Properties of Out-Product Asynchronous Associative Memories". submitted to the NIPS, Nov., 1989.

[5] M. Cottrell. "Stability and Attractivity in Associative Memory Networks". *Biol. Cybern.*, Vol. 58, 1988, 129-139.

[6] E. Gardner. "Maximum Storage Capacity in Neural Networks". *Europhy. Lett.*, Vol. 4, 1987, 481-485.

[7] K. Haines, R. Hecht-Nielsen. "ABAM with Increased Information Storage Capacity". *IEEE Conf. on Neural Networks*, San Diego, Vol. II, 181-189, 1988.

[8] M. Hall, Jr. *Combinatorial Theory.* 2nd Ed. Wiley-Interscience Series in Discrete Mathematics. 1986.

[9] J.J. Hopfield. "Neural Network and Physical Systems with Emergent Collective Computational Abilities". *Proc. Nat. Acad. Sci. USA*, Vol. 79, April, 1982, Biophysics, 2554-2558.

[10] R.J. McEliece, E.C. Posner, E.R. Rodemich, S.S. Venkatesh. "The Capacity of the Hopfield Associative Memory". *IEEE Trans. Information Theory*, Vol. IT-33, No. 4, July, 1987, 461-482.

[11] P.D. Wasserman. *Neural Computing: Theory and Practice.* Van Nostrand Reinhold. 1989.

[3] Briefly the theorem says $M(M(V)) = M(V)$. Any learning patterns V of such property is called *idempotent* in [4], where a general condition for V to be idempotent is given.

Generalized Neural Network Model And Its Properties

X. Xu
W. T. Tsai

Computer Science Department
University of Minnesota
Minneapolis, MN 55455

ABSTRACT

This paper proposes a neural network which generalizes Hopfield's model. In other words, Hopfield's model is a special case of our model. Various properties of the model are studied. The generalized model is more powerful, both in theory and practice (Xu, Tsai, & Huang, 1988, Xu & Tsai, 1989). It is also easier to formulate some application problems in the proposed model than in Hopfield's model.

1. Introduction

Among many neural network models, Hopfield's model (Hopfield, 1982, 1984) is particularly attractive because it has been demonstrated that it can potentially solve computationally difficult problems such as traveling salesman problem (Hopfield & Tank, 1985). This paper introduces a new neural network model which generalizes Hopfield's model. In other words, Hopfield's model is a special case of the proposed model. Various properties of the new model is studied. It is contended that the new model is more powerful and in many cases easier to use for solving practical problems.

2. The Proposed Model

A neural network is a collection of neurons interacting with each other. The behavior of a neural network is completely determined by the specification of the interaction. By specifying different interactions among neurons, one gets different neural networks.

2.1. Interaction Level among Neurons

This section introduces the concept of interaction levels.

Let V_i denote the state of neuron i, suppose $V_i \in [0, 1]$, the interaction level specifies how v_i depends on other neurons' states:

Interaction Level p: At this level, other neurons' states in the network influence V_i in the form

$$\sum_{i_1=1}^{N} \cdots \sum_{i_p=1}^{N} t_{ii_1 \cdots i_p} V_{i_1} \cdots V_{i_p}$$

where $t_{i_1 i_2 \cdots i_p} \in R$.

Hopfield's model has interaction level 0 and 1 only, and this may cause some limitations.

2.2. The Proposed Binary Model

In this model, $V_i \in \{0, 1\}$, for i = 1, ..., N, where N is the number of neurons in the network, and neuron interactions are on level 0, 1, 2,... to $K-1$, $K \leq N$. Let $V_i(t)$ denote the state of neuron i at moment t, and each neuron updates randomly in time its state according to the following equation:

$$V_i(t+1) = \text{sgn}\,\Big(\frac{1}{(K-1)!}\sum_{i_1=1}^{N} \cdots \sum_{i_{K-1}=1}^{N} t_{ii_1 \cdots i_{K-1}} V_{i_1}(t) \cdots V_{i_{K-1}}(t) + \cdots +$$
$$\frac{1}{2}\sum_{i_1=1}^{N}\sum_{i_2=1}^{N} t_{ii_1i_2} V_{i_1}(t) V_{i_2}(t) + \sum_{i_1=1}^{N} t_{ii_1} V_{i_1}(t) + I_i \Big), \qquad i = 1, ..., N$$

where *sgn* is defined as follows:

$$\text{sgn}\,(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$

where I_i is the threshold value of neuron i. Furthermore, $t_{ii_1 \cdots i_p} = t_{\sigma(ii_1 \cdots i_p)}$, with $\sigma(ii_1 \cdots i_p)$ denoting a permutation of $ii_1 \cdots i_p$. This means that interactions on all the levels are symmetric, and all "diagonal" elements of t (t with two or more subscripts equal) are non-negative.

Like the Hopfield neural network, the above neural network has the property of converging to stable states.

Theorem 1: Let $(V_1, ..., V_N)$ denotes the state of the network, if the neural network updates its state sequentially, i.e., no more than one neuron is allowed to update its value at any time, then the neural network will converge to a stable state (dependent of the initial state).

Proof: Due to the space limitation, the detailed proof is omitted, the idea is to construct the following function:

$$E = -\frac{1}{K!}\sum_{i_1=1}^{N} \cdots \sum_{i_K=1}^{N} t_{i_1 \cdots i_K} V_{i_1} \cdots V_{i_K}$$
$$- \cdots - \frac{1}{2}\sum_{i_1=1}^{N}\sum_{i_2=1}^{N} t_{i_1i_2} V_{i_1} V_{i_2} - \sum_{i_1=1}^{N} I_{i_1} V_{i_1}$$

The function E is called the energy function of neural network or the Lyapunov function. To prove the theorem, one only has to show that the function E is non-increasing, and show that when E stablizes, so do the neurons' values (for detailed proof, see (Xu, Tsai, & Huang, 1988)).

The above theorem is a successful generalization of the one for the Hopfield neural network. But on many other aspects, properties are more difficult to establish than just applying the same methodology (from Hopfield's network). As an example, let us examine the oscillating cycle length in parallel updating mode, i.e., when all neurons always update their values at the same time.

It is well known that for Hopfield's network, the oscillating cycle length is at most 2 in parallel updating mode. We introduce an example to enlight the difference introduced by more complex interactions among the neurons.

Example 1 Consider a neural network with $n + 2$ neurons, those neurons are numbered from 1 to n, a and b, as shown in Figure 1. The double-arrow line indicates an interaction of level 1 between the neurons, and the values above them indicates the interaction strength or weights. Those arrow lines incident to the square box represent an interaction on level 2, which involves neurons 1, a and b. Also suppose that the threshold value of each neuron is as following:

$t_i = n + 1.5 - i, \qquad i = 1, ..., n$

$t_a = t_b = 0.5$

and the level 2 interaction strength

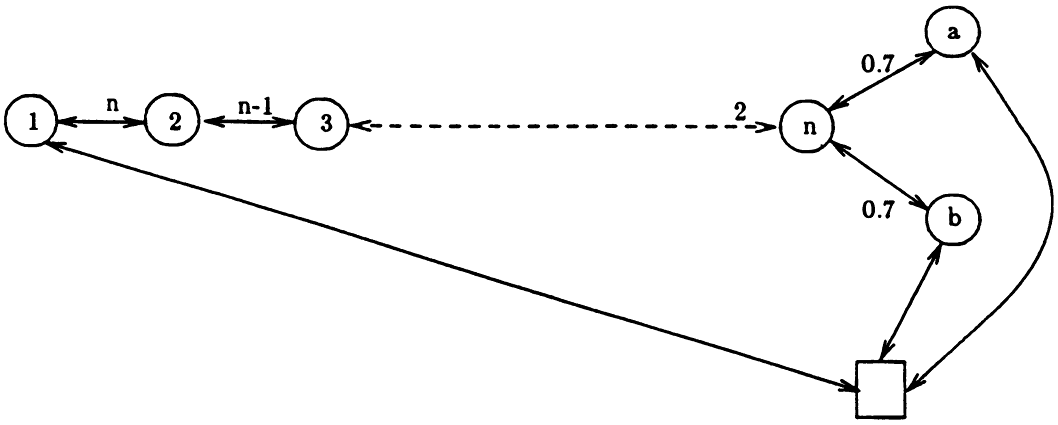


Figure 1

$t_{1ab} = n + 1$

If the neural network starts from an initial state (1, 0, ..., 0) (i.e., neuron 1 has value 1, and all other neurons having value 0, the last two bit is for neurons a and b), then by parallel updating, we will have the following oscillating cycle:

$V = (1, 0, ..., 0) \rightarrow (0, 1, ..., 0) \rightarrow ... \rightarrow (0, ..., 0, 1, 0, 0) \rightarrow \rightarrow (0, ..., 0, 1, 1) \rightarrow V.$

which is of length $n + 1$. The example used only one interaction of level 2, and it increased the cycle length to $O(n)$.

2.3. The Proposed Continuous Model

In this case, neuron state is a continuous variable instead of being binary, i.e., $V_i \in [0, 1]$. It is a kind of binary model simulated in an analog way. Each neuron's state is determined by the following differential equation:

$$\frac{du_i}{dt} = -u_i + \left(\frac{1}{(K-1)!}\sum_{i_1=1}^{N} \cdots \sum_{i_{K-1}=1}^{N} t_{ii_1\cdots i_{K-1}} V_{i_1} \cdots V_{i_{K-1}} + \right.$$

$$\left. ...+ \frac{1}{2}\sum_{i_1=1}^{N}\sum_{i_2=1}^{N} t_{ii_1i_2}V_{i_1}V_{i_2} + \sum_{i_1=1}^{N} t_{ii_1}V_{i_1} + I_i\right), \qquad i = 1, ..., N$$

with $u_i = G(V_i)$, where G is a strictly increasing function, with $G(1) = \infty$ and $G(0) = -\infty$. Other conditions are the same as in the binary model except there are no restrictions on diagonal elements of t.

Theorem 2: The above continuous neural network always converges to stable states.

Proof: Like in the binary case, we consider the following energy function:

$$E = \sum_{i_1=1}^{N} \int_0^{V_{i_1}} G(v)\, dv - \frac{1}{N!}\sum_{i_1=1}^{N} \cdots \sum_{i_N=1}^{N} t_{i_1\cdots i_N} V_{i_1} \cdots V_{i_N}$$

$$- \cdots - \frac{1}{2}\sum_{i_1=1}^{N}\sum_{i_2=1}^{N} t_{i_1i_2}V_{i_1}V_{i_2} - \sum_{i_1=1}^{N} I_{i_1}V_{i_1}$$

It can be shown that $\frac{dE}{dt} \le 0$, and when $\frac{dE}{dt} \rightarrow 0$, $\frac{du_i}{dt} \rightarrow 0$ and $\frac{dV_i}{dt} \rightarrow 0$.

3. Applications in Optimization

There are many attempts to use neural network (Hopfield's model, mostly) to solve optimization problems. The generalized neural network provides a more powerful mean. For example, the generalized neural network has been successfully used to improve the performance of Hopfield's network for Traveling Salesman Problem (Xu & Tsai, 1989). This may be the first time a neural algorithm has produced competitive results compared with the ones obtained by conventional heuristic algorithms on TSP. The new model is also easier to use to formulate some optimization problems into neural network frame.

As an illustrative example, consider the 3-Satisfiability problem, which is a well-known NP-Complete problem (Garey & Johnson, 1979):

Given a set of boolean variables $U = \{V_1, ..., V_N, \overline{V}_1, ..., \overline{V}_N\}$, and given a set C of 3-clause on U, $C = \{c_1, ..., c_M\}$, with $c_i = X_i + X_j + X_k$, where X_i, X_j and X_k are from U, the question is that if there exist an assignment of the boolean variables such that all clauses have "TRUE" value.

This problem can be stated in an equivalent version, i.e., if there exists an assignment of the boolean variables such that all clauses have "FALSE" value, where the clauses are now in the form $c_i = \overline{X}_i\overline{X}_j\overline{X}_k$.

Formulating the problem (of the second version) in the new neural network model is straightforward, let X_i denote V_i with $i \leq N$, and X_{N+i} denote $\overline{V}_i = 1 - V_i$, then the following function represents the object function to be minimized:

$$E = \sum_{(i,j,k)\in C} X_i X_j X_k$$

by $(i,j,k) \in C$ it really means that $X_iX_jX_k \in C$, note that $1 - V_i$ has been used instead of $\overline{V}_i$.

The neural network is not difficult to specify, there are exactly N neurons, each neuron corresponds to a boolean variable, so by V_i we mean either the boolean variable or the corresponding neuron, then the neuron updating function is as follows:

$$V_i\,(t+1) = \text{sgn}\,(\, - \sum_{V_iX_jX_k\in C} X_j\,(t)X_k\,(t) + \sum_{\overline{V}_iX_jX_k\in C} X_j\,(t)X_k\,(t)\,)$$

where by $V_iX_jX_k \in C$ we denotes the membership of the clauses in the set C, while by $X_j\,(t)X_k\,(t)$ we denote the states at time t of corresponding neurons.

Note that 1-Satisfiability and 2-Satisfiability are in P, but 3-Satisfiability is NP-Complete. Formulating the 3-Satisfiability problem in original Hopfield's model is very complex.

4. Reference

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability,* New York: W. H. Freeman and Co.

Hopfield, J. J. (1982). "Neural networks and physical systems with emergent collective computational abilities" *Proc. Natl. Acad. Sci. U. S. A.* 79, pp. 2554-2558.

Hopfield, J. J. (1984). "Neurons with graded response have collective computational properties like those of two-state neurons.", *Proc. Natl. Acad. Sci. U. S. A.* 81, pp. 3088-3092, May.

Hopfield, J. J., & Tank, D. W. (1985). "'Neural' computation of decisions in optimization problems", *Biological Cybernetics*, pp. 141-152, July.

Xu, X., Tsai, W. T., & Huang, N. K. (1988). "A generalized neural network model", CSci TR 88-30, Department of Computer Science, University of Minnesota.

Xu, X, & Tsai, W. T. (1989). "Effective neural algorithms for traveling salesman problem", CSci TR 89-50, Department of Computer Science, University of Minnesota.

Effects of neuron properties on the performance of associative memory networks

Hiro F. Yanai and Yasuji Sawada
Research Institute of Electrical Communication
Tohoku University, Sendai 980, JAPAN

1. Introduction

Model neural networks have been used as mathematical models of the central nervous system of biological systems, and also investigated with an aim to discover the principle of parallel information processing. For the sake of understanding the central nervous system of a biological system, i.e. brain, there are some ways we can take. One is to try to understand the brain by collecting experimental facts obtained in the field of biology or physiology. Another way is to examine it from the macroscopic viewpoint based on psychological facts. However, it is important as well to analyze model neural networks constructed from well-defined elements to grasp essential functions of the brain.

For the development of the research of neural networks, first, one should make clear the ability of homogeneous neural networks. One must find out what property of components(neurons, synapses) affects what part of ability of a neural network system. In fact, it is known that the property of neurons affects the ability of auto-associative memory networks[1),2)]. Secondly, by examining how the shortcomings of a homogeneous network are overcome in a structured network, the principle of information processing of a neural network may be clarified.

In the present paper, we mainly pay attention to the first aspect. We use discrete model neural networks with correlation-type synaptic weights. In section 2, the relation between associative recalling ability and self-connections of neurons is investigated for the network with fixed synaptic weights. In a broad sense, self-connections are considered parameters determining the response characteristics of neurons. In section 3, we see how the behavior of a neural network system is affected by the individual property of neurons to learning signals. There we discuss the robustness of a system to learning signals (environment) for unsupervised learning. The improvement of performance by the presence of adaptation mechanism is presented. This would be an attempt to accept an adaptation property as a paradigm for the behavior of neurons in unsupervised learning.

2. Relation between associative recalling ability and response property of neurons

Consider a neural network consisting of N neurons and with feedback to itself. From the state of neurons $x_i(t)$ at time t, the state at t+1 is calculated by

$$x_i(t+1) = \mathrm{sgn}[\sum_{j \neq i} w_{ij} x_j(t) + b x_i(t)], \qquad (1)$$

where $x_i(t)=-1$ or 1, w_{ij} is a synaptic weight from j-th to i-th neuron, b is a self-connection of a neuron, and

$$\mathrm{sgn}[u] = \begin{cases} -1, & u<0, \\ 1, & u\geq 0. \end{cases}$$

By constructing synaptic weights

$$w_{ij} = (1/N)\sum_m s^m{}_i s^m{}_j$$

from patterns to be memorized s^1, s^2, ..., s^M($s^m{}_i=-1$ or 1), the network behaves as an auto-associative memory. When $b>0$, neurons have hysteretic properties. Given pattern ratio $r(=M/N)$ which is a measure of loading to the network, associative recalling ability for this type of network is known to be the best for $b \approx r$[1].

If we construct synaptic weights

$$w_{ij} = (1/N)\sum_m s^{m+1}{}_i s^m{}_j,$$

the network can work as a sequence-associative memory. If each component of consecutive memory patterns is generated randomly and independently, recalling ability is the best for $b=0$. However, in general, if each component of patterns is generated by the rule $\mathrm{Prob}[s^{m+1}{}_i \neq s^m{}_i]=p$, recalling ability is shown to be the best for $b \approx -q$ by numerical experiments, where $q=1-2p$. Fig.1 shows examples of these results. From the state $\mathbf{x}(t)$ at time t and the pattern to be taken s^t at that time, direction cosine is defined by $a_t=(1/N)\sum_i s^t{}_i x_i(t)$. Fig.2 shows an experimental result to see the best value of b. There a_{20} is plotted against b starting from $a_0=1$.

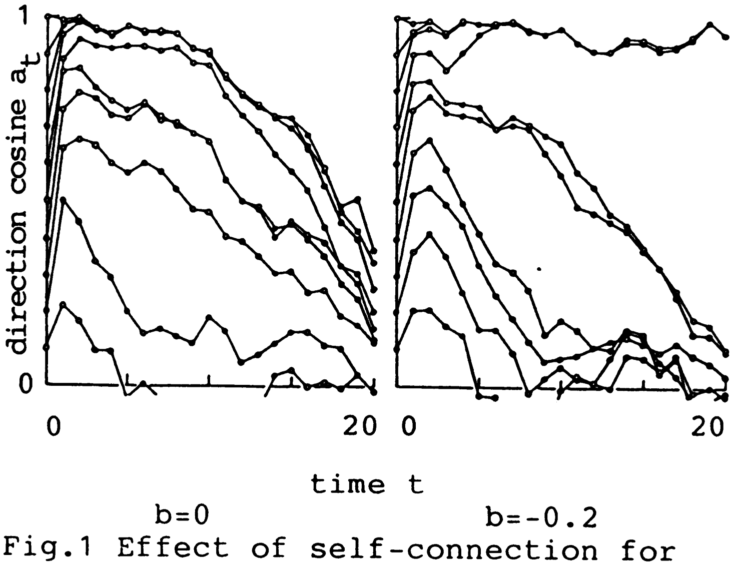


Fig.1 Effect of self-connection for sequence association. N=500, M=70, q=0.2.

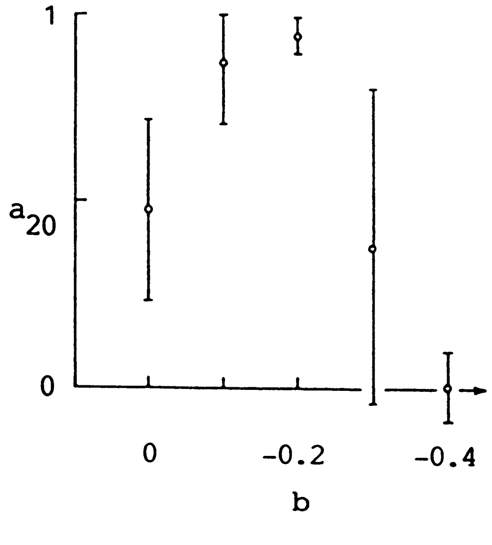


Fig.2 Effect of self-connection. N=500,M=70,q=0.2. a_0=1. Ten trials.

To interpret the experimental results for sequence association, we have derived an equation for the single-step transition of direction cosines by using the method of statistical neurodynamics[3]. When a single sequence of length M is memorized to the network, we have

$$a_{t+1} = \frac{1}{4}[(1+a_t)(1+q)F(\frac{(1+Q)a_t+b}{\sigma})+(1+a_t)(1-q)F(\frac{(1-Q)a_t-b}{\sigma})$$
$$+(1-a_t)(1+q)F(\frac{(1+Q)a_t-b}{\sigma})+(1-a_t)(1-q)F(\frac{(1-Q)a_t+b}{\sigma})],$$
$$\sigma = \sqrt{r+2q^2(r+a_t^2)/(1-q^2)}$$

where

$$Q=q(1+q^2)/(1-q^2), \quad r=M/N, \quad F(u)=\frac{1}{\sqrt{2\pi}}\int_{-u}^{u} e^{-\frac{t^2}{2}}dt.$$

When $q>0$ and $a_t=1$, for example, a_{t+1} is maximized for $b \gtrsim -q$, and the best value of b increases with r. Note that $b<0$ corresponds to refractoriness of a neuron, and for this case eq.(1) is a version of Caianiello's neuronic equation[4].

3. Learning with adaptive neurons --- robustness to learning signals

In this section, we see how the outcome of unsupervised learning is affected by the adaptation(in a similar sense as habituation) of neurons. There are some models proposed for the adaptation of a network as a system(e.g., ref.5). But our concern is the effect of adaptation of individual neurons on the system. The behavior of the network is described by

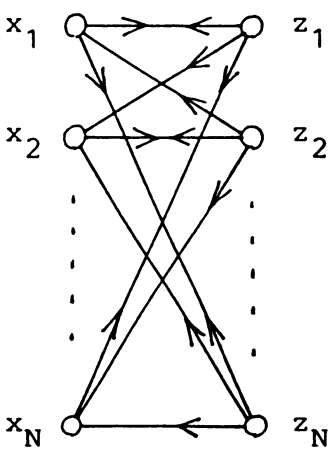
$$z_i = \mathrm{sgn}[u_i], \quad u_i = \sum_j w_{ij}x_i - h_i$$

and the learning equations with adaptation(of thresholds) are

$$\tau dw_{ij}/dt = -w_{ij} + \beta z_i x_j, \tag{2}$$

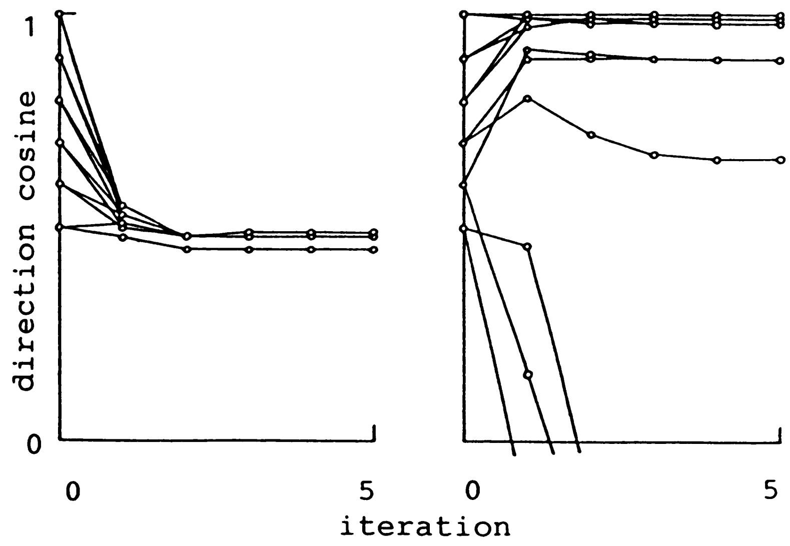
$$\tau' dh_i/dt = u_i, \tag{3}$$

where $\beta>0$, and $\tau, \tau' \gg 1$(adiabatic approximation). As an example, unsupervised learning for Bidirectional Associative Memory (BAM)[6]-type network is considered here(see Fig.3). Learning signals(patterns) are presented repeatedly to a representation layer as a vector **x**, and at the same time the synaptic weights between the two layers and thresholds h_i are modified according to eqs.(2) and (3) until they reach equilibrium. Thresholds of neurons in a representation layer are fixed at zero, and neurons in a hidden layer have adaptation property. As is well known, correlation(Hebbian)-learning network is sensitive to similarity between learning signals[7]. In the case where learning signals are such that $\mathrm{Prob}[x^m_i=-1]=p_i$, there occurs total loss of signal information for $p_i \neq 0.5$(Fig.4(a) presents this fact). If there are adaptation properties in neurons, however, the network is robust



representa-tion layer (R) hidden layer (H)

Fig.3 BAM-type network



(a) non-adaptive (b) adaptive

Fig.4 Effect of adaptation property for BAM-type network. N=200,M=10,p_i= 0.3 for all i, β=10. R→H: 20 connections per neuron, H→R: full connection

to unfavorable learning signals(Fig.4(b)). Finally, we should mention that it is essential for this network to have sparse forward(R→H) connections.

4. Summary

For correlation-type model neural networks, we have shown that associative recalling ability as a system is greatly affected by the property of individual neurons, i.e., hysteresis, refractoriness and adaptation.

References

1)Yanai,H.& Sawada,Y.: Associative memory network composed of neurons with hysteretic property, Neural Networks, to appear
2)Yanai,H.& Sawada,Y.: Integrator neurons for analogue neural networks, IEEE Trans. Circuit and Systems, to appear
3)Amari,S.& Maginu,K.: Statistical neurodynamics of associative memory, Neural Networks **1**, 63-73(1988)
4)Caianiello,E.R.: Outline of a theory of thought-processes and thinking machines, J.Theoret.Biol. **2**, 204-235(1961)
5)Kohonen,T.& Oja,E.: Fast adaptive formation of orthogonalizing filters and associative memory in recurrent networks of neuron-like elements, Biol.Cybern. **21**, 85-95(1976)
6)Kosko,B.: Bidirectional associative memories, IEEE Trans. Systems,Man and Cybern. **18**, 49-60(1988)
7)Amari,S.: Neural theory of association and concept-formation, Biol.Cybern. **26**, 175-185(1977)

Neural Networks for Maximum Likelihood Error Correcting Systems

Jar-Ferr Yang, Chi-Ming Chen, and Jau-Yien Lee
Department of Electrical Engineering
National Cheng Kung University
Tainan, Taiwan, R. O. C.

Abstract

In this paper, the neural networks for the soft-decision error correction systems based on the solution of maximum likelihood are presented. The match filtering structure of the combined detections and error correcting neural networks is suggested to further improve the correcting performance. Considering imperfect functions of demodulator, equalizer, and synchronizer, two simple learning algorithms, the least mean square error (LMS) and the waveform reshaped averaging (WRA) methods, are employed to adjust neural weights for achieving robust error correcting systems. Simulations of the proposed coding neural networks comparing to hard-decision error correction system are presented.*

I. Introduction

Conventional error correcting systems build on digital logic design have been employed for improving the reliable performance of digital data transmission and storage systems. For the limitation of logic design, the optimal bit detection has to be completed and sampled prior to the error correction systems. This is so-called hard-decision coding [1]. In order to improve the correction performance, the soft-decision concept has been introduced by increasing the quantization levels of the signal set to achieve a M-ary coding techniques at the expense of highly increased system cost and complexity [1,2]. Furthermore, this conventional logic error correcting system perform adequately only under the assumption that the supporting systems and itself are in good conditions. There are however instances such as imperfect functions of demodulator, equalizer, and synchronizer or deficient components of coding circuit systems, etc. where such an assumption is not valid. In this case, the performance of error correction systems decreases dramatically or fails to function properly.

Artificial neural systems technology recently has earned a lot of attention for many researches and applications since Hopfield and Tank's publishes [3,4]. Neural networks have been recognized as a greatest potential technology in areas where many hypotheses are pursued in parallel and high computation rates are required. Error correcting is one of the techniques which need highly parallel processing at high speeds for critical tasks as memory read checking. At the same time, the native learning behaviors of neural networks can adapt themselves to operate acceptably even in the conditions of imperfect functions. The Hamming nets [5] and their realized neural circuits [6] ebulliently initiate the neural system technology to the applications of error correction coding.

In this paper, we propose an error correcting neural network for soft-decision error corrections system which is based on the solution of maximum likelihood (ML) in Section II. Furthermore, the optimal detection neural network based on the concept of match filtering of coded waveforms is employed to further improve the performance of conventional coding in Section III. In order to restore the degradation of neural systems operating in ill-behaved conditions, two simple learning algorithms, least mean square error (LMS) and the waveform reshaped averaging (WRA) methods are suggested to adjust neural weights for achieving robust error correcting systems in Section IV. Simulations of the proposed coding neural networks comparing to hard-decision error correction systems are presented in Section V.

II. Maximum Likelihood Error Correction Coding Neural Networks

Consider the case in which the unquantized output of the demodulator is fed to the (n,k) code decoder. Thus, each binary waveform is demodulated by the optimum demodulator by the optimum

* This Research was supported by Taiwan International Standard Electronic Ltd., Taiwan, Republic of China.

demodulator (a match filter followed by a sampler) and a codeword is represented by a sequence of n random variables. Let S_{ij} denote the binary waveform of the i-th bit of the j-th codeword. Each binary decision variable can be written as

$$(1) \qquad Z_i = S_{ij} + N_{ij}$$

and i=1,2,...,n and j=1,2,...2^k . The variables $\{N_{ij}\}$ are the samples of the additive Gaussian noise with zero mean and variance (noise power) N_o /2. Thus, the density function of random variable Z_i is

$$(2) \qquad P_n(Z_i \mid S_{ij}) = 1/(\pi N_o)^{\frac{1}{2}} \exp\left\{-(Z_{ij} - S_{ij})^2/N_o\right\}$$

Once a codeword random variables Z_i, i=1,2,...,n is received without the knowledge of j of transmitted codeword. The decoding process consists of choosing one out of the 2^k possible transmitted sequences. Under the assumption of memoryless channel which N_{ij} are independent for all i and j. Likelihood function of j-th codeword is then given by

$$(3) \qquad L_j' = \prod_{i=1}^{n} P_n(Z_i \mid S_{ij}) = 1/(\pi N_o)^{\frac{1}{2}} \exp\left\{-\sum_{i=1}^{n} (Z_i - S_{ij})^2/N_o\right\}$$

For simplicity, the log-likelihood function instead of likelihood function in Eq. (3) is employed for the decision of possible codeword with maximum value among all codewords. Excluding the irrelevant terms in Eq.(3) for decision making, the optimal decoding criterion based on the solution of maximum likelihood becomes that the codeword corresponding to the maximum value among

$$(4) \qquad L_j = \sum_{i=1}^{n} Z_i S_{ij} - S_{ij}^2/2$$

j=1,2,...,2^k is selected [2]. It has been shown that this soft-decision approach presents higher reliable than that achieved by hard-decision error correcting scheme [2]. That is, the decoder can take advantage of the additional information contained in the unquantized samples that represent each individual binary transmitted waveform.

Figure 1 shows the soft-decision neural network consisting of a decoding (D-) net, a likelihood (L-) net, and a maximum picking (M-) net. The L-net which calculates the likelihood functions in Eq.(4) is composed of 2^k neurons and n input nodes. The offset ϑ_j of the j-th neuron is equal to the sum of $S_{ij}2/2$, i=1,2,...,n. The weights $W_{ij} = S_{ij}$ connect the i-th input node to the j-th neuron for i=1,2,...2^k and j=1,2,...,n to calculate the log-likelihood function L_j in Eq.(4). The M-net which was discussed in [1] selects the maximum output neuron from the 2^k possible outputs of the L-net. The D-net decodes the maximum output of M-net into the corresponding k-bit message codeword for the completion of the error correcting.

For the antipodal signals set, $S_{ij} = +\sqrt{E}$ if the i-th digit of the j-th codeword is 1 and $S_{ij} = -\sqrt{E}$ if the digit is 0. The all offsets become constant ϑ_j = n/2 which can be ignored. Then the likelihood function in Eq. (4) can be further simplified as

$$(5) \qquad L_j = \sum_{i=1}^{n} Z_i \operatorname{sign}(S_{ij})$$

Thus the weights in the L-net are W_{ij} = sign(S_{ij}) = +1 if the ith digit of the j-th codeword is 1, and W_{ij} = sign (S_{ij}) = −1 if the digit is 0. In this case, this neural network is identical to Hamming nets and circuits suggested in [5,6].

III. Match Filtering Error Correcting System

Soft-decision error correction system can improve the performance of hard-decision error correction system. However, both approaches need an optimal bit-demodulator which is a bit-waveform match filter followed by a bit-sampler. The difference between these two approaches is that the former uses the unquantized value of the sampler but the latter uses the quantized one. It is intuitive that the correction performance will be further improved if the whole waveform of the corresponding codeword are employed instead of sampled by a bit-sampler. This received waveform signal of the j-th codeword is given by

$$(6) \qquad Z(t) = S_j(t) + N(t), \quad t_o < t < t_o + nT$$

where T is the duration of digit symbol. The likelihood function [2] of the j-th codeword after introducing any orthonormal signal set followed by the similar derivations in Section II is finally given by

$$(7) \qquad L_j = \int_{t_o}^{t_o+nT} Z(t)S_j(t)\,dt - 1/2\int_{?}^{?} S_j^2(t)\,dt$$

Thus, the likelihood function is equivalent to match filtering which is the optimal detection of the codeword waveform. The implementation of match filter can be approximated by

$$(8) \qquad \hat{L}_j = \sum_{i=0}^{n-1}\sum_{q=1}^{m} Z(i,q)\,S_j(i,q) - S_j^2(i,q)/2$$

where Z(i,q) and $S_j(i,q)$ are the signal of Z(t) and S(t) sampled at $t_0 + iT + q(T/m)$. When a waveform antipodal signal set is sampled once (m=1), Equation (8) is equivalent to Eq. (4). This case can be seen in baseband digital transmissions which their equalizations as well as their bit and frame synchronizations are assumed perfect.

The match filter error correction neural network which has the same configuration as shown in Figure 1 is composed of nm input nodes and 2k neurons in the L-net. The j-th neuron output is

$$(9) \qquad \Gamma_j = -\theta_j + \sum_{i=0}^{n-1}\sum_{q=1}^{m} W_{iq,j}\,Z_{iq}$$

The optimal weights $W_{iq,j} = S_j(i,q)$, i=1,2,...,n and q=1,2,...,m, connect the signal Z_{iq} = Z(i,q) of the (i,q)-th input node to the j-th neuron with offset

$$(10) \qquad \theta_j = \sum_{i=0}^{n-1}\sum_{q=1}^{m} S_j^2(i,q)/2$$

For the general antipodal signal set, these offsets ϑ_j, j=1,2,...2^k become constant which can be ignored.

IV. Learning Algorithms for Robust Coding Neural Networks

Although the aforementioned error correction neural networks have better correcting performance than traditional logic systems, these neural systems with learning capability as well as logic systems perform adequately only under the assumption that there are no imperfect functions of demodulator, equalizer, and synchronizer and no deficient components in the decoding systems. In this section, the native learning behaviors of neural networks were further utilized to adjust the weights of the L-net for enhancing the neural correcting system operated acceptably even in case of imperfectness.

First, the least mean squared error (LMS) criterion [7] is used as a learning rule of neural systems. During the learning period, the j-th transmitted codeword is assumed known for neural systems, the desired output of the j-th neuron is obtained by substituting Z(i,q) = Sj(i,q) into Eq. (8) as

$$(11) \qquad L_{j,opt} = \sum_{i=0}^{n-1} \sum_{q=1}^{m} S_j^2(i,q)/2$$

The mean square error of the output of the j-th neuron is defined as

$$(12) \qquad \Phi_j = E\left[|e_j|^2\right] = E\left[|\Gamma_j - L_{j,opt}|^2\right]$$

where $e_j = \Gamma_j - L_{j,opt}$ is the error between the desired and the actual outputs of the j-th neuron. The LMS learning algorithm [8] can be simplified in terms the current error e_j and data Z_{iq} as

$$(13) \qquad W_{iq,j}(p+1) = W_{iq,j}(p) - 2\mu_j e_j Z_{iq}(p)$$

where $\mu_j > 0$ is the convergence factor and index p denotes the p-th iteration step of the learning process. For assuring the convergence of the LMS learning algorithm, the bound [8] of convergence factor has to be chosen in

$$(14) \qquad 0 < \mu_j < 1/L_{j,opt}$$

In Section IV we have learned that the optimal weights have to be the transmitted waveforms themselves. If the weights are updated by adding the received signal waveforms, then they will gradually converge to the received waveforms which is the optimal ones. So, the waveform reshaped averaging (WRA) method is suggested as a learning algorithm for neural systems. The WRA learning rule is given by

$$(15) \qquad W_{iq,j}(p+1) = (1 - \alpha_j)\, W_{iq,j}(p) + \alpha_j Z_{iq}(p)$$

where α_j is an averaging (or forgetting) factor. For truly arithmetic average, the non-constant factor $\alpha_j = 1/p$, p=1,2,... can be used [9].

V. SIMULATIONS

For the purpose of verifying the aforementioned error correcting neural networks, the simulation results of hard-decision logic system, soft-decision neural network, and match filtering neural network with and without learning are presented in this section. Without loss of generality, the (7,4) Hamming code and antipodal signal sets are employed in the following simulations.

After 80000-codeword runs at various signal to noise ratios (SNR), Figure 2 shows the codeword error probabilities of hard-decision, soft-decision, match filtering for signals with square antipodal and raised cosine waveforms. These results are simulated in the cases of perfect equalization and synchronization. The soft-decision neural correcting system is better than the hard-decision logic system. The match filtering neural correcting systems (with m=9) outperform both hard- and soft-decision systems which are the bit-detection and bit-sampled of the waveformed signal. The performance of square waveform match filtering neural correcting system which needs the infinite bandwidth of transmission media is better than that of the raised cosine match filtering one.

For the raised cosine match filtering neural system, Table 1 shows the numbers of error codewords after 80000-codeword runs at SNR=0db in the conditions for: (a) perfect synchronization; (b) 2T/9-time-delay unsynchronized (unsync) without learning ability; (c) unsync with 10-iteration LMS (μ = 0.005) learning; (d) unsync with 10-iteration WRA (α = 0.1) learning. The results show that the imperfect

(2T/9 delay) synchronization deduces the correcting performances dramatically. However, both learning algorithms mostly recovered the ill-behaved synchronization. The WRA learning rule has better improvement of performance than that of the LMS learning rule for compensating unsynchronized problems.

Algorithms	perfect sync	2T/9 delay unsync	unsync with LMS learning	unsync with WRA learning
Codeword error rate	0.0002	0.5034	0.0354	0.0069

Table 1. The codeword error rate of neural error correction networks

V. CONCLUSION

In this paper, the maximum likelihood error correction neural network systems are presented. Two simple learning algorithms, least mean squared error (LMS) and the waveform reshaped averaging (WRA) methods, are also suggested to adjust the proposed neural weights to acquiring robust correcting capabilities. In other words, these neural correcting systems with learning capabilities not only increase the correcting performance but also elaborate the applicability for the whole digital data transmission systems. The WRA learning algorithm has better correction performance than the LMS in imperfectness of synchronization. However, the LMS learning algorithm has its own potential for imperfect equalization error correction system which will be discussed in the future publication by the authors.

REFERENCES

[1] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1983.

[2] S. Senedetto, E. Biglieri, and V. Castellani, *Digital Transmission Theory*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1988.

[3] J. J. Hopfield and D. W. Tank, "Neural Computation of Decision in Optimization Problems," *Biological Cybernetics*, Vol. 52, pp. 141-152, 1985.

[4] J. J. Hopfield and D. W. Tank, "Computing with Neural Circuits: A Model," *The Science*, Vol. 233, 1986.

[5] R. P. Lippmann, "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, April, pp.4-22, 1987.

[6] Y. Takefuji, P. Hollis, Y. P. Foo, and Y. B. Cho, "Error Correcting System Based on Neural Circuits," *Proceedings of International Neural Symposium*, 1987.

[7] B. Widrow, J. M. McCool, M. G. Larmimore, and C. R. Johnson, Jr., "Stationary and Nonstationary Learning Characteristics of the LMS Adaptive Filter," *IEEE Proceedings*, Vol. 64, pp. 1151-1162, 1976.

[8] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*, Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1985.

[9] K. C. Sharman, "Adaptive Algorithms for Estimating the Complete Covariance Eigenstructure", *Proceedings IEEE Int. Conf. Acoust., Speech, Signal Processing*, 1986.

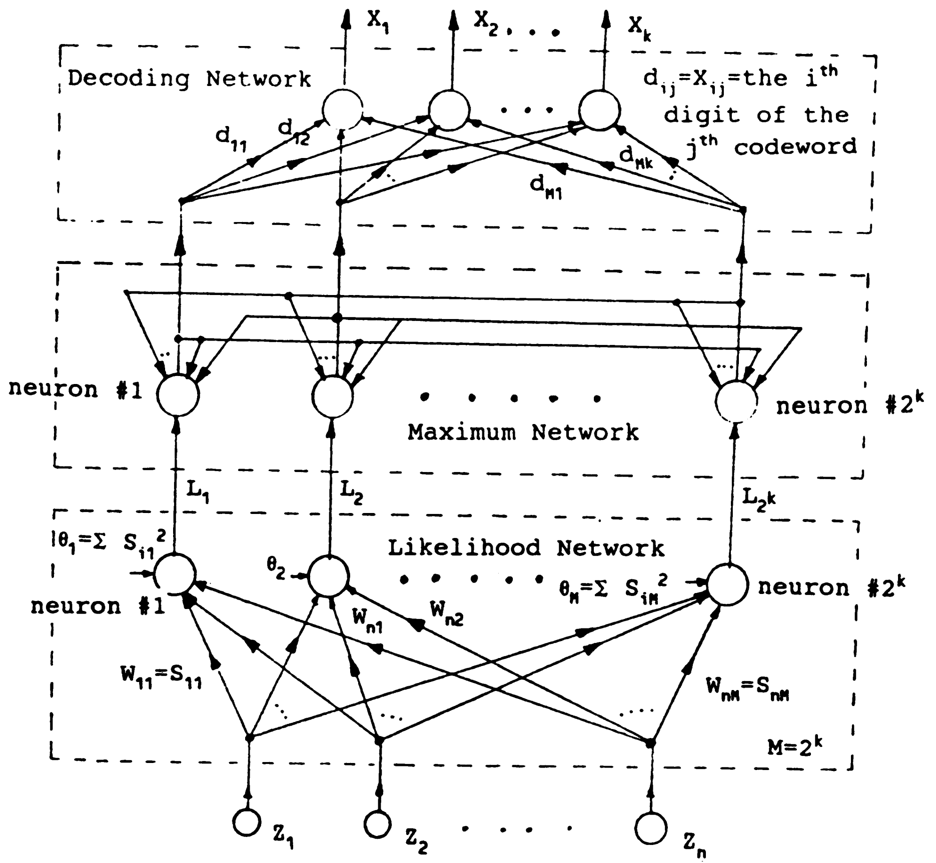


Figure 1. Maximum likelihood error correction neural system

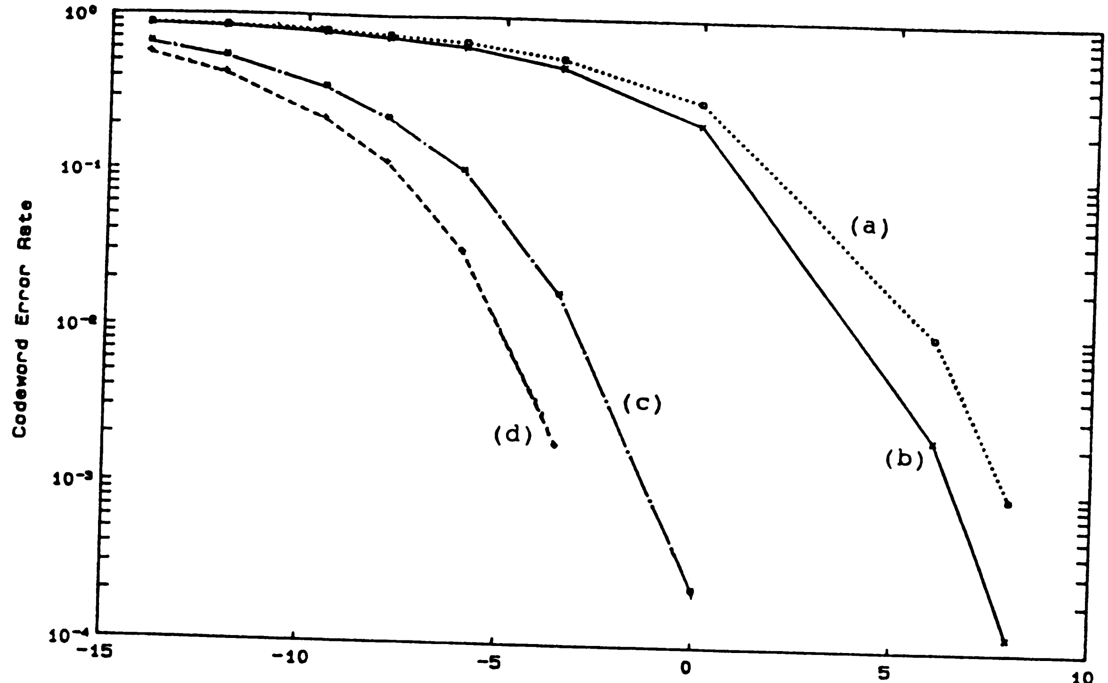


Figure 2. Simulated performances of error correcting systems: (a) hard-decision logic; (b) soft-decision neural network; (c) raised-cosine and (d) square-waveform matched filtering neural systems.

A New Kind of Associative Memory Network Model

Hong Feng Yin Ju Wei Tai

Institute of Automation, Academia Sinica

P. O. BOX 2728, Beijing, China

ABSTRACT -- *A new kind of associative memory network model which is more similar to human associative memory compared with Hopfield net and other model is proposed in this paper. The learning algorithm of the model is based on Perceptron. The nodes in the network connect with each other, but the connections are not symmetry. Under some conditions, the number of samples which can be stored in a net is given theoretically, and samples to be memorized can become the stable attractors of the nonlinear dynamical neural system. For the purpose that the net has ability to escape from a non-sample attractor to attain a sample attractor when the system operates dynamically, a deepening impression learning algorithm is given. In order to achieve the ability that arbitrary number of samples can be stored in a net, an augmented node method is also provided. In fact, the linear inseparable problems in pattern classification can be solved by the method.*

1. Introduction

It is very important to investigate the principal of associative memory. It is well known that ability of associative memory characterizes the feature of human brain. People can recall clearly the things that have been forgotten. In recent years, artificial neural network is provided as a satisfactory tool for simulating associative memory. From a distorted image or a part of image, a clear and complete image can be obtained in the network by associative principle. The Hopfield net is one of the neural network models which has capacity of simulating associative memory[1] [2]. But there are some limitations of the given models. A new associative memory model as well as the learning algorithm is proposed in this paper. The shortcomings existed in other neural networks can be overcome by this approach. And the theoretical proofs are also given.

2. The principle of associative memory

Hopfield net has two major limitations when used as an associative memory. First, the number of samples that can be stored and accurately recalled is limited. If a large amounts of samples are stored, many non-sample attractors will be produced, but samples may not be the attractors of the network. Hopfield showed the number of samples(M) can be stored in the net is less then 0.15 times of the nodes number(N) in the net when the samples are generated randomly. Another limitation of Hopfield net is that it can not distinguish sample from many similar versions. When a sample is considered as a initial state of the net, the result of the association may be not itself but another similar sample.

The new neural network model proposed in this paper is more similar to the procedure of human associative memory. The model is also a net that nodes connect each other, but the connections are not symmetry. That is, there are two weights between two nodes i and j, w_{ij} is the weight which send information from i to j, w_{ji} is the weight through which i receive information from j, two are different. The behavior of such net is also similar to Hopfield net and is described by a nonlinear dynamical system. For an arbitrary initial state, when the time is long enough, the states which can be attained in system can be attained are called attractors of the dynamical system. If an attractor is a sample, we call it a sample attractor, otherwise, it is called non-sample attractor.

IF a pattern $X(0)$ iS a initial state of the system, after k steps, the state of the system is $X(k)$, then the evolution equation is

$$x_j(k+1) = f\left(\sum_{i \neq j} w_{ij} x_i(k)\right). \tag{1}$$

Where $J(x)$ is a step function

$$J(x)=\begin{cases}+1 & x\geqslant 0\\ -1 & x<0\end{cases}$$

Finally, the net move to an attractor, if the attractor is a sample attractor, we consider that the sample is associated by the pattern $X(0)$.

The procedure of neural net memory is completed by the following Perceptron algorithm[3]. The components of the sample vectors may be two values +1 or −1. The samples are arranged circularly.

$$X(0),\ X(1),\ \ldots,\ X(t),\ \ldots$$

where $X(Mn+k)$ is same as the kth sample vector. For $X(t)$, the weights correcting algorithm is

$$w_{ij}(t+1)=w_{ij}(t)+\eta\left(x_j(t)-J\left(\sum_{k\neq j}w_{kj}(t)x_k(t)\right)\right)x_i(t) \qquad (2)$$

When all weights do not change for all samples, it is considered that the learning process is end. About such neural net model, there is theorem given following:

Theorem 1: For a given M samples, and each sample is composed of N components. If the vectors constructed by any $N-1$ components of M samples are linearly independent in $N-1$ dimensional Euclidean space, and $M<N$, then the learning process be above method is certainly convergent,and all samples can become the stable attractors of the dynamical net system. That is, if a set of samples are well distributed in the vector space and the number of samples is less then the number of the net nodes, the net can memorize all of them.

proof: (omitted)

Theorem 1 indicates that the number of samples that can be stored in a net is almost as same as the number of net nodes. If the samples are well distributed in the space, the conditions in theorem 1 can be satisfied easily. At this point, all samples can be memorized by the nets. Comparing with Hopfield net, although such a net with same nodes is two times large as Hopfield net, the number of samples that can be stored in such net is three times larger then Hopfield net. It can also distinguish very similar patterns. Some problems existed in Hopfield net can be solved using in net. The learning process is very similar to some human behaviors to memorize some pieces of information through reciting over and over again.

3. A method for enhancement of associative

Experiment results explicated that, when large amounts of samples are stored in the net, many non−sample attractors are also produced. As input patterns have some errors with samples stored in net, the associative results are often non−sample attractors, which make recall failure. A method to solve such problems is proposed.

This method is consistent with such psychological phenomenon of deepening impression. After some samples have been remembered, people often repeat again more times. So the impression of the samples is deepened. From the view of neural net system, in this time, there are some difference between the sample attractor and non−sample attractor. So through the procedure of recalling, it can be tried to escape from non−sample attractors and to attain the sample attractors. Where the algorithms are corrected as follows:

The learning algorithm:

When $x_j(t)=J\left(\sum_{k\neq j}w_{kj}(t)\right)$ *and* $\left|\sum_{k\neq j}w_{kj}(t)\right|>T$

$$w_{ij}(t+1)=w_{ij}(t)\quad .$$

Others, correcting weights

$$w_{ij}(t+1)=w_{ij}(t)+\eta x_j(t)x_i(t)$$

Where $T>0$ is an impression threshold. The above learning algorithm implicates that the samples are not only the attractors of the net. For a threshold T, when the net attains a sample attractor, all values of the nodes received is also larger then T. But for non−sample attractors, such conditions are often not satisfied. Based on this fact, the recall can escape from non−sample attractors. A motion equation is given in following:

The initial input pattern is $X(0)$, after k step, the state of system is $X(k)$, let

$$s_j=\sum_{i\neq j}w_{ij}x_i(K)$$

Then, $x_i(k+1) = +1$, *When* $s_i \geqslant T$; $x_i(k+1) = -1$, *when* $s_i < -T$.

when, $-T < s_i < T$

$$x_i(k+1) = \begin{cases} +1 & \textit{with probability} \quad (T+s_i)/2T \\ -1 & \textit{with probability} \quad (T-s_i)/2T \end{cases}$$

So when the net attains a sample attractor, it no longer move. While attaining a non−sample attractor, it continues its motion. Such a method is similar to Simulated Annealing method (SA). When a random disturbance is happened, the states of the net can escape from unexpected attractors. The method also differs from SA in some aspects. First, the aimed state of SA is only one, but this method may have many aimed states. When attaining a aimed state, the net no longer moves, but SA continue moving. The case if a state is the aimed state can not be decided by SA itself. Other methods are also needed to decide whether or not the associative results are the samples in Hopfield net, itneed not in this net.

4. An augmented node method for memory of arbitrary number samples

Theorem 1 explicates that the number of samples that net can be stored in a net is limited by the number of net nodes. The number of net nodes is as same as the number of sample components. So when the sample components are few, the neural net can only store few samples. This is a very large limitation in solving many real problems.

To solve such problems, an augmented node method is proposed. One node is augmented for one sample. For sample vectors X_1, X_2, ..., X_M, the augmented sample vectors are

$X'_1 = (X_1,\ +1,\ -1,\ ...,\ -1,\ +1\ )$

$X'_2 = (X_2,\ -1,\ +1,\ ...,\ -1,\ +1\ )$

......

$X'_M = (X_M,\ -1,\ -1,\ ...,\ +1,\ +1\ )$

Besides, the original nodes of the net not only connect with each other but also connect with all of augmented nodes. An augmented node also connects with all original nodes, but does not connect with other augmented nodes. A threshold node is also added. The state of the threshold node has value +1 and does not change. The behavior of the net can be described as, when the net move to a sample attractor, only one augmented node is excited, the other augmented nodes are inhibited. The net has $N + M + 1$ nodes. For this net, we have following theorem.

Theorem 2: If a neural net is constructed by above method, then the learning process according to (2) is convergent.

proof: (omitted)

The augmented node of a sample has obvious psychological meaning, it represents the concept of the sample. When a sample inputs to the net, its augmented node is excited. When a concept node is excited, the sample represented by the node can be associated. The original nodes and augmented nodes represent different information respectively in two layers.

When the augmented nodes connect with each other, another associative memory is formed in concept layer. For example, characters are the concepts of the images, words are the concepts of the characters. Sentences can be considered as a kind of concepts of words. In this way, information represented in different layer can be connected closely.

Certainly, this method can also solve the linear inseparable problems in pattern classification. Such problems are difficult problems, which had inhibited the development of the neural net research[4]. XOR problems can not be solved directly by Perceptron.

There are no connections between two input nodes in solving classification problems. The net has three layers: input node layer, augmented node layer and output node layer.

When learning, the net first learn the weights between the input layer and augmented layer, and then decides the weights which connect with output nodes. For an input pattern, the information is propagated to augmented layer at first, then output nodes produce the output values according to all the receiving information.

5. The simulation results

Here the computer simulation results are showed. 10 two valued images to be memorized are showed in figure 1. Each image has 10 × 10 components. By Hopfield net to store them, when the 10 samples are used as initial inputs, it can be seen that only "8" and "9" attain themselves and become the stable attractors of the net. The associative results of other samples are "8" and "9" sample or other non-sample attractors. That are showed in figure 2.

Using the nets proposed in this paper to store the same samples, the learning algorithm is convergent. That is, all samples can be remembered in the nets. When samples are used as initial inputs, all can move to themselves in one step. Using the samples with noise as initial inputs, they also associate to their original samples as showed in figure 3. If the initial inputs are arbitrary patterns as showed in figure 4., the associative results may be non-sample attractors. Some times they may be limited cycles.

By using deeping impression net to store the samples, in which the input pattern is the same as above, it can be seen that all associative results are the samples as showed in figure 5.

36 samples are also stored in a net.

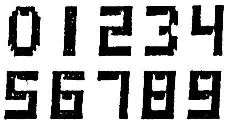


Fig 1.

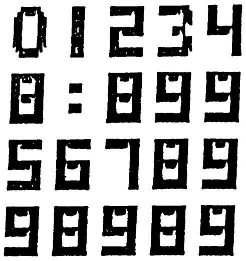


Fig 2.

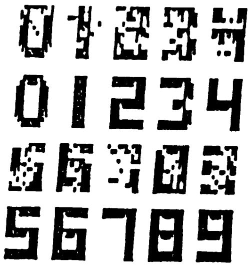


Fig 3.

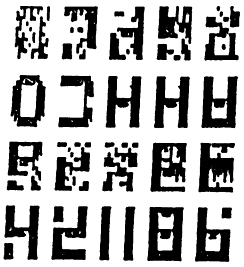


Fig 4.

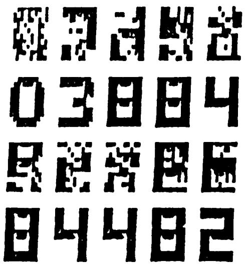


Fig 5.

6. Discussion

In this paper we proposed an associative memory model and several methods, the model has some promising properties of Hopfield net, Perceptron, Back-propagation and Simulated Annealing etc., and overcomes some shortcomings of them. The number of the samples can be stored in a net is proven theoretically. The satisfactory simulating results are also gotton. The model is similar to human associative memory and thinking way in some aspects. The method for enhancement of associative memory can make net escape from non-sample attractors and attain sample attractors. A net can store arbitrary number samples when the augmented node method are used. The method has also solved the linear inseparable problem in pattern classification. The model and methods enhance the ability of neural nets so that they are able to solve more real problems. In fact, memory is the base of intelligence. So the research on associative memory is also an important aspect of exploring intelligent behavior.

REFERENCES

[1] J. J. Hopfield, "Neurons with Graded Response Collective Computational Properties Like These Two-State Neurons", Proc. Natl Acad. USA, May 1984.

[2]J. J. Hopfield and D. W.Tank, "Computing with Neural Circuits: A Model" Science, 233 ,1986.

[3] F. Rosenblatt, "The Perceptron: A Perceiving and Recognizaing Automaton ", Project PARA, Cornell Acronaut. Lab Rept. 85-460-1, 1957.

[4] M. Minsky and S. Parpert, PERCEPTRONS. Cambridge, MA, MIT Press, 1969.

Neural Network for Image Representation using Back Propagation

Tatsuhiro Yonekura   Shigeki Yokoi   Jun-ichiro Toriwaki

Faculty of Information Engineering,   Engineering Department
Nagoya University   Furocho,Chikusa-ku,Aichi, 464,Japan

Abstract

This paper presents a method of representing image data in terms of a basis pattern set called "primitives" which are B.P.learned by neural networks. The main feature of this network lies on the way of providing the sample pairs (sample input and associated supervisory data),that is, inputting the pulse signal to a single unit of the input layer, to associate with the supervisory image pattern to appeare in the output layer. In this way, the several image data can be compressively stored in connection matrices. The coding scheme of representing pattens in connection matrices is theoretically analyzed . Although there have been already a few network models proposed for the same purpose (image data compression and 2-D signal transform by using non-orthogonal function sets) by G.W.Cottrell et,al and J.G.Daugman respectively, we discuss in the proceedings ,the advantages of our present model regarding the feature extraction as well as the learning speed of images.

1. Introduction

The neural-net proposed for 2-D signal data compression by G.W.Cottrell et.al[1)], shown in Fig.1, is referred as an auto-association (or self-association) network because of its self-mapping scheme of input/output responses. The property of this network is (1)a capability of 2-D signal encoding is obtained in the connection between the input layer and the hidden layer, whereas that of decoding is obtained in the connection between the hidden layer and the output layer,therefore,(2).being able to be used as a compressive transmission system of image data.

Issues on them are, however, (1)not being good in data compression rate as that of the singular value decomposition[3)], (2)not able to be used for feature extractor because the network does not keep the image data in it and so, the internal representation of the network hardly gives a description of the images.

Thus, our goal is to establish a computational model which analyzes and compresses the 2-D signal data by representing it in terms of some suitable elementary patterns which the network itself may organize internally.

Now, the present paper,

1). introduces a method of transformation of pattern signal into the form of network's connection matrices using multi-layered network,

2). shows that a sequence of images are expanded in terms of a basis pattern set which is organized by the network itself, and finally

3). demonstrates some experimental results indicating the feature extraction capability of the network.

2. Pulse Pattern Network

The general architecture of our neural network for signal transform is shown in Fig.2.

We may have now three layered neural network, with M units on the input layer, N units on the hidden layer and [X x Y] units on the output layer.

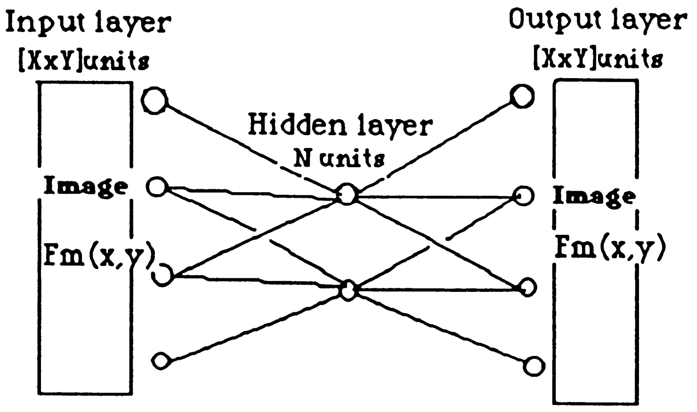


Fig.1 Auto-association network for data compression
M images are used for both input and supervisory signal to the neural net.

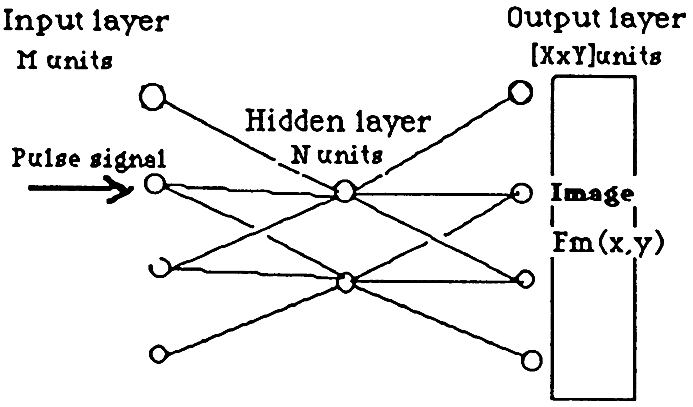


Fig.2 Pulse Pattern Network for image transform
A certain input unit is stimulated when a corresponding image is shown as a supervisory signal.

Let us deal with a set of M images : Fm(x,y), where m=1,2,...M; to be used as a *supervisory signal* , each of which is associated with a "pulse" signal(stimulation at maximum level on a certain unit) of input layer, so that each unit of the input layer can be fired to recollect a corresponding image. That is, the Pulse-Pattern Network (PPN) is defined as a multi layered network whose training sample pairs (input and supervisory data) are given as follows;
INPUT : a pulse signal (a single unit stimulated at maximum magnitude)
SUPERVISORY : an image to be associated with the unit in the input layer.
In order to learn the pulse-versus-pattern correspondence, back propagation algorithm[4] is applied.
We may be able to associate each input unit with any image in the image set by using this scheme. Let us give a brief explanation on this mechanism.

3. Theoretical Background

Now we shall discuss some properties of the network by analyzing the internal representation of the three layered PPN. Our concerns are
1). how is a set of images represented in the network?
2). and what does PPN have to do with an auto-association network?
Provided that we have a set of M images ($F_m(x,y)$: m=1,..M; x=1,..X; y=1,..Y), each of which has the size of [XxY] . Also we have a network with M input units, N hidden units, and [XxY] output units (see Fig.2). Moreover, let $H_n(x,y,t)$ (where, n=

0,1..N; x=1,...,X; y=1,...,Y) denote the output pattern appeared when only the 'n'th hidden unit is excited (assume that all other hidden units are inhibited) and especially H_0 denote the pattern when no hidden unit excited, at the time when t steps of back propagation training are done. Since the i/o response of each unit employs *sigmoidal function* ,

$$f(x) = \frac{1}{1+\exp(-x)} \cdots (1)$$

By the definition, H0,..Hn are expressed as a function of the connection weight Wn(x,y), between the hidden unit no.'n' and the unit on the grid(x,y) of the output layer ,with the exception of W0(x,y) being the threshold of the unit on the grid(x,y).

$$H0(x,y,t) = \frac{1}{1+\exp(W0(x,y))} \cdots (2)$$

$$Hn(x,y,t) = \frac{1}{1+\exp(W0(x,y)-Wn(x,y))} \cdots (3)$$

In more general case where each unit of the hidden layer outputs the value An, where $0 \leq An \leq 1$, the pattern appeared on the output layer C(x,y,t) is also written in terms of Wn(x,y) as follows,

$$C(x,y,t) = \frac{1}{1+\exp(W0(x,y)-\sum_{n} Wn(x,y)\times An)} \cdots (4)$$

By substituting Wn(x,y) of Eqs.(2) and (3) in Eq.(4), we have

$$C(x,y,t) = \frac{1}{1+\dfrac{1-H0(x,y,t)}{H0(x,y,t)} \times \prod_{n=1}^{N} \left(\dfrac{H0(x,y,t)\times(1-Hn(x,y,t))}{(1-H0(x,y,t))\times Hn(x,y,t)} \right)^{An}} \cdots (5)$$

For simplification, rewriting this in a symmetric form regarding C(x,y),H0(x,y) and Hn(x,y), we have,

$$\ln\left\{\frac{C(x,y,t)}{1-C(x,y,t)}\right\} - \ln\left\{\frac{H0(x,y,t)}{1-H0(x,y,t)}\right\} = \sum_{n=1}^{N} An \times \left[\ln\left\{\frac{Hn(x,y,t)}{1-Hn(x,y,t)}\right\} - \ln\left\{\frac{H0(x,y,t)}{1-H0(x,y,t)}\right\}\right] \cdots (6)$$

where, the function $g(x)=\ln(x/(1-x))$ for $0<x<1$, can be approximated by $(4x-2)$ by using Taylor expansion, neglecting terms higher than the third order. Thus we have

$$C(x,y,t) - H0(x,y,t) = \sum_{n=1}^{N} (An \times (Hn(x,y,t) - H0(x,y,t))) \cdots (7)$$

Now Eq.(7) indicates that the general output pattern C(x,y,t) can be approximately represented by a linear combination of En(x,y,t) = Hn-H0 which is the contribution of the hidden unit no.'n' to the output layer, subtracting the bias pattern. Training by back propagation insures the total error e(t), total sum of $(F(x,y) - C(x,y,t))^2$ over all (x,y), decreases as t increases. Now we call these En(x,y,t) with H0(x,y,t), as 'primitives', in which way we could say that any supervisory pattern shown in the output layer must be represented in terms of primitives which are to be optimized as it learns.

Secondly, each coefficient An , of the *primitive* En in Eq.(7) is expressed by the input signal and the input-hidden connection weights

$$An = \frac{1}{1+\exp\left[Pn - \sum_{m=1}^{M}\{Pm,n \times X(m)\}\right]} \cdots (8)$$

where Pn is a threshold of the hidden unit no.'n', Pm,n is a connection weight between the input unit no.'m' and the hidden unit no.'n', and X(m) represents the external input to the unit no.'m' of the input layer. In the case of PPN, we can assume that it is the only time when Fm(x,y), the sample image no. 'm', is shown that no other input unit than no.'m' is fired at maximum of 1(others are all 0). Therefore, in the PPN, the coefficient An^m of the *primitive* En, being used to recollect the sample pattern no.'m' is written as,

$$An^m = \frac{1}{1+\exp(Pn - Pm,n)} \cdots (9)$$

This indicates that the coefficient An^m depends only on the connection weight between the input no.'m' and the hidden no.'n' , so it cannot be corrupted by unnecessary competition whereas the auto-association network can (i.e, in case of auto-association net, the coefficient $A'n^m$ is represented as a function of the product sum of Fm(x,y) and P(x,y)n over all grid(x,y) of the input layer, using the same notation). So, there exists a linear mapping T: Pm,n->P(x,y)n , provided that the organized *primitive* set (En) is identical between the PPN and auto-association, which we show in the proceedings.

Some experimental results of image transform in terms of *primitives* are also shown with their learning speed of data compression for both PPN and auto-association network in the proceedings (we've already acquired some results showing that PPN gets less error by 30 thru 50 % than auto-association does for the same image data with the same number of units and learning steps over a few hundreds).

4. Conclusion

1). we proposed a computational model namely Pulse Pattern Network (PPN) for image representation using back propagation. The usage of the layered network is quite different than that of a multi layered 'Perceptron', since PPN is trained for recollection of the pattern data rather than for perception.
2). we analyzed the internal representation of PPN which learns recollection, so we could introduce an idea of *primitive* referring the basis pattern set (though they are non-orthogonal nor complete, as Gabor wavelets(2)) organized by the network.
3). The clustering (or feature extraction) capability of PPN is also shown by a few experimental results in the proceedings.

Researches are to be made as to the following subject :
1). theoretical study for the capacity of the multi (three or more) layered PPN.
2). application work for feature extraction of images using PPN.
3). analysis of the relationship between the two schemes(Perceptron-like network and PPN) , i.e, how Perceptron and PPN should interact each other to represent and/or recognize the pattern effectively (these two are sometimes referred as the bottom-up approach and the top-down approach (5)(6)(7), so this is actually the problem of bottom-up / top-down interaction).

References

(1). Cottrell,G.W., Munro.P. and Zipser.D. : "Image Compression by Back Propagation: An Example of Extensional Programming", ICS Report 8702 ,(1987)
(2). Daugman.J.G.: "Relaxation Neural Network for Non-Orthogonal Image Transform" ICNN 1988 vol.1 pp.547-560 (1988)
(3). Bourland.H. and Kamp.Y.: " Auto-Association by Multilayer Perceptrons and Singular Value Decomposition",Biol. Cybern.59, pp.291-294 (1988)
(4). Rumelhart.D.E. and McClelland.J.L.: "Parallel Distributed Processing", Vol.1, Chap.8, MIT Press,(1986)
(5). Omori.T and Nagase.T.: "Image Understanding by Neuron Network - representation and operation of internal image-" IJCNN 1989 vol.2 pp.235-240 (1989)
(6). Ullman. S. :"Visual Routines; where bottom-up and top-down meet" in "Pattern Recognition by Humans and Machines", vol.2, no.6, pp.159-218 Academic Press,(1986)
(7). Grossberg.S.:"Neural networks and natural intelligence", Chap.6, MIT Press,(1988)

Learning Theory

Neural Representation of Information

Shun-ichi Amari

Faculty of Engineering, University of Tokyo
Bunkyo-ku, Tokyo 113, Japan

It is interesting to know how information is represented in the brain. There have been considered two different types of representations. One is a localized representation, in which a signal is represented by an excitation of neurons at a localized position of a neural system. THe self-organizing formation of a cortical map, the learning vector quatizer, etc. are such examples. The other is a distruibuted representation, in which a signal is represented by a distributed excitation pattern of a neural system. The associative memory model is a typical example of the distributed representation. We have an intermediate one, a sparsely encoded representation which is a distributed one but the number of excited neurons are relatively few.

It may be of no sense to discuss which the true representation is. We believe that both representations are used in the brain in a modified manner. Neural networks have ability to represent information in either manner, in particular to create such representations by learning or self-organization. The important thing is to elucidate the characteristics, i.e., capabilities and limitations, of both types of neural information representations and to show their merits and demerits. We summarize the theoretical studies on these subjects and give some new results. These are useful not only for elucidating neural memory mechanisms but also for designing neural pattern recognition systems.

In order to show characteristics of automatic formation of localized representation of signals, we show a model of a simple self-organizing nerve field. Such a model was proposed by Willshaw and von der Malsburg [1976]. A revised model was mathematically analyzed in detail by Amari [1980, 1983], Takeuchi and Amari [1979] and by others, and a simple but powerful model was proposed by Kohonen [1982, 1984]. A self-organizing local representation provides with a convenient prepocessor of a large-scale pattern recognition system. It is important to study 1) metrical properties and 2) topological properties of such a representation. We show how the resolution of a localized representation is determined in such a map. At the same time, it is shown that such a model has an amplification property that frequently applied signals become to occupy a larger part of a neural system to represent them. The topological property is much more interesting : How is the topology of a signal space kept in its map (localized representation) in a neural system, when their dimensionality is different ? Takeuchi and Amari [1979] proved an interesting property that an automatic quantization emerges under a certain condition in the map, even when both signal space and neural system have a continuous structure. The Kohonen map does not have this property, because of the lack of dynamics of the neural system.

The correlation type associative memory is a learning distributed representation studied for long years. A signal is represented by an attractor of a recurrently connected neural network. One interesting problem is the information capacity of such a representation : How many attractors can be formed without confusion in a network of n elements ? Another interesting problem is the areas of the basins of attraction and its dynamical process of recalling. It is also interesting to know the effect of encoding a signal into a distributed excitation pattern to be consolidated as an attractor.

It was known that, for a random encoding, the capacity is $n/(2\log n)$, if exact recalling is required (McEliece et al. [1987]). The capacity is about $0.15n$, if approximate recalling is permitted (Hopfield [1982], Amit et al. [1985], Amari and Maginu [1988]). The strange behavior of dynamics of recalling process was analyzed by Amari and Maginu [1988]. A capcity of about $0.27n$ is obtained by Morita [1988] by modifying the recalling dynamics. It was also shown that a cascaded neural system has a larger capacity of $0.27n$ (Meir and Domany [1987]). Amari [1988] showed that a recurrent net has also a larger capacity of $0.27n$, when a number of sequences of patterns are memorized in a recurrent network as state transition sequences instead of attractors.

It was known that the information capacity increases drastically (Palm [1980]), if a sparse encoding scheme is used. However, the basin of attraction is small in this case. Amari [1989] showed that the information capacity is $n^2/\{24(\log n)^2\}$ in a sparse encoding case and that the basin of attraction is sufficiently large, if an activity control mechanism is added.

A temporary or working memory is required to have a huge information capacity, because it is required to keep any signals temporarily as steady states. To this end, it is known that a randomly connected symmetric network has a capacity of about $2^{0.3n}$. On the other hand, a basic competition model of Amari and Arbib [1977] has a capacity of ${}_nC_k$, because it can store any pattern in which k neurons are excited out of n. It is another interesting problem how to encode signals in the case of a working memory and how to compare one signal with others. Recent physiological findings of Miyashita [1988] are suggestive in this respect.

The present paper discusses these problems from a theoretical point of view.

References

S. Amari and M. A. Arbib, Competition and cooperation in neural nets. *Systems Neuroscience* (J. Metzler ed.), pp.119-165, Academic Press, 1977

S. Amari, Topograhic organization of nerve fields. *Bull. of Math. Biology*, Vol.42, pp.339-364, 1980

S. Amari, Field theory of self-organizing neural nets, *IEEE Trans. Systems, Man and Cybernetics*, Vol.SMC-13, Nos. 9 & 10, pp.741-748, 1983

S. Amari, Associative memory and its statistical-neurodynamical analysis. *Neural and Synergetic Computers*, ed. H. Haken, Springer, Series in Synergetics, 42, pp. 85-99, 1988

S. Amari and K. Maginu, Statistical neurodynamics of associative memory. *Neural Networks*, Vol.1, No. 1, pp.63-73, 1988

S. Amari, Characteristics of sparsely encoded associative memory. *Neural Networks*, 2, in press

D. J. Amit, H. Gutfreund and H. Sompolinsky, Storing infinite numbers of patterns in a spin glass model of neural networks. *Physical Review Letters*, 55, pp.1530-1533, 1985

J. J. Hopfield, Neural networks and physical systems with emergent collective computational abilities. *P. Nat. Acad. Sci. U.S.A.*, vol. 79, pp. 2445-2458, 1982

T. Kohonen, Self-organized formation of topologically correct feature maps. Biol. Cybern., 43, pp.59-69, 1982

T. Kohonen, *Associative Memory and Self-Organization.* Springer-Verlag, 1984

R. J. McEliece et. al., The Capacity of the Hopfield associative memory. *IEEE Trans., Inf. Theory*, IT-33, pp. 461-482, 1987

R. Meir, and E. Domany, Exact solution of a layered neural network memory. *Phy. Rev. Lett.*, 59, pp. 359-362, 1987

Y. Miyashita, Neural correlate of visual associative long-term memory in the primate temporal cortex. *Nature*, vol. 335, 817-820, 1988

M. Morita, to appear

G. Palm, On associative memory. *Biol. Cybern*, 36, pp. 646-658, 1980

A. Takeuchi and S. Amari, Formation of topographic maps and columnar microstructures. *Biol. Cybernetics*, Vol.35, pp.63-72, 1979

D. J. Willshaw and C. von der Malsburg, How patterned neural connections can be set up by self-organization. *Proc. Roy. Soc*, B-194, pp.431-445, 1976

Adjoint-Operator Algorithms for Learning in Neural Networks

J. Barhen **N. Toomarian** **S. Gulati**

Center for Space Microelectronics Technology
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

Abstract : A new methodology for neural learning of nonlinear mappings is presented. It exploits the concept of ***adjoint operators*** to enable a fast global computation of the network's response to perturbations in all system parameters.

1. Introduction

A considerable effort has recently been devoted to the development of efficient computational methodologies for learning. Attention has largely focussed on the back-propagation algorithm because of its simplicity, generality and the promise that it has shown in regard to various applications [6,8]. More recently, Pineda [5] has derived a generalization to back-propagation for recurrent networks. In a similar vein, Williams and Zipser [9] have presented algorithms for learning tasks with temporal dependencies. Pearlmutter [4] has proposed a similar technique which minimizes an error functional between output and targeted temporal trajectories. In a significantly different approach, Barhen, Gulati and Zak [2,3] recently introduced neural formalisms to efficiently learn nonlinear mappings using a new mathematical construct, i.e., terminal attractors [10]. Terminal attractor representations were used not only to ensure infinite local stability of the encoded information, but also to provide a qualitative as well as quantitative change in the nature of the learning process. In particular, they imply loss of Lipschitz conditions at energy function minima, which results in a dramatic increase in the speed of learning.

The development of learning algorithms is generally based upon the minimization of a "neuromorphic" energy-like function. A fundamental requirement of all previously mentioned methods is the computation of the gradient of this objective function with respect to the various parameters of the neural architecture, e.g., synaptic weights, neural gain, etc. In the present paper we introduce a new methodology for their efficient analytical computation, as a single solution of a set of "adjoint" equations.

2. Adjoint Operators

Consider, for the sake of generality, that a problem of interest is represented by the following system of N coupled nonlinear equations

$$\bar{\varphi}(\bar{u},\bar{p}) = 0 \tag{2.1}$$

where $\bar{\varphi}$ denotes a nonlinear operator [11]. Let $\bar{u}$ and $\bar{p}$ represent the N-vector of dependent variables and the M-vector of system parameters, respectively. We will assume that generally $M >> N$ and that elements of $\bar{p}$ are, in principle, independent. Furthermore, we will also assume that, for a specific choice of parameters, a unique solution of Eq. (2.1) exists. Hence, $\bar{u}$ is an implicit function of $\bar{p}$. A system response, R, represents any result of the calculations that is of interest. Specifically

$$R = R(\bar{u},\bar{p}) \tag{2.2}$$

i.e., R is a known nonlinear function of $\bar{p}$ and $\bar{u}$ and may be calculated from (2.2) when the solution $\bar{u}$ in Eq. (2.1) has been obtained for a given $\bar{p}$. The problem of interest is to compute the "sensitivities" of R, i.e., the derivatives of R with respect to parameters p_μ, $\mu = 1,\cdots,M$. By definition

$$\frac{dR}{dp_\mu} = \frac{\partial R}{\partial p_\mu} + \frac{\partial R}{\partial \bar{u}} \cdot \frac{\partial \bar{u}}{\partial p_\mu} \tag{2.3}$$

Since the response R is known analytically, the computation of $\partial R/\partial p_\mu$ and $\partial R/\partial \bar{u}$ is straightforward. The quantity that needs to be determined is the vector $\partial \bar{u}/\partial p_\mu$. Differentiating the state equations (2.1), we obtain a set of equations to be referred to as "forward" sensitivity equations

$$\frac{\partial \bar{\varphi}}{\partial \bar{u}} \cdot \frac{\partial \bar{u}}{\partial p_\mu} = -\frac{\partial \bar{\varphi}}{\partial p_\mu} \tag{2.4}$$

To simplify the notations, we are omitting the "transposed" sign and denoting the N by N forward sensitivity matrix $\partial \bar{\varphi}/\partial \bar{u}$ by **A**, the N-vector $\partial \bar{u}/\partial p_\mu$ by ${}^\mu\bar{z}$ and the "source" N-vector $-\partial \bar{\varphi}/\partial p_\mu$ by ${}^\mu\bar{s}$. Thus

$$A\,{}^\mu\bar{z} = {}^\mu\bar{s} \tag{2.5}$$

Computation of the response gradient using the forward sensitivity equations would require solving a system of N nonlinear algebraic equations for each parameter p_μ, since the source term in Eq. (2.5) explicitly depends on μ. This difficulty is circumvented by introducing adjoint operators. Let A^* denote the formal adjoint of the operator A [1,7,12]. The adjoint sensitivity equations can then be expressed as

$$A^*\,{}^\mu\bar{z}^* = {}^\mu\bar{s}^*. \tag{2.6}$$

By definition, for algebraic operators [12]

$${}^\mu\bar{z}^* \cdot (A\,{}^\mu\bar{z}) = {}^\mu\bar{z}^* \cdot {}^\mu\bar{s} = {}^\mu\bar{z} \cdot (A^*\,{}^\mu\bar{z}^*) = {}^\mu\bar{z} \cdot {}^\mu\bar{s}^* \tag{2.7}$$

Since Eqs. (2.3), can be rewritten as

$$\frac{dR}{dp_\mu} = \frac{\partial R}{\partial p_\mu} + \frac{\partial R}{\partial \bar{u}}\,{}^\mu\bar{z}, \tag{2.8}$$

if we identify

$$\frac{\partial R}{\partial \bar{u}} \equiv {}^\mu\bar{s}^* \equiv \bar{s}^* \tag{2.9}$$

we observe that the source term of the adjoint equations is independent of the specific parameter p_μ. Hence, the *solution of a single set of adjoint equations will provide all the information required to compute the gradient of R with respect to all parameters.* To underscore that fact we shall denote ${}^\mu\bar{z}^*$ as $\bar{v}$. Thus

$$\frac{dR}{dp_\mu} = \frac{\partial R}{\partial p_\mu} + \bar{v} \cdot {}^\mu\bar{s}. \tag{2.10}$$

3. Applications to Neural Learning

We formalize a neural network as an adaptive dynamical system whose temporal evolution is governed by the following set of coupled nonlinear differential equations

$$\dot{u}_n + \kappa_n u_n = \sum_m T_{nm}\, g(\gamma_m u_m) + {}^kI_n \tag{3.1}$$

where u_n represents the mean soma potential of the nth neuron and T_{nm} denotes the synaptic coupling from the $m-$th to the $n-$th neuron. The constant κ_n characterizes the decay of neuron activity. The sigmoidal function $g(\cdot)$ modulates the neural response, with gain given by γ_m; typically, $g(\gamma z) = \tanh(\gamma z)$. The "source" term, ${}^kI_n$ encodes component-contribution by the presented attractors ${}^k\bar{a}$ of the k-th training pattern via the expression

$${}^kI_n = \begin{cases} [{}^ka_n - g(\gamma_n u_n)]^\beta & \text{if } n \in S_X \\ 0 & \text{if } n \in S_H \cup S_Y \end{cases}. \tag{3.2}$$

The topographic input, output and hidden network partitions S_X, S_Y and S_H are architectural requirements related to the encoding of mapping-type problems. Details are given in [2]. In previous articles [2,3,10] we have demonstrated that in general, for $\beta = (2i+1)^{-1}$ and i a strictly positive integer, such attractors have infinite local stability and provide opportunity for learning in real-time.

To proceed formally with the development of a learning algorithm, we consider an approach based upon the minimization of a constrained "neuromorphic" energy-like function E given by the following expression

$$E(\bar{u},\bar{\lambda},\bar{p}) \;=\; \frac{1}{2}\sum_n\sum_m \omega_{nm}\,T_{nm}^2 \;+\; \frac{1}{\alpha}\sum_k\sum_n {}^k\lambda_n\;{}^k\Gamma_n^\alpha \tag{3.3}$$

where the constraints are of the form,

$${}^k\Gamma_n \;=\; \begin{cases} {}^k a_n \;-\; g(\gamma_n\;{}^k\tilde{u}_n) & \text{if } n\;\in S_X \cup S_Y \\ 0 & \text{if } n\;\in\; S_H \end{cases} \tag{3.4}$$

Typically, a positive value like 2 is used for α. The weighting factor ω_{nm} is constructed in such a fashion, as to favor locality of computation. The indices n, m span over all neurons in the network. Lagrange multipliers corresponding to the $k-n$th constraint are denoted by ${}^k\lambda_n$. The superscript $\sim$ denotes quantities evaluated at steady state. The proposed objective function includes contributions from two sources. First, it enforces convergence of every neuron in S_X and S_Y to attractor coordinates corresponding to the components in the input-output training patterns, thereby prompting the network to learn the underlying invariances. Secondly, it regulates the topology of the network by minimizing interconnection strengths between distant synaptic elements to favor locality of computation.

Lyapunov stability requires an energy-like function to be monotonically decreasing in time. Since in our model the internal dynamical parameters of interest are the synaptic strengths T_{ij} of the interconnection topology, the characteristic decay constants κ_i, the gain parameters γ_i and the Lagrange multipliers ${}^k\lambda_i$, this implies that

$$\dot{E} \;=\; \sum_i\sum_j \frac{dE}{dT_{ij}}\,\dot{T}_{ij} \;+\; \sum_i \frac{dE}{d\kappa_i}\,\dot{\kappa}_i \;+\; \sum_i \frac{dE}{d\gamma_i}\,\dot{\gamma}_i \;+\; \sum_l\sum_i \frac{dE}{d\,{}^l\lambda_i}\,{}^l\dot{\lambda}_i \;<\; 0 \tag{3.5}$$

One can always choose, with $\tau_T \;>\; 0$

$$\dot{T}_{ij} \;=\; -\tau_T\,\frac{dE}{dT_{ij}} \tag{3.6}$$

where τ_T introduces an adaptive parameter for learning, [see, e.g., [2,3]) Similar expressions can be constructed for $\dot{\kappa}$ and $\dot{\gamma}$, e.g.,

$$\dot{\kappa}_i \;=\; -\tau_\kappa\,\frac{dE}{d\kappa_i} \qquad \text{and} \qquad \dot{\gamma}_i \;=\; -\tau_\gamma\,\frac{dE}{d\gamma_i} \tag{3.7}$$

with τ_κ, $\tau_\gamma \;>\; 0$. Then, substituting in Eq. (3.5) and denoting by $\oplus$ tensor contraction, one obtains

$$\nabla_\lambda E \;\oplus\; \dot{\lambda} \;<\; \tau_T\,(\,\nabla_T E \oplus \nabla_T E\,) \;+\; \tau_\kappa\,(\,\nabla_\kappa E \oplus \nabla_\kappa E\,) \;+\; \tau_\gamma\,(\,\nabla_\gamma E \oplus \nabla_\gamma E\,) \tag{3.8}$$

Without loss of generality, one can assume $\tau \;=\; \tau_T \;=\; \tau_\kappa \;=\; \tau_\gamma$. The equations of motion for the Lagrange multipliers ${}^l\lambda_i$ must now be constructed in such a way that Eqn. (3.8) be strictly satisfied. In addition, when the constraints are satisfied, i.e., as ${}^l\Gamma_n \;\rightarrow 0$ in Eqn. (3.4), we require that ${}^l\dot{\lambda}_i \;\rightarrow\; 0\;\;\forall\; l$. We have adopted the following analytical model for the evolution of λ,

$${}^l\dot{\lambda}_i \;=\; \tau\,\frac{\Pi}{\Lambda \;+\; 1/(\Lambda \;+\; \theta)}\;{}^l[\nabla_\lambda E]_i \tag{3.9}$$

where $\Pi \;=\; \nabla_T E \oplus \nabla_T E \;+\; \nabla_\kappa E \oplus \nabla_\kappa E \;+\; \nabla_\gamma E \oplus \nabla_\gamma E$ and $\Lambda \;=\; \nabla_\lambda E \oplus \nabla_\lambda E$ and θ is an arbitrary positive constant. It is straightforward to prove that this model fulfills the above requirements.

In relating adjoint theory to the neural learning algorithms, we identify the neuromorphic energy-like function, E in Eq. (3.3), with the system response R. Let $\bar{p}$ denote the following system parameters:

$$\bar{p} \;=\; \{\, T_{11}, \cdots T_{NN} \mid \kappa_1, \cdots \kappa_N \mid \gamma_1, \cdots \gamma_N \mid \cdots \} \tag{3.10}$$

The adiabatic solution to the nonlinear equations of motion (3.1), for each training pattern k, $k = 1, \cdots K$ is given by

$${}^k\varphi_n(\,{}^k\tilde{u}, \bar{p}) \;=\; -\kappa_n\,{}^k\tilde{u}_n \;+\; \sum_m T_{nm}\; g(\gamma_m\,{}^k\tilde{u}_m) \;+\; {}^kI_n \;=\; 0. \tag{3.11}$$

So, in principle ${}^k\tilde{u}_n \;=\; {}^k\tilde{u}_n\,[T,\ \bar{\kappa},\ \bar{\gamma},\ {}^ka_n, \cdots]$. Using Eqs. (3.11), the forward sensitivity matrix can be computed and compactly expressed as

$$\begin{aligned} {}^kA_{nm} \;=\; \frac{\partial\,{}^k\varphi_n}{\partial\,{}^k\tilde{u}_m} \;&=\; [\,-\kappa_n \;+\; \frac{\partial\,{}^kI_n}{\partial\,{}^k\tilde{u}_m}]\;\delta_{nm} \;+\; T_{nm}\,{}^k\hat{g}_m\,\gamma_m \\ &=\; {}^k\eta_n\;\delta_{nm} \;+\; \gamma_m\,{}^k\hat{g}_m\;T_{nm} \end{aligned} \tag{3.12}$$

where $\hat{g}_m$ represents the derivative of g_m with respect to u_m. The adjoint sensitivity matrix is

$${}^kA^*_{nm} \;=\; {}^k\eta_m\delta_{mn} \;+\; \gamma_n\,{}^k\hat{g}_n\;T_{mn}. \tag{3.13}$$

Using Eqs. (2.9) and (3.3), we can compute the adjoint source,

$${}^ks^*_n \;=\; -\,{}^k\lambda_n\;{}^k\Gamma_n^{\alpha-1}\;\gamma_n\,{}^k\hat{g}_n \tag{3.14}$$

The system of adjoint equations can then be constructed using Eqs. (3.13) and (3.14), to yield :

$$\sum_m\,[\;{}^k\eta_m\;\delta_{mn} \;+\; T_{mn}\,{}^k\hat{g}_n\,\gamma_n\;]\;{}^k\tilde{v}_m \;=\; -\,{}^k\lambda_n\;{}^k\Gamma_n^{\alpha-1}\;\gamma_n\,{}^k\hat{g}_n \tag{3.15}$$

Notice that the above system, (3.15), is linear in ${}^k\tilde{v}$. Furthermore, its components can be obtained as the equilibrium points, (i.e., $\dot{v}_i \;\rightarrow\; 0$) of the concomittant dynamical system

$${}^k\dot{v}_n \;-\; {}^k\eta_n\,{}^kv_n \;=\; \gamma_n\,{}^k\hat{g}_n\;[\,\sum_m T_{mn}\,{}^kv_m \;+\; {}^k\lambda_n\;{}^k\Gamma_n^{\alpha-1}\,] \tag{3.16}$$

To proceed with our derivation of learning algorithms, we differentiate the steady state equations (3.11) with respect to each parameter, p_μ, to obtain the forward source term, ${}^\mu s^k_n$

$${}^\mu s^k_n \;=\; -\Big(\,[\,-{}^k\tilde{u}_i\,]\;\delta_{p_\mu,\kappa_i} \;+\; [\;\delta_{ni}\;g(\gamma_j\,{}^k\tilde{u}_j)\,]\;\delta_{p_\mu,T_{ij}} \;+\; [\;T_{ni}\,{}^k\hat{g}_i\,{}^k\tilde{u}_i + \frac{\partial\,{}^kI_n}{\partial\gamma_i}\,]\delta_{p_\mu,\gamma_i}\Big) \tag{3.17}$$

Substituting Eq. (3.17) in (2.10), and recalling that our abstract response corresponds here to the energy function E, yields

$$\frac{dE}{dp_\mu} \;=\; \frac{\partial E}{\partial p_\mu} \;+\; \sum_k {}^k\tilde{v}\cdot{}^\mu\,\tilde{s}^k \tag{3.18}$$

The explicit energy gradient contributions for parameters $p_\mu \;=\; T,\ \bar{\kappa},\ \bar{\gamma}$, immediately result :

$$\frac{dE}{dT_{ij}} \;=\; \omega_{ij}\;T_{ij} \;-\; \sum_k {}^k\tilde{v}_i\;g(\gamma_j\;{}^k\tilde{u}_j) \tag{3.19}$$

$$\frac{dE}{d\kappa_i} \;=\; \sum_k {}^k\tilde{u}_i \sum_n {}^k\tilde{v}_n \tag{3.20}$$

$$\frac{dE}{d\gamma_i} = -\sum_k {}^k\lambda_i \, {}^k\Gamma_i^{\alpha-1} \, {}^k\hat{g}_i \, {}^k\tilde{u}_i + \sum_k \sum_n [\, T_{ni} \, {}^k\hat{g}_i \, {}^k\tilde{u}_i + \frac{\partial \, {}^kI_n}{\partial \gamma_i} \,] \, {}^k\tilde{v}_n \tag{3.21}$$

Substituting Eqs. (3.19)-(3.21) into Eqs. (3.6) and (3.7), we then obtain the complete learning dynamics.

4. Conclusions

In this paper we have presented a new theoretical framework for learning continuous nonlinear mappings using artificial neural networks. Central to our approach is the concept of *adjoint operators* which enables a fast computation of energy function gradients with respect to all system paramters using a single solution of the adjoint equations.

Acknowledgements

The research described in this paper was performed by the Center for Space Microelectronics Technology, Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by agencies of the U.S. Department of Defense, and by the Office of Basic Energy Sciences of the U.S. Department of Energy, through agreements with NASA. We wish to acknowledge discussions with M. Zak and F. Pineda.

References

[1] R.G. Alsmiller and J. Barhen, "The Application of Adjoint Sensitivity Theory to a Liquid Fuels Supply Model", *Energy*, 9(3), 1984, 239-253; and references therein.

[2] J. Barhen, S. Gulati and M. Zak, "Neural Learning of Constrained Nonlinear Transformations", *IEEE Computer*, 22(6), 1989, 67-76.

[3] J. Barhen, M. Zak and S. Gulati, " Fast Neural Learning Algorithms Using Networks with Non-Lipschitzian Dynamics", in *Proc. Neuro-Nimes '89*, Nimes, France, 1989 (in press).

[4] B.A. Pearlmutter, "Learning State Space Trajectories in Recurrent Neural Networks", *Neural Computation*, 1(2), 1989, 263-269.

[5] F.J. Pineda, "Dynamics and Architecture in Neural Computation" *Journal of Complexity*, 4, 1988, 216-245.

[6] D.E. Rumelhart and J.L. McClelland, *Parallel and Distributed Processing*, MIT Press, Cambridge, MA, 1986.

[7] N. Toomarian, E. Wacholder and S. Kaizerman, "Sensitivity Analysis of Two-Phase Flow Problems", *Nucl. Sci. Eng.*, 99(1), 1987, 53-81.

[8] P. Werbos, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences", *Ph.D. Thesis*, Harvard Univ., 1974.

[9] R.J. Williams and D. Zipser, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks", Neural Computation, 1(2), 1989, 270-280.

[10] M. Zak, "Terminal Attractors for Addressable Memory in Neural Networks," *Physics Letters A*, 133, 1988, 218-222.

[11] If differential operators appear in Eq. (2.1), then a corresponding set of boundary and/or initial conditions to specify the domain of φ must also be provided. The learning model discussed in this paper focuses on the adiabatic approximation only (steady state networks). Nonadiabatic learning algorithms will be discussed in a forthcoming article.

[12] Adjoint operators can only be considered for densely defined linear operators on Banach spaces. For the neural application under consideration we will limit ourselves to real Hilbert spaces. Such spaces are self-dual. Furthermore, the domain of an adjoint operator is determined by selecting appropriate adjoint boundary conditions [11]. The associated bilinear form evaluated on the domain boundary must generally be also included.

A Method To Establish An Autonomous Self-Organizing Feature Map

Russel E. Hodges and Chwan-Hwa Wu

Electrical Engineering Department, 200 Broun Hall, Auburn University, Auburn, AL 36849

Abstract

A set of nonlinear differential equations is developed to adaptively govern the learning rate and the radius of the bubble for the Kohonen's Self-Organizing Feature Map (SOFM). This set of equations not only preserves the features of the Kohonen's SOFM, but also autonomously accelerates the learning process for "familiar" inputs. The behavior of the set of nonlinear differential equations is presented. The performance of this autonomous SOFM is measured and compared to that of Kohonen's SOFM.

INTRODUCTION

Kohonen's Self Organizing Feature Map (SOFM) is a widely used vector quantizing neural network paradigm [1-3]. The short-cut algorithm [1,4] requires that the functions governing the learning rate an neighborhood radius are provided by users. However, the optimal set of functions are different for various applications. Therefore, the SOFM paradigm depends upon human supervision to achieve the optimal performance.

The goal of this study is to obtain an autonomous vector quantizer based on Kohonen's SOFM. This vector quantizer is adaptable, based on the relationship between the weights and input, to any distribution of input data and any size of map. Therefore, the SOFM can approach an autonomous neural network for suitable applications. The approach presented in this paper is based on an intuitive sense to describe the set of nonlinear differential equations which can handle the learning process autonomously.

The Kohonen SOFM Paradigm

Kohonen's Algorithm creates a vector quantizer by adjusting weights from common input nodes to L output nodes arranged in a two-dimensional grid as shown in figure 1 [1]. Using the user-specified equations for the learning rate and bubble radius, continuous-valued input vectors are presented sequentially in time without specifying the desired output. After enough input vectors have been presented, the SOFM can cast the weights of the nodes according to the probability density function of the input vectors. Moreover, the slowly decaying learning rate is able to allow weights to be organized such that the topologically close nodes can interpolate weights among themselves. Therefore, the weights of nodes are ordered in a topological manner.

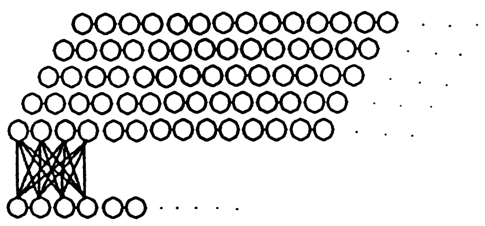


fig. 1. General 2-D node arrangement for the Kohonen self organizing map

Developing an Autonomous Model

The approach to be used in developing the autonomous vector quantizer is to specify the basic physics that occur in the vector quantization process. With the physics specified, a system of differential equations that govern the learning rate and neighborhood radius will be constructed using the specified constraints.

Physics of the Autonomous SOFM

We believe that a set of differential equations to self-organize the network adaptively should be developed based on the current state of the nodes and their relationship to the input vector. An important element of the vector quantizer is that when an input vector is applied to the network, the connection weights in the bubble adjust to decrease the vector distance between the input and the connection weights of the neurons within the bubble [4]. This distance decreases geometrically until the differential equations governing the updating of the weights have converged to a satisfactory degree. In addition, the connection weights of the neurons inside the bubble are adjusted to form the interpolation effects. These include two effects of the learning process: First, the weights of topologically close nodes can affect each other by the excitation of a number of close input vectors. Secondly, the adaptive radius of the bubble can provide a smooth transport of the weights to form a topological order. After a basis of the SOFM is established, the learning process should be accelerated adaptively toward the simple recognition and updating process. Instead of an even amount of processing for every input vector as Kohonen's SOFM, a true neural network is believed to be capable of performing the learning process faster after the basic knowledge is acquired. Based on the physics described above, a system of nonlinear differential equations [5] governing the learning rate and bubble size adaptively is presented in the following.

Development of the Governing Differential Equations

Define the following terms:

R(t): Neighborhood radius at time t
α(t): Learning rate at time t
d(t): Minimum distance between the input vector and all of the connection vectors at time t

The system of differential equations will first be given followed by an explanation of the dynamics of R(t) and α(t).

$$\frac{d\alpha(t)}{dt} = \sigma^{\gamma}[e^{-\sigma} e^{-1/\gamma} \tanh(1+\alpha\gamma)/R^2(0) - \alpha e^{-\alpha}] - \alpha \tag{1-a}$$

$$\frac{dR(t)}{dt} = -\sigma^{\gamma} e^{-(1-\alpha)} e^{-1/\sigma} e^{-(1-\gamma)} - \sigma^2 e^{-\sigma^2} \tag{1-b}$$

where $\gamma = d/d(0)$ and $\sigma = R/R(0)$

α(t) is usually taken to be a monotonically decreasing function of time. For the construction of the differential equations α is forced to start from zero and increase rapidly in an oscillating manner. The effect of this oscillation is to "shake" the connection weights to give a smooth interpolation effects for the connection weights within a certain neighborhood. α continues to increase until a balance point is achieved. This balancing point is a function of the present value of α(t), R(t), and d(t). Once α(t) has reached its maximum value, it begins to decay in a near exponential manner. α(t) decreases to a very small number and then approaches zero asymptotically. The actual behavior of α for the first input vector to the SOFM can be seen in figure 2.

R(t) is taken to be a monotonically decreasing function. The rate at which R(t) decays depends on the present state of R(t), α(t), and d(t). As d(t) becomes smaller, R(t) should decrease faster. This ensures that the nodes within the initial bubble are not all shifted too much toward the input. As α(t), increases R(t) should decrease at a faster rate. This ensures that the right interpolation effects are maintained for the updating of the nodes. As α(t) decreases, the rate at which R(t) decreases is reduced. This also helps to establish the interpolation effect to transport weights of nodes in a topological order. The present value of R(t) is used to further decrease R(t). This produces the exponentially decaying effect of R(t). R(t) can be seen in figure 3.

The adaptive acceleration of the learning process can be pictured from $\sigma^{\gamma}$and $e^{-(1-\gamma)}$ as follows (not limited to these two terms). After a rough frame of the SOFM is established, d(t) is not so large as that of the early phase to establish the SOFM. Since γ and σ are normalized, and α has almost the same magnitude for a new input vector as that of the α in the early phase of building the SOFM, d(t) can be reduced rapidly due to the same learning rate. This causes σ^{γ} and $e^{-(1-\gamma)}$ to grow faster than during the early phase to establish the SOFM. Consequently, R is reduced faster. As seen from Eq.(1), the combination of $e^{-1/\gamma}$ and $e^{-\sigma}$ causes α to decrease faster due to the fact that the former has a more significant effect than that of the latter. Thus, the subsequent feed backs between α and R can make the learning process accelerate.

1.) $e^{-1/\gamma}$: This term contributes to the average decreasing trend of $\alpha(t)$ before allowing d to become zero.
2.) $e^{-\sigma}$: This term contributes to the average increasing trend of $\alpha(t)$ as R(t) is decreasing. The effect of this term becomes insignificant as R(t) approaches approaches zero due to the interactions of the other terms.
3.) $\tanh(1+\alpha\bullet\gamma)$: This term helps to stabilize $\alpha(t)$ during the increasing phase.
4.) $\alpha e^{-\alpha}$: This term maintains a bound on how large $\alpha(t)$ is allowed to become. This function also establishes the decay of α.
5.) σ^{γ}: This is an adaptive term to control the learning rate and radius based on the current R and d.
6.) $e^{-(1-\alpha)}$: This term tends to decrease R(t) faster for larger α and vice versa.
7.) $e^{-1/\sigma}$: When R approaches zero, this term diminishes the changing rate of R. Therefore, R(t) can never become a negative number.
8.) $e^{-(1-\gamma)}$: At the early phase of establishing the SOFM, this term provides a time period such that the radius is appropriate for the weights to be organized in a topological order. When d becomes small, this term helps to decrease the radius faster.
9.) $\sigma^2 e^{-\sigma^2}$: This term contributes to the decay of R.

A set of initial conditions that work well for the system of equations are:

$\alpha(0) = 0$

$R(0) = \left[N_x N_y / \pi \right]^{1/4}$ where N_x and N_y are the number of nodes in the x and y direction respectively

The results of the simulation are shown in figures 1-4. Figure 1 compares the processing times of the autonomous SOFM and Kohonen's SOFM as the number of training vectors increase. The speed of the autonomous SOFM is superior to that of Kohonen's SOFM because thousands of input samples are required to form an organized feature map. The adaptive acceleration can be observed after 40 samples are presented. Figure 2 shows a typical $\alpha(t)$ for one training sample. Figure 3 shows a typical R(t) for one training sample. Figure 4 shows the weight distribution after 5,000 uniformly distributed training samples are applied to the map. Note that the initial distribution of the weight vectors is centered in a small box in the middle of the map. The performance is comparable to that of the Kohonen's SOFM.

This particular set of differential equations can work relatively well for all the cases tested. There are other terms that could be added to enhance the overall performance. A problem with this particular set of differential equations is in the rate of convergence. As $\alpha(t)$ becomes very small, the rate at which $\alpha(t)$ converges to zero decreases as well. For the simulation presented, the stopping criterion for a training sample was that the rate of change of d(t) approaches zero. With $\alpha(t)$ converging to zero slowly this criterion took a large number of iterations to reach for the first few thousand training samples. A possible solution to this problem is to add some nonlinearities into the equations that take into account the derivatives of α, R, and d. These nonlinearities can be used to enhance the overall performance of the differential equations and are under investigation.

Concluding Remarks

An autonomous SOFM which is able to control the learning process adaptively is presented. A set of nonlinear differential equations is developed to govern the learning rate and the radius of the bubble adaptively based on the current state of the network and their relationship to the input vector. This set of equations allows the network to preserve the features of Kohonen's SOFM. In addition, acceleration of the learning process is observed after the basis of the network is set up. The speed of learning of this paradigm is superior to that of the Kohonen's SOFM. We believe that the behavior of the system is closer to that of human beings.

References

1. R.P. Lippmann, "An Introduction to Computing with Neural Nets," *IEEE ASSP* Magazine, 1987, pp 36-54.
2. T. Kohonen, Self-Organization and Associative Memory, Second Edition, Springer-Verlag, 1988.
3. T. Kohonen, "Self-Organized Formation of Topologically Correct Feature Maps," *Biol. Cybern.*, 43, 1982, pp 59-69.
4. T. Kohonen, "The "Neural" Phonetic Typewriter," *Computer* 21, 1988, pp 11-22
5. R.L. Devaney, An Introduction to Chaotic Dynamical Systems, Benjamin/Cummings, 1986
6. T.Kohonen, "Self-Organizing Feature Maps," *ICNN 88,* Technical tutorial seminar, 1988.

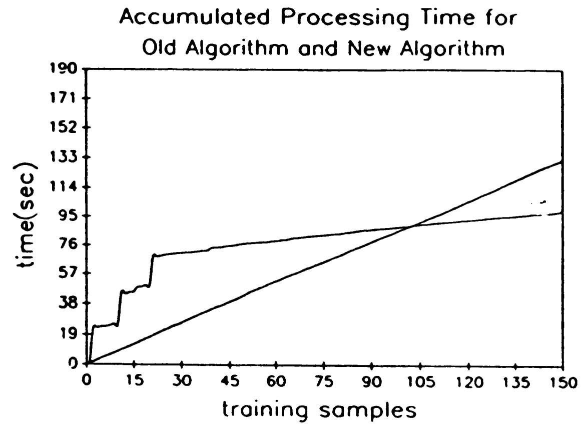


fig. 1. The straight line corresponds to Kohonen's algorithm. The other curve corresponds to the new algorithm.

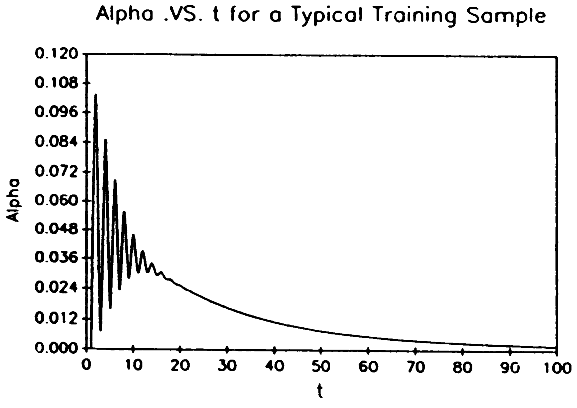


fig. 2. $\alpha(t)$ for a training sample.

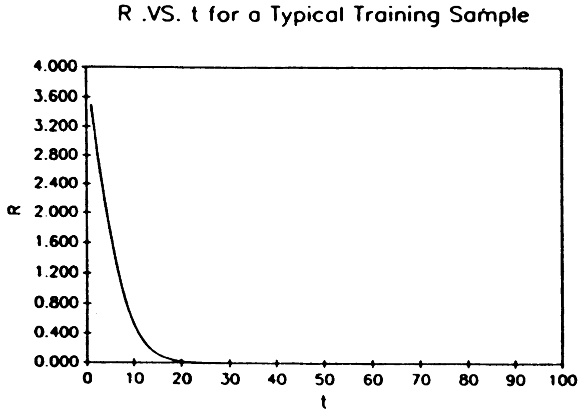


fig. 3. R(t) for a training sample.

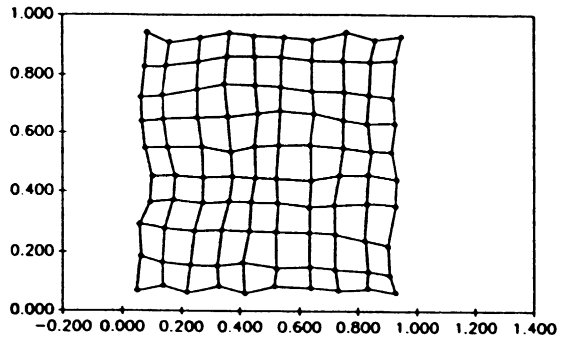


fig. 4. Final weight distribution. (5000 inputs)

Expectation Driven Learning with an Associative Memory

by G.Lukes, B.Thompson and P.Werbos
National Science Foundation and SYSCON[1]
Washington, D.C. 20550

Introduction

Supervised learning requires an explicit target paired with each input. When explicit targets are not available, internally generated targets are required. Expectation Driven Learning (EDL) generates internal expectations that provide these targets for any supervised learning method. A system which incorporates EDL must be provided intrinsic measures of value. Expectations are of these measures of value and provide targets for learning after each action (i.e. real time learning). In our experiments, the automaton (which moves on a two-dimensional grid) has expectations of the minimum future cost of actions leading to a goal state; learning occurs when expectations in the associative memory are modified. In these experiments, we vary the degree of generalization in the associative memory and note the effect on learning. EDL is well suited to problems where a model of the environment may not be available and must be acquired or refined through experience.

Overview of the Method and Prior Literature

Expectation driven approaches to reinforcement learning and utility maximization over time have received attention from many authors. Werbos (1989b) has argued that this problem is central to understanding human intelligence as well as a number of important engineering applications. For example, Jordan (1989) and Kawato (1989) have shown how trajectory-following problems in robotics can be solved by maximizing a complex utility function which includes terms to represent distance from the desired trajectory, the smoothness of the motion, and so on.

To maximize utility across future time, one needs some way to account for the link between present actions and future results. Neural network literature describes two ways of doing this. One way is to obtain an exact model of the external environment and use backpropagation through time to account for that link. This method has been used in the Nguyen and Widrow (1989) truck backer-upper, in Kawato's and Jordan's robots, and in the Werbos (1989d) model of the natural gas industry. The other approach is the adaptive critic approach.

The term "adaptive critic" was coined by Barto, Sutton and Anderson (1983), but it aptly describes a larger family of methods. An adaptive critic method is any method which includes the adaptation of a "critic" network -- a network which produces a kind of global evaluation of how well the network is doing. By this definition, Klopf (1982), Grossberg (1988), Werbos (1977, 1989a and b), Sutton (1988), Williams (1988), and Holland (1975) may all be seen as adaptive critic methods. (See Grossberg (1982, 1988) and Werbos (1989a) for discussions of the requirements of large scale problems. Lakoff (1987) discusses the induction of higher order categories.)

Anderson (1987) explores the use of backpropagation with adaptive critics. Werbos (1989b), in defining Heuristic Dynamic Programming (HDP), allows for any supervised learning network but does not give actual simulations. Booker (1988) demonstrates a classifier based system. In Sutton (1988), temporal difference methods provide results for a similar class of problems. The remainder of this paper will discuss the adaptive critic, the associative memory we use as a critic network, and the experimental results, in that order.

The Adaptive Critic and Dynamic Programming

In dynamic programming, the user supplies a utility function[2], $U(\underline{X})$, which is a function of the vector $\underline{X}$ ($\underline{X}$ describes the state of the external environment). The user then solves an equation, the Bellman equation, which yields another function $J^*(\underline{X})$. This second function, J^*, has the following property: by maximizing J^* in the immediate future (i.e. t+1), you automatically pick the strategy of action which maximizes the sum of U over the long-term future.

[1] The views expressed here are personal views of the authors, not those of NSF, nor SYSCON.

[2] In our notation, vectors are underlined, scalars are not. Thus $U(\underline{X})$ is a function, U, of a vector, $\underline{X}$, that returns a scalar.

Heuristic Dynamic Programming (HDP) was first formulated as a neural-network approximation to dynamic programming in Werbos (1977). We have implemented Action Dependent Heuristic Dynamic Programming (ADHDP), where the costs are on the actions (or transitions), not on the states (as in standard HDP). ADHDP is particularly good for dynamic systems where costs are incurred by taking actions, instead of being intrinsic to a state. We start from the following simplification of Howard's (1960) version of the Bellman equation for an absorbing Markov chain, similar to the simplification found in Werbos (1989a):

$$J^{*}(\underline{X}(t)) = \min_{\underline{u}(t)} E\{ U(\underline{X}(t),\underline{u}(t)) + J^{*}(\underline{X}(t+1)) \} \qquad (1)$$

where $\underline{u}(t)$ is a vector representing the choice of actions at time t, where E{} denotes the expected value, and where $\underline{X}(t+1)$ depends -- of course -- on the choice of action $\underline{u}(t)$. In our automaton, the action $\underline{u}(t)$ has a direct cost, U; thus we seek to minimize the sum of U over time. Note that U depends on both $\underline{X}(t)$ and $\underline{u}(t)$. Both $\underline{X}(t+1)$ and U may also be affected by random noise.

For our purposes, it is more convenient to use a critic that estimates a different function, J', defined as follows:

$$J'(\underline{X}(t), \underline{u}(t)) = E\{ U(\underline{X}(t), \underline{u}(t)) + J^{*}(\underline{X}(t+1)) \} \qquad (2)$$

Combining this with into equation 1, we arrive at the following recursive equation:

$$J'(\underline{X}(t), \underline{u}(t)) = E\{ U(\underline{X}(t), \underline{u}(t)) + \min_{\underline{u}(t+1)} J'(\underline{X}(t+1), \underline{u}(t+1)) \} \qquad (3)$$

ADHDP, as implemented for this automaton, is defined as adapting a critic network whose output is an approximation of J' (equation 3). In the automaton, the choice of actions $\underline{u}(t)$ was limited; we worked with only eight possible moves (like a King's moves in chess). Therefore, we chose to represent $J'(\underline{X}(t),\underline{u}(t))$ by the vector $\underline{J}'(\underline{X}(t))$, where $\underline{J}'$ has eight components corresponding to the eight possible moves.

As the system explores the state space, information required to make more accurate estimates of equation 3 will accumulate. Sutton (1988) and Werbos (1989c) have proven theorems about the consistency and convergence of very similar adaptive critic networks.

The Associative Memory

We have used an prototype based associative memory for our critic network, which was inspired by the work of Kanerva (1988) and of Albus (1981), though analogies to Kohonen (1988) and others are possible. Like a production rule, each prototype θ has a left hand side ($\underline{L}_\theta$), that corresponds to the conditional, and a right hand side ($\underline{R}_\theta$) that corresponds to the result. The two primary operations with the associative memory are reading from the memory and writing to, or updating, the memory.

We read from the memory by presenting an input pattern ($\underline{L}$) to the left hand side. The prototypes that will participate in the retrieval are selected by equation 4 (there are N^L elements in $\underline{L}$). This equation determines the strength of each prototype with respect to the current input pattern. The result, $\underline{R}$, of a read operation is just the weighted sum of the right hand sides (equation 5). The constant C is the threshold that selects which prototypes will be used in storing or retrieving a pattern.

We update the associative memory by presenting both an input pattern ($\underline{L}$) and a target pattern ($\underline{R}^T$). First we do a read (as above) to determine $\underline{R}$. Then we calculate the new right hand side for each prototype by equation 6.

$$\Phi_\theta = \begin{cases} 1 & \{ i;\ 1 \le i \le N^L \} \qquad | L_i - L_{\theta i} | \le C \\ 0 & \text{otherwise} \end{cases} \qquad \text{(4 - strength)}$$

$$\underline{R} = \Sigma\, \underline{R}_\theta \Phi_\theta \qquad \text{(5 - retrieval)}$$

$$\underline{R}_\theta = \underline{R}_\theta + \frac{(\underline{R}^T - \underline{R})\,\Phi_\theta}{\Sigma\,\Phi_\theta} \qquad \text{(6 - update)}$$

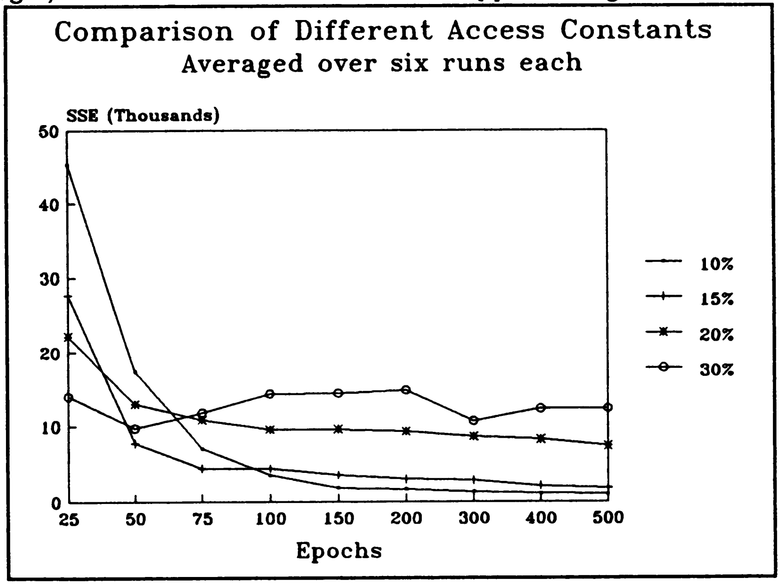
The associative memory is structured as follows:

$$\underline{X}(t) \to \underline{U}(t), \underline{J}(t+1) \qquad (7)$$

with $\underline{X}(t)$ on the left hand side, and $\underline{U}(t)$ and $\underline{J}(t+1)$ on the right hand side. Please note that for our experiments $\underline{X}$ is a vector of two components, <x,y>, that locate the automaton on the two dimensional grid. $\underline{J}$ and $\underline{U}$ are both vectors of eight components, (e.g. <j1,j2,j3,j4,j5,j6,j7,j8>), each element of which corresponds to one of the eight different moves. When the associative memory is initialized, the left hand side is a uniform distribution of the possible values of $\underline{X} \pm C$ (to account for edge effects). $\underline{J}$ and $\underline{U}$ are initilized to small, non-negative, random values.

Experimental Results

In our experiments, the associative memory has 800 prototypes. We vary the access constant, C, (see equation 4) so that the associative memory reads from and writes to 10, 15, 20 and 30 percent of the prototypes. The automaton environment is a 10 X 10 two dimensional grid. An epoch is defined as a walk from a randomly selected cell on the edge of the grid to the goal state at the center of the grid. A minimum of five steps is required for each epoch. Each step can be in one of eight directions and incurs a local cost equal to ten times the euclidean distance between the two states. Actions off the edge of the grid are legal, and take the automation to the opposite edge.



The automaton tries to discover the lowest cost path to the goal from each edge state. It does this by learning accurate estimates for the minimum sum of future costs (J') associated with alternative actions from each state. The automaton is given no model of: direction; the relative distance between two states; the relative differences or similarities between alternative actions; and the relative position of the goal from its current state. The automation is also unable to store the sequence of states it has visited. It is, therefore, unable to recall past sequences it has tried, identify a previously visited state, or recall the past history of a specific state. The automaton is only able to read from the current state. It uses the results of that read to select one of eight moves and update the estimate J' for the previous state. While the problem is simple for a human with a well developed model of direction and distance, it presents a very difficult machine learning problem; the automaton must induce a model based only on local cost information in forward time.

The results (see graph) illustrate the advantage of greater distribution across prototypes (generality) during the initial phase of learning, and the advantage of less distribution (specificity) during the final phase of learning. In addition, the performance of the more distributed memories was degraded by an inability to escape local optima. Given the associative memory and action selection rule, local optima are formed whenever the estimate of $J'(\underline{X}(t),\underline{u}(t))$ for the optimal action(s) is falsely higher than the correct estimate of $J'(\underline{X}(t),\underline{u}(t))$ for all nonoptimal actions. In this case, developing more accurate estimates of the nonoptimal actions for a state will not lead to the exploration of an optimal action. Bad estimates will then be filtered back to states on paths which must pass through these suboptimal states on their way to the goal.

Because of the radial nature of the problem, local optima were particularly hard to avoid near the goal. Neighboring cells toward the edge of the grid have a far greater similarity in J' values for the same action than do neighboring cells at the center (around the goal). Generalizing across increasingly different distributions of J' makes the memory more susceptible to being trapped in local optima. Current research is exploring a variety of mechanisms for dynamically altering the access constant and modifying the distribution of prototypes over the state space to overcome these problems.

References

Albus, J. S. (1981). *Brains, Behavior, and Robotics.* Peterborough, NH: Byte Books.

Anderson, C.W. (1987). Strategy Learning with multilayer connectionist representations. In *Proceedings* of the Fourth International Workshop on Machine Learning., p.103-114. Irvine, CA: Morgan Kaufmann.

Barto, A.G., R.S. Sutton, and C.W. Anderson. (1983). Neuron-like adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics* 13: 834-846.

Booker, L.B. (1988). Classifier Systems that Learn Internal World Models. In *Machine Learning* 3: 161-192.

Grossberg, S. (1982). How the Brain Builds a Cognitive Code. *Studies of Mind and Brain.* Drodrecht, Holland: Reidel Press.

Grossberg, S. (1988). Some Psychophysiological and Pharmacological Correlates of a Developmental, Cognitive, and Motivational Theory. *The Adaptive Brain*, part one. Netherlands: Elseview Science Publishers B.V.

Holland, J.H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor: The University of Michigan Press.

Howard, R. (1960). *Dynamic Programming and Markov Processes.* Cambridge: The MIT Press.

Jordan, M.I. (1989) Generic Constraints on Under Specified Target Trajectories. In *Proceedings* of the International Joint Conference on Neural Networks (IEEE, June).

Kanerva, P. (1988). *Sparse Distributed Memory.* Cambridge: MIT Press.

Kawato, M. (1989). Computational Schemes and Neural Network Models for formation of multijoint arm trajectory. In MSW, *op. cit.*

Kohonen, T. (1988). *Self-organization and associative memory.* New York: Springer-Verlag.

Klopf, A. H. (1982). *The Hedonistic Neuron: A Theory of Memory, Learning, and Intelligence.* Washington, DC: Hemisphere.

Lakoff, G. (1987). *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind.* Chicago: Chicago Press.

Nguyen, D., Widrow, B. (1989). Truck backer-upper: an example of self-learning in neural networks. In MSW, *op. cit.*

Sutton, R.S. (1988). Learning to predict by the methods of temporal differences. In *Machine Learning* 3: 9-44.

Werbos (1977). Advanced Forecasting Methods for Global Crisis Warning and Models of Intelligence. *General Systems Yearbook.* (Appendix B).

Werbos (1989a). A Menu of Designs for Reinforcement Learning Over Time. In *Neural Networks for Robotics and Control.* Editors: Miller, W. T., Sutton, R. S., Werbos. Cambridge: MIT Press.

Werbos (1989b). Backpropagation and neurocontrol: a review and prospectus. In *Proceedings* of the International Joint Conference on Neural Networks (IEEE,June).

Werbos (1989c). The consistency of HDP applied to a simple reinforcement learning problem. In *Neural Networks*, forthcoming.

Werbos (1989d). Maximizing long-term gas industry profits in two minutes in Lotus using neural network methods. *IEEE Transactions Systems Man and Cybernetics.* March/April.

Williams, R.J. (1988). *Toward a theory of reinforcement-learning connectionist systems.* Tech. Rep. NU-CCS-88-3, College of Computer Science, Northeastern University, Boston, MA.

The Real-Time Classification of Temporal Sequences with an Adaptive Resonance Circuit

Albert L. Nigrin*
Duke University Computer Science Department
109 North Building, Duke University, Durham, NC 27706
aln@cs.duke.edu

1 Introduction

Adaptive Resonance Theory has been used to explain many psychological experiments [2,3,4]. It has also been used to construct networks (see ART1 and ART2 in [4, ch6][1]) that learn to classify binary and continuous spatial patterns. In this paper, we present a network that uses the framework of Adaptive Resonance Theory to perform the stable classification of temporal patterns in real-time. [5]

The problem is as follows: How can a network discover the significant chunks that are present in a temporal pattern of inputs? For example, humans learn to segment sentences of continuous speech into distinct words, even without any clear breaks between words.

In order to deal only with temporal considerations, we have abstracted the problem as follows: Instead of presenting the network a slowly fluctuating continuous pattern, we present a sequence of items at a constant rate. The network must discover the longest sequences that occur repeatedly.

Consider the following sentences. The letters were presented to our network, one at a time (even across sentence boundaries) and the sentences were repeated at random.

```
FBIABCDNOEG
HJKFBILMPNO
QFBINOSTUV
```

The lists FBI and NO occur in several contexts. After fewer than 10 presentations, the network learned to recognize these sequences as significant chunks.

The network classifies lists quickly, even when the number of different lists to be classified is large. Extensive simulations have shown that the number of training set iterations needed to classify multiple different lists, rises at a slower than linear rate when compared with the number of lists.

Higher level structure has also been learned. (The above example does not contain enough context to perform a correct generalization, however.) By increasing the number of layers in the network, it is possible to learn lists of lists, etc. Simulations show that the training time needed to learn these higher level classifications rises linearly with the number of levels in the network.

To provide a a flexible learning system our network satisfies the following criteria.

1. All STM and LTM interactions occur in real-time. The continual presentation of items eliminates the possibility of either off-line processing or long equilibration times.
2. The network solves the stability-plasticity tradeoff. The formation of new categories does not erode previous classifications made by the network. Furthermore, until the capacity of the network is exceeded, the network retains the ability to form new classifications, in response to new patterns.
3. The model performs both fast and slow learning. Fast learning provides classification in as quickly as 1 trial, while slow learning allows the model to generalize across multiple different examples.

Before the network can begin to classify an item sequence, it must transform the sequence into a spatial pattern of activity. The architecture that automatically performs this transformation is discussed in section 2. Section 3 presents a two layer network that can self organize in response to single list presentations. Section 4 showns how a network (possibly cascaded to learn higher level abstractions) can learn to classify lists, even when they are embedded in longer lists (sentences).

*This research was supported in part by ARO under grant DAAG-29-84-K-0072.

2 Transforming temporal patterns into a spatial patterns

This section demonstrates how to transform sequences of items into spatial patterns. We use a diminishing pattern of activity across network cells to reflect the order in which the items were presented. Figure 1 shows the activity pattern across network cells, after the presentation of various lists.

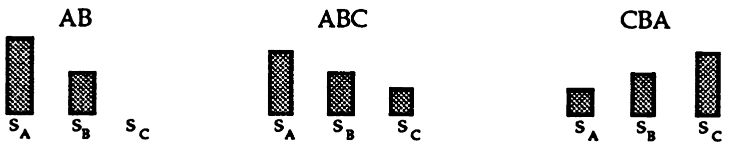


Figure 1: Activity pattern across cells s_A, s_B, and s_C after presentation of lists AB, ABC and CBA.

Assume there are n different items $v_1, v_2, \ldots, v_n$. Item v_i is presented to the network, by supplying a constant level of input to cell s_i for a fixed time period. The transformation from an item sequence to a diminishing activity pattern is obtained through the use of a feedback on-center off-surround architecture. Each cell obeys the activation equation:

$$\frac{d}{dt}s_i = -As_i + (B - s_i)[f(s_i) + I_i] - s_i[\sum_{j \neq i} f(s_j)] \tag{1}$$

where s_i is the activity of the i^{th} cell in the architecture (The same symbol refers to both to the name and activity of a cell), A is a constant representing passive decay, B is a constant representing the maximum activity of each cell, I_i is external input and $f(s)$ is a sigmoid function that is linear in the cell's desired range of activity. It was proved that upon removal of external input, the relative activity of the cells remains constant (if their activities are in the linear range of the function f) [2, ch8].

The diminishing activity pattern is obtained as follows: The presentation of item v_i causes s_i to reach an activity level, much less than B, that is relatively independent of the number of items previously presented. Then, during the presentation of successive items, s_i's activity rises due to the positive feedback term $(B - s_i)f(s_i)$ (until the total activity of the field saturates). Earlier items are represented by larger activities since the activity of their respective cells rises for longer periods of time.

The total activity in the field can fluctuate within a large range and is entirely dependent on the number of items that have been presented. To eliminate this dependency, we normalize the length of the vector representing the activity of the s field. Normalization is accomplished with a feedforward on-center off-surround architecture, composed of t cells, where each t cell obeys:

$$\frac{d}{dt}t_i = (1 - t_i^2)[f(s_i)^2] - t_i^2[\sum_{j \neq i} f(s_j)^2] \tag{2}$$

3 Two layer Network for classifying non-embedded lists

In this section, we will show how to classify lists when they are presented one at a time. The network that performs this classification is composed of two fields, $F^{(1)}$ and $F^{(2)}$. $F^{(1)}$ transforms the incoming temporal pattern into a spatial pattern, using the architecture of the previous section (shown in figure 2a). $F^{(2)}$ classifies the spatial patterns across $F^{(1)}$, into different categories.

The $F^{(2)}$ field is composed of many repeated copies of the circuit shown in figure 2b. Each copy of the circuit (a cell packet) represents one list. Bottom up input enters $F^{(2)}$ at the b cells, where it is gated by LTM weights. This gated input is then sent to the competition level composed of c cells (The activity equation for the c cells is of the same general form as equation 1 except f is instantiated to produce winner-take-all dynamics.). Finally the output of the b and c cells are combined multiplicatively at the e cell, which sends inhibitory signals to other c cells in $F^{(2)}$.

$F^{(2)}$ cells learn to classify patterns by modifying their LTM weights to become parallel to the $F^{(1)}$ activity pattern they are coding. Furthermore, $F^{(2)}$ cells automatically change their threshold and gain functions to reflect the size of the pattern they are coding (equations not shown).

Changes in the LTM weights at the b cells are driven by the activity of the c cell ([5][4, ch7]), as shown in figure 2b. The LTM learning law is a variation of the following:

$$\frac{d}{dt} z_{ji} = \epsilon \, (c_i^{(2)})^2 \, (-z_{ji} + t_j^{(1)} c_i^{(2)}) \tag{3}$$

where $x_i^{(j)}$ refers to the activity of the i^{th} cell of type x at level $F^{(j)}$, ϵ is the learning rate and z_{ji} is the LTM weight from $t_j^{(1)}$ to $b_i^{(2)}$.

The output of the b cell is not a straight linear sum as in $I_i^+ = \sum t_j^{(1)} z_{ji}$. Otherwise later items would have an increasingly small influence on the output of the cell. Consider the case where the pattern $t_j = 2t_{j+1}$ exists across (t_1, t_2, t_3, t_4), in response to the presentation of a 4 item list. To classify this pattern, the LTM weights on some cell, $b_i^{(2)}$, are modified so that $z_{ji} = 2z_{j+1,i}$. When the full list is later presented, the contribution from the last item is only 1.1% of the total output of $b_i^{(2)}$ ($t_4^{(1)} z_{4i} = .01 \times \sum_{j=1}^{4} t_j^{(1)} z_{ji}$). The presence or absence of the last item has a negligible effect on the value of I_i^+.

To remedy this, we multiply the gated sum I_i^+ by an efficacy term $I_i^\times$ when computing the output of $b_i^{(2)}$ ($b_i^{(2)} = I_i^+ I_i^\times$). $I_i^\times$ increases as a cell's inputs better match its LTM weights and is computed by an equation similar to: $I_i^\times = \prod (C_1 + C_2 \, min(1, \frac{Z_{ji}}{T_j}))$ where $Z_{ji} = z_{ji} / \sum_{\forall k} z_{ki}$, $T_j = t_j^{(1)} / \sum_{\forall k} t_k^{(1)}$, and C_1 and C_2 are constants. Since the $I^\times$ term can vary greatly, depending on the number of inputs that activate a cell, the output from the b cell must be normalized before it affects the c cell (This is done by a scheme that is not shown here).

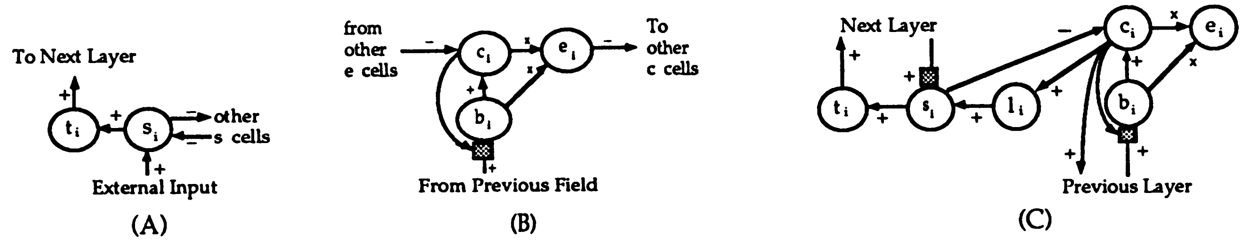


Figure 2: (a) Network to transform temporal patterns to spatial ones. (b) Network to classify spatial patterns of the previous field. LTM connections are indicated by shaded boxes. All other connections are non-modifiable (c) Combining a) and b) to create a circuit that both classifies patterns and maintains the temporal order of the classification. The l cell acts as a latch that activates for a brief period once its c cell exceeds threshold. Slight modifications allow feedback to provide context for lower level classifications[5].

We will now discuss stability of the LTM weights. In ART1 and ART2, a mechanism was designed to compare the $F^{(2)}$ feedback pattern at $F^{(1)}$, with the activity pattern at $F^{(1)}$. If these patterns differed sufficiently, the active $F^{(2)}$ packet was reset[4, ch6][1]. Our mechanism differs for the following reasons: (1) If a new pattern is similar to many previously classified patterns, the network will reset many times before trying to form a new category. This is not acceptable in a real time environment. (2) When lists are embedded in sentences (as in the next section), erroneous mismatch will occur if additional items are presented before the list is fully classified (unless stability is sacrificed by allowing looser mismatch criteria). (3) By shutting off the offending $F^{(2)}$ packet, information is lost about the content of the input.

Our mechanism solves the stability plasticity tradeoff in a manner similar to the mechanism behind Masking Fields[4, ch7][5]. Our network implements the following guidelines: (1) A packet representing some pattern, x, can emit enough inhibition to prevent other packets from classifying patterns of size less or equal than x, but cannot emit enough inhibition to prevent other packets from classifying patterns of size greater than x. (2) When incoming input does not match the LTM weights at a cell, learning at that cell ceases, and lateral inhibition is reduced, to allow other packets to code the pattern.

For example suppose b_i has coded ABC. If $ABCD$ is later presented, l_i can not send enough lateral inhibition to stop other cells from learning $ABCD$ (by guideline 1). If AB or CBA is presented, the inhibition emitted by l_i is reduced sufficiently to once again allow other cells to learn these patterns (by guideline 2).

4 Multilayer network for classifying embedded lists

In this section, we show how to cascade fields to provide higher level concepts. The fields are constructed by combining the circuits shown in figure 2a and 2b to form the circuit in 2c. The same homologous circuit is used at each level of a hierarchy. The type of information represented by a cell packet depends only on the field where it is located. For example, packets at $F^{(1)}$ represent items, packets at $F^{(2)}$ represent lists, packets at $F^{(3)}$ represent lists of lists, etc.

Modifiable feedback connections are added from $F^{(i)}$ to $F^{(i-1)}$ and feedback learning is accomplished via outstar learning, as in ART1 and ART2 [1,4]. The bottom up and top down LTM weights become roughly symmetric. Thus, those $s^{(i-1)}$ cells that provide a large amount of input to a particular $F^{(i)}$ packet receive large amounts of feedback when that $F^{(i)}$ packet is active.

Information is processed in the following way. Patterns across t cells in $F^{(i-1)}$ activate $F^{(i)}$ cells, where competition occurs at the c layer. The competing c cells send feedback to only those portions of the $F^{(i-1)}$ pattern that are being considered for classification. Once the activity of a cell $c_j^{(i)}$ exceeds some threshold, classification is considered to have occurred, setting off the following:

1. Cell $s_j^{(i)}$ is activated and maintains temporal information as shown in section 2. This allows cells at $F^{(i+1)}$ to classify patterns across $F^{(i)}$.
2. Cell $c_j^{(i)}$ is shut off by inhibition from cell $s_j^{(i)}$.
3. Once $c_j^{(i)}$ is shut off, lateral signals from e_j no longer inhibit other c cells. This allows other packets at $F^{(i)}$ to classify the remaining (or evolving) pattern at $F^{(i-1)}$.
4. Cell $c_j^{(i)}$ no longer sends feedback to $F^{(i-1)}$. Using the mechanism of the gated dipole[2,3,4,5], s cells at $F^{(i-1)}$ that previously received large feedback signals from $c_j^{(i)}$, turn off once feedback ceases. Only those portions of the $F^{(i-1)}$ activity pattern not yet classified (that consequently never received large feedback from $F^{(i)}$) remain active. Other $F^{(i)}$ packets compete to classify this remaining pattern.

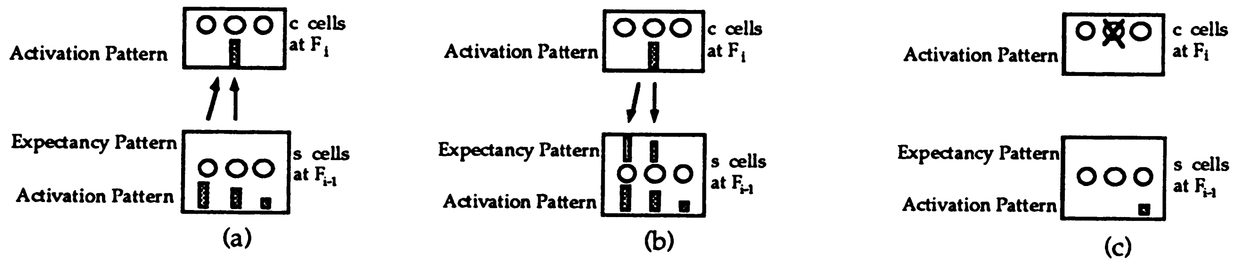


Figure 3: (a) Signals from $F^{(i-1)}$ activate packets at $F^{(i)}$. (b) Active $F^{(i)}$ cells emit large feedback to $F^{(i-1)}$. (c) $s^{(i-1)}$ cells that are part of a classified pattern are reset when feedback from $F^{(i)}$ is abruptly terminated.

References

[1] Carpenter, G.A. and Grossberg, S. "ART 2: Self-organization of stable category recognition codes for analog input patterns", *Applied Optics*, vol. 26 pp. 4919-4930, Dec. 1987.

[2] Grossberg, S. *Studies of Mind and Brain: Neural Principles of Learning, Perception, Development, Cognition and Motor Control.* Boston: Reidel Press, 1982.

[3] Grossberg, S. *The Adaptive Brain* vol. 1 & 2, North-Holland: Elsevier Science Publishing, 1987.

[4] Grossberg, S. *Neural Networks and Natural Intelligence.* Cambridge, MA: MIT Press, 1988.

[5] Nigrin, A. *The Real-Time Classification of Temporal Patterns with an Adaptive Resonance Circuit.* Ph.D Thesis, In preparation.

A Neural Model of Interpolation or Interpolation with Blobs

Alexander Shustorovich
Advanced Computing Laboratory,
Eastman Kodak Company,
Rochester, New York 14653-5815

1. Introduction.

When we speak about interpolation we usually mean interpolation with splines. We are in a "piecewise polynomial" paradigm. There are good reasons for that: "One uses polynomials for approximation because they can be evaluated, differentiated, and integrated easily and in finitely many steps using just the basic arithmetic operations of addition, subtraction and multiplication." ([1], p. 1). Also very important is the fundamental property that "... the complete cubic spline interpolant ... minimizes $\int_a^b [f''(x)]^2 dx$ over all twice differentiable functions f which agree with g at $\tau_0, \cdots, \tau_{n+1}$." ([1], p. 63). In addition to that, the elegant mathematical theory of B-splines is very well developed and computationally relatively cheap.

There are, though, some problems with splines, and the most important among them is the fact that some piecewise polynomial interpolants "do not look right". Sometimes what people would draw is very different from the computer-provided curve. This happens because people *do not* minimize $\int [f'']^2$ when they draw curves through data points, and also they do not react to jumps in the second derivative unless it changes its sign - concave segments are easy to distinguish from convex ones, but humans cannot do much more than that. The deficiences of the standard technique and the interest in how humans perform the task motivate the attempts to use the neural modeling approach to interpolation and curve fitting.

2. Sigmoids and Blobs.

Usually individual neuron response is modeled with some kind of sigmoid function. By far the most common among them is the logistic curve $S(x)=\frac{1}{1+e^{-x/w}}$. It is easy to see that it is zero for large negative values of x, increases as x increases, $S(0)=1/2$, and $S(x)$ becomes equal to one for large positive values of x. Also, it is symmetric, $S(-t)=1-S(t)$, and it is smooth with the derivative monotonically increasing from zero to its maximum at $x=0$ and monotonically decreasing to zero for larger values of x. Later in this text such functions will be referred to as sigmoids.

It is possible to combine two sigmoids (actually two copies of the same function) to model bell-shaped sensor responses. Consider $b(x)=S(x+1/2)-S(x-1/2)$. This function increases from zero to its maximum $b(0)=S(1/2)-S(-1/2)$, and then symmetrically decreases back to zero. If we model evenly distributed sensors, this means that two neighboring neurons should combine their individual responses to provide a blob whose maximum is at a point halfway between them.

In this Section and the next we shall discuss only the case of uniform knot sequences, that is, $x_i=i$ for all integers from some interval. For the beginning let us consider only very steep sigmoids equal (or approximately equal) to zero for $x \le -1/2$, rising on $[-1/2, 1/2]$ and approximating one for $x \ge 1/2$. With this condition $b(x)=S(x+1/2)-S(x-1/2)$ is nonzero only on $[-1,1]$.

Let us denote $b_i(x)=b(x-i)$. This blob is a translated copy of $b_0(x)=b(x)$. Also, we will need sigmoids w times dilated, $S_w(x)=S(x/w)$. These sigmoids rise on $[-w,w]$, and the corresponding blobs $b^w(x)=S_w(x+1/2)-S_w(x-1/2)$ are nonzero only on $[-w-1/2, w+1/2]$. The blob support has the length $2w+1$. Again, $b_i^w(x)=b^w(x-i)$ is a translated copy of the $b^w(x)$ and its maximum is now at $x=i$ instead of $x=0$. Interpolation with blobs means that, given an arbitrary (smooth) function $g(x)$, we want to construct a sum $B(x)=\sum_i c_i b_i^w(x)$ which agrees with $g(x)$ at points $\{x_i\}$, that is, at integer points in our case.

The first property of blobs to mention is that if we add them together, their sum is equal to one:

$$\sum_{k=-N}^{N} b_k^w(x)=\sum_{k=-N}^{N}(S_w(x-k+1/2)-S_w(x-k-1/2))=S_w(x+N+1/2)-S_w(x-N-1/2)$$

and for any value of w for sufficiently large values of N, this sum converges to 1. For any x the first value becomes equal to one and the second to zero as N increases. If we are interested only in the interval $[0,m]$, it is enough to consider only blobs with their supports intersecting $[0,m]$, that is, $b_i^w(x)$ for $-w-1/2 \le i \le w+1/2$.

The consequence: the coefficients $\{c_i=1\}$ provide us with $\sum_{i=0}^{m} c_i b_i(x_j)=1$ for all j.

In mathematical terminology the sequence of translated blobs provides a *partition of unity*. The same property holds for B-splines, but there is one very important difference between these two constructs: it is impossible to dilate B-splines and blobs directly (the partition of unity will be ruined), but it is very easy to dilate corresponding sigmoids *before* they are combined to produce blobs. We shall use this flexibility in Section 5.

It was proved in [2] that any bell-shaped function providing the partition of unity can be represented as a blob, that is, as a difference of two shifted copies of an appropriate sigmoid. That is why all smooth B-splines can be considered particular cases of blobs.

The next problem we shall try to solve with blobs is the approximation of the straight line $g(x)=x$. It is especially easy if the support width is equal to 2; as $b_i(i)=1$ and $b_i(j)=0$ for all $i \ne j$, it follows that the coefficients should be equal to blob indexes, $c_i=i$. But if we look at the graph of this "approximation", it is completely unsatisfactory; the resulting curve goes along the straight line in waves.

The way to deal with this difficulty is to use wider blobs. If we use blobs made of dilated sigmoids, the approximation converges to the linear function, and this effect does not depend on the choice of any particular sigmoid. Figure 1 represents a sequence of approximations for w=0.5, w=0.7, w=1.5, and w=2.5. The last *curve* (w=2.5) in Fig. 1 is impossible to distinguish from the actual straight line, and the support width necessary for this level of approximation is only 6. It is important to keep in mind that *all* finite-support and also standard logistic sigmoids provide *absolutely* the same effect in simulations. The formal proof of this invariance can be found in [2].

3. Mathematical Formulation of the Problem.

We need to find $\{c_i\}$, $i=0, \cdots, m$ so that $\sum_{i=0}^{m} c_i b_i(j)=g_j$ for all integer points j, $0 \le j \le m$.

Let us denote $p_0=b(0)$, $p_1=b(1)$, $\cdots$. For our example from the previous Section with w=2.5, we have $p_3=p_4=\cdots=0$. We have to calculate the values of the sum at $m+1$ integer points.

As we do not consider blobs to the left of zero and to the right of m, $B(0)=c_0p_0+c_1p_1+c_2p_2$, where p_0 is the input of $b_0(x)$, p_1 - of $b_1(x)$, p_2 - of $b_2(x)$. Also, $B(1)=c_0p_1+c_1p_0+c_2p_1+c_3p_2$. Here, p_1 is the input of $b_0(x)$ coming from the left of x=1; p_0, of $b_1(x)$ itself; p_1, of $b_2(x)$; and p_2, of $b_3(x)$. All the $B(j)$ are computed in this manner, and as $B(j)$ should be equal to g_j, we have a system of m+1 equations with m+1 unknown coefficients c_i. In matrix notation $G=[P]\times C$, where C is the vector of the coefficients, G is the vector of the function values and the matrix $[P]$ consists of shifted lines $\{p_2, p_1, p_0, p_1, p_2\}$ with p_0 along the main diagonal and zeroes instead of p_j for $j \ge 3$, which is the result of the choice of our blob width. Matrices of this structure are called the Töplitz matrices and they can be inverted. As we can calculate $C=[P]^{-1}\times G$, the solution is unique.

It is necessary to mention that the problems of end effects are very much the same as with splines. To produce a nice-looking approximation on $[0,m]$ we have to provide some information (or guesses) about the behavior of the modeled function g for several points to the left and to the right. The symmetrization $g_{-i}=2g_0-g_i$ and $g_{m+i}=2g_m-g_{m-i}$ works well enough.

4. Interpolation Model for Non-Uniform Knot Sequences.

Up to this point our model was very simple; it used uniformly translated copies of the same blob. But what happens if the knot sequence is not uniform? It seems possible to stretch the analogy to splines, but after several days of making my colleagues draw curves through simulated data points, I became certain that people do not model the function itself, they model its derivative. This is the approach discussed in this Section.

Let us denote $x_0, x_1, \cdots, x_m$ as the increasing sequence of x-coordinates and $g_0, g_1, \cdots, g_m$ as the corresponding sequence of y-coordinates of the data to be interpolated. Let us denote also $\Delta_i=\dfrac{g_i-g_{i-1}}{x_i-x_{i-1}}$, $i=1$,

$\cdots, m$. We are interested in interpolating the values of Δ_i. After that we will integrate the resulting curve to fit the set of $\{x_i, g_i\}$.

At first, for the sake of simplicity, let us consider only $x_i = i$, but now we will use blobs "hanging" over midpoints, $b_i(x) = b(x-i+1/2)$. As before, we want to find $\{c_i\}$, $i=1, \cdots, m$, such that for $D(x) = \sum_{i=1}^{m} c_i b_i(x)$ every $\int_{x_{i-1}}^{x_i} D(x)dx$ equals Δ_i for all $i=1, \cdots, m$. Let us denote $F_i = \int_{i-1/2}^{i+1/2} b(x)dx$. It is easy to see that (as $x_i = i$)

$$\int_0^1 D(x)dx = c_1 \int_0^1 b_1(x)dx + c_2 \int_0^1 b_2(x)dx + \cdots + c_m \int_0^1 b_m(x)dx.$$

As $\int_0^1 b_i(x)dx = \int_{i-1}^{i} b_1(x)dx$, we conclude that $\int_0^1 D(x)dx = c_1F_0 + c_2F_1 + \cdots + c_m F_{m-1} = \Delta_1$. Also,
$\int_1^2 D(x)dx = c_1 \int_1^2 b_1(x)dx + c_2 \int_1^2 b_2(x)dx + \cdots + c_m \int_1^2 b_m(x)dx = c_1F_1 + c_2F_0 + c_3F_1 + c_4F_2 + \cdots + c_m F_{m-2} = \Delta_2$, etc. Finally, we have a system of m equations with m unknown coefficients $\{c_i\}$. In matrix notation it is $[F] \times C = \Delta$. The matrix $[F]$ is symmetrical and even for the logistic sigmoid all the $F_6, F_7, \cdots$ are virtually zeroes. The inverse matrix always exists and the solution to our problem is $C = [F]^{-1} \times \Delta$. All we have to do after this is to calculate the modeled function values from the necessary level $g(0) = g_0$ with the help of integration, that is, just adding the values of the obtained model for the derivative.

For non-uniform knot sequences let us consider scale factors $s_i = \frac{1}{x_i - x_{i-1}}$, and let us use the piecewise linear function $x_i = \xi(i)$, mapping $0, 1, \cdots, m$ into $x_0, x_1, \cdots, x_m$. This function is continuous though it is not smooth. If we consider $g(x_i) = g(\xi(i)) = (g \times \xi)_i$, we can apply the algorithm described above to model the derivative of the function $g \times \xi$, which differs from the piecewise derivative of g only by a factor of $(x_i - x_{i-1})$. That is why, after we evaluate the derivative of $g \times \xi$, all we have to do is to multiply it by s_i on each of the intervals $[x_{i-1}, x_i]$. Of course, this means that the model for the derivative of g will not be smooth, but as we integrate it to calculate the values of g itself, the model for g becomes smooth.

Sure enough, we have problems with end effects again and we need three or four extra points to the left and to the right. I calculated these additional values for the first difference using symmetry, $\Delta_{m+i} = 2\Delta_m - \Delta_{m-i}$, but of course this depends on the information about the function.

The inverse matrix $[F]^{-1}$ and the blob-values were calculated only once; the remaining arithmetic consisted only of addition, subtraction and multiplication. The calculations took the same time for all sets of pairs $\{(x_i, g_i)\}$. Computationally, interpolation with blobs is cheap.

In my simulations I worked with a menu of blobs produced by different sigmoids and also directly used the Gaussian which is not a blob-function, but becomes very blob-like if it is sufficiently wide. Corresponding variations in the exact shape of the interpolants were always negligible compared to the effect of changing the support width.

5. Interpolation Model with Variable Blob Widths.

Before this approach can be used in practice, we have to learn how to choose the blob width. If we insist that the monotonic curves should look nice, we have to use blobs with a support of not less than 4 (as for cubic splines). The exact behavior of these curves somewhat depends on the choice of the particular sigmoid, but for all practical purposes the variations are negligible. The problem is that if the sequence $\{g_i\}$ is not monotonic, the "inertia" of the wide blobs' cross-influence makes the interpolant overshoot compared with what people would draw. In this respect, unfortunately, blobs are similar to splines.

If we model the derivative instead of the function itself, the same difficulty manifests itself in a different way: the use of narrow blobs provides us with slightly smoothed piecewise linear interpolation, while using wide blobs results in very smooth (and overshooting) curves.

Humans are nonlinear in their approach to interpolation; given the same knots $\{x_i\}$ and two sets of y-values, $\{p_i\}$ and $\{q_i\}$, they would draw nice curves P and Q, but their curve for the set $\{p_i + q_i\}$ will be very different from the sum of the curves P and Q.

The obvious consequence of this nonlinearity is that linear procedures will never guarantee natural-looking results. That is why the best results so far were achieved with a modification of the derivative model in which the blob width to be used at any knot depended on the local behavior of the modeled function.

In the first version of the program I used a simple set of rules to adjust left and right half-widths separately: the basic support width was equal to 5, near all local extrema it was reduced to 4 or even 3 if the ratio of the derivatives was greater than 2; on monotonic segments it was reduced to 4 if the ratio of the derivatives was greater than 2, and the support width would become 2 at the points where the left or right derivative was equal to zero, or even 1 if the derivative was zero for two or more adjacent segments.

For this model the matrix $[F]$ is no longer symmetrical and depends on the data set. Nevertheless, it still has very few non-zero elements along the main diagonal and it can be inverted. One matrix inversion per data set does not seem to be too expensive. Figure 2 contains several examples of interpolants calculated with this model. Future improvements in quality of the interpolants are expected with the next version of the model with continuous support widths instead of the discreet.

6. Conclusions.

A neurally inspired approach to interpolation is proposed in this paper. Its characteristic feature is the introduction of blob-functions, that is, pairwise differences of two sigmoid curves. These blobs demonstrate very interesting properties which permit their use instead of B-splines. Three models of interpolation with blobs were discussed, and the existence and the uniqueness of the solutions proved. Corresponding algorithms were described that are computationally cheap and can be easily implemented in parallel architectures. On the whole, this approach provides an unusual perspective on interpolation, and fits many seemingly idiosyncratic properties of splines into a more general picture. We are still far from understanding the human interpolation process, but these first results seem to be very encouraging.

7. References.

[1] C. De Boor, *Practical Guide to Splines*, Applied Mathematical Sciences, Vol. 27, Springer-Verlag, New York, 1978.

[2] A. Shustorovich, *The Invariance of Overall Sensor Response to the Choice of Sigmoid Neural Response Curves*, Technical Report 245127E, Eastman KODAK Company, 1988. Poster at IJCNN-89, Washington D.C., June 1989, Proceedings, Vol. 2, p. 583.

8. Figures.

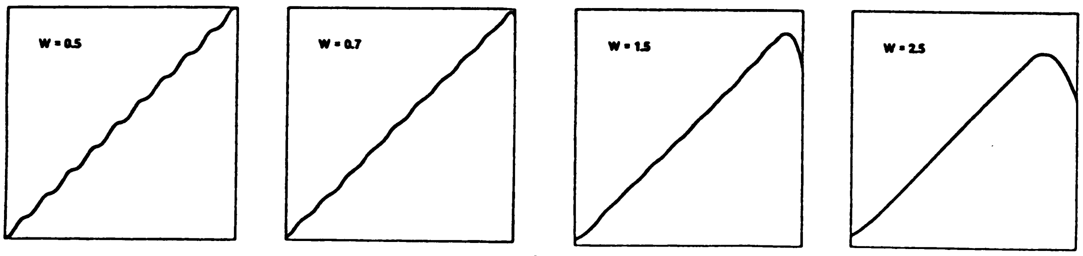


Fig. 1.

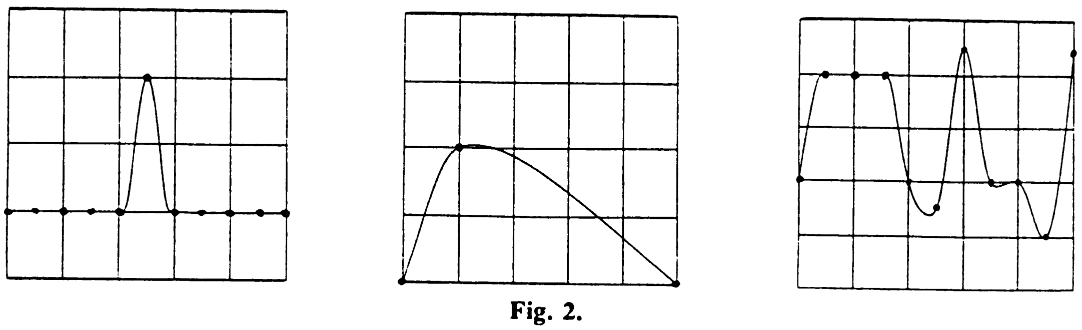


Fig. 2.

MRIII: A Robust Algorithm for Training Analog Neural Networks

David Andes
Naval Weapons Center, China Lake, CA 93555
Bernard Widrow Michael Lehr Eric Wan
Stanford University Department of Electrical Engineering, Stanford, CA 94305-4055

Abstract

Like many training algorithms for artificial neural networks, the backpropagation technique assumes complete a priori knowledge about both the network architecture and the transfer characteristics of the computing devices. This is reasonable if the network is to be constructed with floating-point hardware. If, however, the implementation is to be analog, often the assumed knowledge will not be available in any precise form. Thus, there is some need for a method which is analogous to backpropagation, but better suited for analog circuitry. In this paper we introduce Madaline Rule III (MRIII), a new training rule which serves this purpose. Like backpropagation, MRIII trains differentiable neural networks by steepest-descent. Consequently, when applied to simple feed-forward topologies with known characteristics, both algorithms achieve equivalent solutions. MRIII does not need prior knowledge about the network, so it is relatively immune to the effects of neuron-to-neuron variations and unknown or non-ideal component characteristics. Thus the new algorithm performs well when applied to analog neural networks, including networks comprised of unconventional components, and those with unusual or recurrent interconnections.

1 Introduction

MRIII is essentially a generalization of Madaline Rule II (MRII) [6, 7, 8] to allow the adaptation of networks built from Adalines with differentiable activation functions rather than hard limiting quantizers. Although it is an extension of MRII, the new rule is also closely related to the backpropagation technique, as both are supervised steepest-descent algorithms which iteratively adapt the weights of a neural network toward a local minimum in the mean-square-error surface. The difference between backpropagation and MRIII lies primarily in the methods used to determine the gradient estimates. Backpropagation uses *a priori* knowledge about the characteristics of the network's computing elements to calculate the gradient estimates directly. In contrast, MRIII training relies upon gradients which are determined by explicit measurements. This process involves simply perturbing the values of either the network's weights or node activation levels and measuring the ratio of the change in the network's instantaneous squared output error to this perturbation. Both backpropagation and MRIII determine the appropriate weight adjustments directly from the gradient estimates.

Because MRIII does not require prior knowledge about the network characteristics, it is an ideal method for training analog structures which do not have precisely known characteristics. In fact, David Andes and his research group at the Naval Weapons Center at China Lake, California have already developed a working analog system which uses the new algorithm. More recently, Intel, in collaboration with the Naval Weapons Center, has developed a commercial analog neurocomputing chip which can be trained by MRIII [1].

For our initial discussions about the algorithm, we consider only feed-forward networks. We also assume that all networks are composed of sigmoidal Adalines, that is, neurons that compute an output $y = f(\sum_{i=0}^{L} x_i w_i) = f(\mathbf{W}^T\mathbf{X}) = f(s)$. Here $\mathbf{W}$ is a weight vector, $\mathbf{X}$ is the corresponding input vector, and f is any differentiable nonlinear function, most often the sigmoid function, $f(s) = tanh(s)$. The binary Adaline, used with MRII systems, is shown in Figure 1a, and the sigmoidal Adaline used with MRIII is illustrated in Figure 1b. More general neuron types and network topologies will be discussed below in Section 3. In all networks we will assume that the weights are initially set to small random values.

2 Madaline Rule III

Like the MRII technique, MRIII can be interpreted from the standpoint of the minimal disturbance principle. In this paper, however, we develop a more natural interpretation of the algorithm in the context of gradient descent. There are two basic MRIII procedures, a node-by-node version and a weight-by-weight version. In this section we will discuss the former. The weight-by-weight variant is discussed below in Section 3.1.

The node-by-node version of the MRIII algorithm is a simple procedure. A training pattern is presented to the input of the network and a measurement is taken of the squared response error, $\epsilon_k^2 = |\mathbf{E}_k|^2 = |\mathbf{D}_k - \mathbf{Y}_k|^2$. Here, $\mathbf{Y}_k = [y_{1_k}, y_{2_k}, \ldots, y_{L_k}]^T$ is the vector of neuron outputs in the final layer, $\mathbf{D}_k = [d_{1_k}, d_{2_k}, \ldots, d_{L_k}]^T$ is the corresponding vector of desired responses, and $\mathbf{E}_k = \mathbf{D}_k - \mathbf{Y}_k = [e_{1_k}, e_{2_k}, \ldots, e_{I_k}]^T$ is the vector of response errors in the output layer. After this measurement, a small perturbation δ is added to the summing junction of one of the Adalines and the squared response error is again measured. From this, we calculate

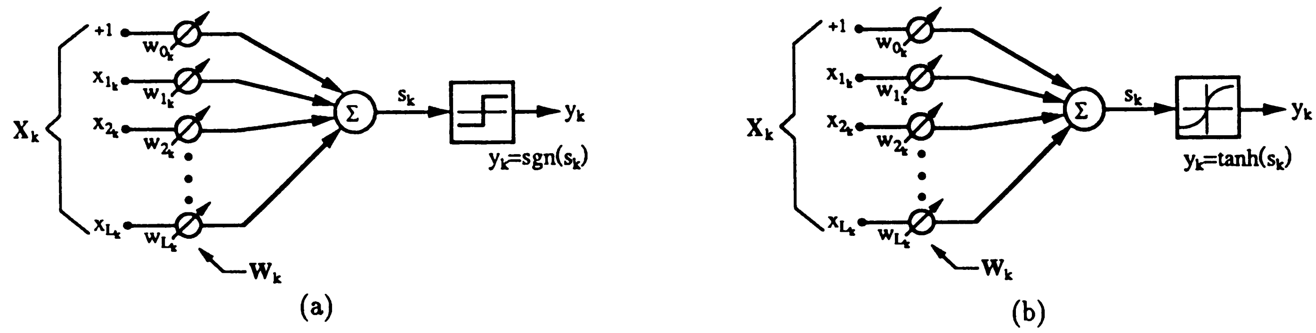


Figure 1: **Adalines** (a) Binary, (b) Sigmoidal

$\Delta\epsilon^2_{m_k} = \Delta|\mathbf{E}_{m_k}|^2 = |\mathbf{E}_{perturbed_{m_k}}|^2 - |\mathbf{E}_{unperturbed_{m_k}}|^2$, the square error fluctuation due to the perturbation of the mth neuron during training presentation k[1].

As shown below, $\Delta\epsilon^2_{m_k}$ can be used to determine the appropriate weight change for the selected neuron. After updating the weights of the perturbed neuron, another neuron is selected, perturbed, and updated in a like manner. We thus progress through the entire network, successively perturbing and adapting each neuron. After the final node has been adapted, we present a new binary pattern and repeat the procedure. As with most other stochastic training procedures, patterns are presented in a random, noncyclical order.

The appropriate weight updates are easily determined by appealing to the backpropagation technique. Backpropagation, as presented by Rumelhart *et al* [4] performs gradient descent on the mean-square-error surface in weight space by the rule $\mathbf{W}_{k+1} = \mathbf{W}_k - \mu\hat{\nabla}_k$, where, as before, $\hat{\nabla}_k$ is the error gradient estimate in weight space based only upon information extracted from the current training presentation. If we fix all weights in the network and adapt only those of neuron m by backprop, we find:

$$\mathbf{W}_{m_{k+1}} = \mathbf{W}_{m_k} - \mu\hat{\nabla}_k = \mathbf{W}_{m_k} - \mu\frac{\partial\epsilon_k^2}{\partial\mathbf{W}_{m_k}} = \mathbf{W}_{m_k} - \mu\frac{\partial\epsilon_k^2}{\partial s_{m_k}}\frac{\partial s_{m_k}}{\partial\mathbf{W}_{m_k}}. \tag{1}$$

We can effect backpropagation in an unusual manner by adding a small perturbation δ to the summing junction of neuron m, and observing the resulting change in network output error. Accordingly, $\frac{\Delta\epsilon^2_{m_k}}{\delta} = \frac{\Delta\epsilon_k{}^2}{\Delta s_{m_k}} \simeq \frac{\partial\epsilon_k^2}{\partial s_{m_k}}$. Substituting into Eq. (1) yields

$$\begin{aligned}
\mathbf{W}_{m_{k+1}} &\simeq \mathbf{W}_{m_k} - \mu\left(\frac{\Delta\epsilon^2_{m_k}}{\delta}\right)\frac{\partial s_{m_k}}{\partial\mathbf{W}_{m_k}} &(2)\\
&= \mathbf{W}_{m_k} - \mu\left(\frac{\Delta\epsilon^2_{m_k}}{\delta}\right)\frac{\partial\mathbf{W}^T_{m_k}\mathbf{X}_{m_k}}{\partial\mathbf{W}_{m_k}} &(3)\\
&= \mathbf{W}_{m_k} - \mu\left(\frac{\Delta\epsilon^2_{m_k}}{\delta}\right)\mathbf{X}_{m_k}. &(4)
\end{aligned}$$

This is the node-by-node version of the MRIII rule. We see that MRIII produces essentially the same weight changes as those determined by backpropagation, except MRIII adaptations occur one neuron at a time. It should now be clear that MRIII differentiates by signal perturbation, while backpropagation differentiates by using *a priori* knowledge about the derivative of the sigmoid.

Because parallel analog implementations of backpropagation require both forward and backward signal paths, backpropagation networks require considerably more circuitry than MRIII-based networks[2]. This complexity comes in addition to that brought on by the accurate derivative relationships that must hold between the forward and backward paths of an analog backprop network. Furthermore, the control circuitry is correspondingly more complicated in the backpropagation system.

The primary weakness of MRIII in comparison to backpropagation involves training speed. Because each weight update changes the response error, it will typically be necessary to recompute the unperturbed network

[1] If we use $\Delta\mathbf{E}_{m_k}$ to denote the response error perturbation vector, $\mathbf{E}_{perturbed_{m_k}} - \mathbf{E}_{unperturbed_{m_k}}$, we can develop an alternate expression for the squared error perturbation: $\Delta\epsilon^2_{m_k} = \Delta|\mathbf{E}_{m_k}|^2 = |\mathbf{E}_{m_k} + \Delta\mathbf{E}_{m_k}|^2 - |\mathbf{E}_{m_k}|^2 = 2\mathbf{E}^T_{m_k}\Delta\mathbf{E}_{m_k} + |\Delta\mathbf{E}_{m_k}|^2 \simeq 2\mathbf{E}^T_{m_k}\Delta\mathbf{E}_{m_k}$. Thus, for small error perturbations, we can approximate $\Delta\epsilon^2_{m_k}$ by $2\mathbf{E}^T_{m_k}\Delta\mathbf{E}_{m_k}$, twice the dot product of the error vector and the error perturbation vector. In comparison to the approach mentioned above, this method of computing $\Delta\epsilon^2_{m_k}$ generally requires fewer multiplications, and may in some cases be simpler to implement.

[2] Assuming the implementation determines backward errors with parallel analog hardware.

response after each neuron is adapted. Otherwise the weight changes can interfere with the effects of the node perturbations and contaminate the measured derivatives. The required number of unperturbed computations can be reduced, however, if we adapt weights only after several measured gradients have been accumulated in memory. Thus, to complete one training presentation, MRIII requires an average of something between one and two complete forward passes through the network for each summing junction in the structure. Backpropagation, in contrast, requires only one forward pass and one backward pass to complete one adaptation of all weights. In implementations where these computations require more time than that associated with weight updates and other overhead, MRIII can be slower than backpropagation by a factor as large as the number of summing junctions in the network. This relationship holds for implementations on both serial and parallel machines. In most current experimental analog systems, however, adaptation time is dominated by operations associated with weight updates rather than forward signal flow through the network. Furthermore, in a forthcoming paper, we discuss a number of enhancements that can be used to improve the speed performance of MRIII.

3 Using MRIII to Adapt Other Topologies

One of the advantages of the MRIII algorithm is its ability to adapt the weights of recurrent networks. MRIII can also be modified to allow the adaptation of networks composed of neurons which differ from the simple sigmoidal Adaline presented in Section 1.

3.1 Networks Comprised of Unusual Neurons

In some cases, it may be desirable to adapt neurons that have forms which differ from those discussed above. In such instances, it may be possible to find the proper form for the MRIII algorithm by replacing $\mathbf{X}_{m_k}$ in Eq. (4) by the correct expression for $\frac{\partial s_{m_k}}{\partial \mathbf{W}_{m_k}}$ from Eq. (2). Even this technique is not always feasible. Perhaps no accurate expression to describe the neurons will be known. For instance, it may be necessary to adapt neurons that integrate pulses, or process frequency multiplexed signals. Here, a "weight" may actually be a parameter which controls the characteristics of an adaptive filter or the frequency of a local oscillator or multivibrator. Along these lines, we may wish to adapt the weights in a recurrent network which is intended to oscillate in normal operation[3]. Although the MRIII approach discussed thus far cannot be applied directly, it is possible to apply a similar methodology to these problems. Specifically, we can sequentially add the perturbation δ to each *weight* in the network, rather than to each node. Here the adaptation rule for weight j of neuron m during training presentation k can be stated as follows:

$$w_{mj_{k+1}} = w_{mj_k} - \mu \frac{\partial \epsilon^2_{mj_k}}{\partial w_{mj_k}} \tag{5}$$

$$\simeq w_{mj_k} - \mu \left(\frac{\Delta \epsilon^2_{mj_k}}{\delta} \right). \tag{6}$$

This is the weight-by-weight version of the MRIII algorithm. For networks where the node-by-node method is a viable option, this version of MRIII will in the worst case be slower by a factor which is proportional to the average number of weights per Adaline.

3.2 Recurrent Topologies

If we wish to apply MRIII to a recurrent network in a nonoscillatory system, we must wait for the network state to settle before we make any measurements. After the continuous or discrete-time network has settled, the unperturbed error can be measured. In the spirit of the feed-forward procedure, a perturbation δ is then applied to one of the network's nodes or weights, and after the output again settles, the error is remeasured. Adaptation can then be performed according to one of two the MRIII rules stated in Eqs. (4) and (6).

Unlike feed-forward networks, it is possible for recurrent networks to have nonanalytic input-output characteristics, including oscillatory behavior. During normal adaptation, such problems occur rarely, but if they arise, it is necessary to correct the oscillation or discontinuity before continuing with the adaptation procedure. This correction can be achieved, for instance, by reducing weight magnitudes slightly or by proceeding immediately to a new input pattern without adapting the weights.

Recurrent networks can also be adapted by simple variants of the backpropagation rule. Pineda [3], shows this for continuous-time networks. Implemented on a digital computer, this technique requires rather costly numerical integration. A purely analog implementation of this technique would depend upon an analog ancillary network that must be able to accurately track changes in the original network. Rumelhart *et al* [4] present a

[3]Of course, such networks would require some sort of postprocessor to provide the system output and error signals.

backpropagation variant for discrete-time networks (now referred to as backpropagation through time) which can be used to train recurrent networks to converge to a desired value after a number of iterations. This approach, however, requires extra memory and a considerable amount of processing. Pearlmutter [2] has studied continuous-time versions of a more general variant of this technique that allows state trajectories to be followed through time[4].

The MRIII technique has the advantage that it offers a very simple method that allows one to implement true gradient descent on *arbitrary* discrete or continuous-time recurrent networks. It should be pointed out, however, that Pineda's method and MRIII both implement gradient descent only after the network's activation values have stabilized to a fixed point, while the backpropagation through time methods are more general in the sense that they can operate whether the network state is stable or dynamically changing. In a forthcoming paper we will present a simple approach that uses MRIII in conjunction with backpropagation to make the backpropagation through time algorithm a feasible method for training analog networks.

4 Discussion

In comparison to analog implementations, digital implementations of neural networks offer numerous advantages which can include amenability to software modification, weight portability, floating-point accuracy and dynamic range, and relative immunity to temperature extremes and noise problems. Nonetheless, analog, or partially analog circuitry can provide enormous advantages over digital circuitry in terms of cost, packing density, power usage, and in some cases, speed. The dynamics of continuous-time analog circuitry, might also provide some computational advantages, particularly when time dynamics are required in recurrent networks. Indeed, the potential advantages of analog implementations are so attractive that a majority of researchers in the field believe that analog circuitry is naturally suited to neural network implementations [5].

Thus, despite the strengths of purely digital approaches, there are strong indications that analog neural networks will play a significant role in future applications. It should be clear that when gradient descent is appropriate for training such systems, MRIII offers many strong advantages over backpropagation. MRIII offers a simple technique for adapting recurrent networks and it provides a method for adapting neurons which have unknown characteristics. MRIII might be used, for instance, to train complicated neural models which are based upon biological nervous systems. Also, in contrast to backpropagation, MRIII is relatively immune to the effects of nonideal components and voltage offsets, both of which are ubiquitous in analog circuitry. These characteristics make MRIII a viable approach for training real analog neural networks.

Acknowledgements

This research was sponsored by SDIO Innovative Science and Technology Office and managed by ONR under contract #N00014-86-K-0718, by the Department of the Army Belvoir R D & E Center under contract #DAAK70-89-K-0001, by NASA under contract #NCA2-389, and by Rome Air Development Center under contract #F30602-88-D-0025, subcontract E-21-T22-S1.

References

[1] M. Holler, *et al*, "An electrically trainable artificial neural network (ETANN) with 10240 "floating gate" synapses," *Proceedings of the International Joint Conference on Neural Networks*, vol. II, Washington, pp. 191-196, June 1989.

[2] B. Pearlmutter "Learning state space trajectories in recurrent neural networks," *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kauffman: San Mateo, CA, pp. 113-117, June 17-26, 1988.

[3] F. J. Pineda, "Generalization of backpropagation to recurrent neural networks," *Physical Review Letters*, vol. 19 no. 59, pp. 2229-2232, 1987.

[4] D. E. Rumelhart, G. E Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel distributed processing*, D. E. Rumelhart and J. L. McClelland, eds., ch. 8, Cambridge, MA: M.I.T. Press, 1986.

[5] J. Sage, ed. "Advanced implementation technology," in *DARPA Neural Network Study,*, pt. VI, Fairfax, VA: AFCEA International Press, pp. 345-377, 1988.

[6] B. Widrow, R. G. Winter, and R. Baxter "Learning phenomena in layered neural networks," *Proceedings of the IEEE First International Conference on Neural Networks*, vol. II, San Diego, pp. 411-429, June 1987.

[7] B. Widrow and R. G. Winter "Neural nets for adaptive filtering and adaptive pattern recognition," *IEEE Computer*, pp. 25-39, March 1988.

[8] R. G. Winter, *Madaline rule II: A new method for training networks of adalines*, Ph.D. dissertation, Dept. Electrical Engineering, Stanford Univ., Jan 1989.

[4]When discretized by the Forward Euler approximation, if T and Δt are set to one, and constraints at intermediate times are removed, Pearlmutter's main result is equivalent to that of Rumelhart *et al.*

Orthogonal Extraction Training Algorithm

Harold K. Brown
David F. Lange
John L. Hart

Department of Electrical Engineering
University of Central Florida
Orlando, Florida 32816

Abstract

When large amounts of input data, such as that from a video camera, are input to a neural network, the first layer is needed to reduce the raw data into a unique representation which is particular to the training set. Further, in certain applications, basic features of the training set may not be well understood or so complex that it becomes impractical to construct the required feature set. For these conditions, the Gram-Schmidt Orthogonalization procedure [1,2] can be utilized to develop an orthogonal vector set which is constructed from the basis function and unique to the training set as well. This technique is tested using individual alphanumeric characters placed in a 12 x 12 pixel array. After training the network, additional images containing random noise are presented to the neural network. Preliminary results show good recognition rates.

1 Introduction

Applying neural networks to real time imaging problems is critical to many applications, but at the same time imposes certain constraints as listed in Table I. To address these constraints, studies have concluded that non-iterative training may be the only feasible approach to solving the problem which is addressed by the algorithm presented here. The algorithm constructs an orthogonal vector set from the input basis functions based on the training set forming an n-dimensional space. The input vector is then expanded using the orthogonal vector set. The resulting vector is then projected into this n-dimensional space for further analysis. Next, an approximation is made to estimate the nearest neighbor and thus establish a match. Finally, the output is generated using a single layer perceptron.

A)	The network must be able to train in real time.
B)	The number of input permutations is far greater than the number of training sets.
C)	The hardware used for training must be the same as that for normal operation such that the system is upwardly scalable.
D)	No prior knowledge of the training vector set is known.

Table I. Neural networking system constraints for real time image recognition training.

This technique meets the needs of each one of the constraints listed above. To meet the speed requirement, the training algorithm utilizes only a single pass. The number of inputs is scaled down considerably after the orthogonalization of the input. The upward bound on the number of orthogonal functions is the lessor of the number of input basis functions (pixels) or the number of training sets. The actual size of the n-dimensional space can be further reduced by training only when an undesired output is generated, thus adding new orthogonal functions only when required.

Once the input has been expanded with the basis function, the n-dimensional vector is projected to a coordinate that is characteristic of the input. An estimate is made to the nearest neighbor in the decision space. The result of the estimate is used to map the associated output. If the output is in error then the n-dimensional decision space is increased by one by generating an additional orthogonal vector, adding a new unique vector, or adding both to the decision space for future nearest neighbor comparisons.

In an attempt to develop very fast neural networking training algorithms which have the ability to train in real time and still respond to arbitrary training sets, alternative training methods to the back propagation technique and other iterative training algorithms are being developed. As shown in the results below, good outcome can be obtained by this method. The primary penalty appears in the system memory requirements, but in certain applications this is acceptable. It is believed however, that improvements to the training algorithm can be made to reduce the probability of requiring an additional basis function for a given training vector. These improvements are expected to result in decreased memory requirements with little impact on the overall training time.

2 Orthogonal Extraction of Basis Function

The first layer utilizes a modified perceptron structure. The connection weights to this layer are determined using the Gram-Schmidt orthogonalization procedure. Equation 1 is employed to calculate the weights.

$$ {}^{t}W^{l}_{t,m} = \frac{ {}^{t}X^{l-1}_{m} - \left[\sum_{i=0}^{t-1} \left(\sum_{j=0}^{(N^{l-1}-1)} {}^{t}X^{l-1}_{j}\,{}^{t-1}W^{l}_{i,j} \right) {}^{t-1}W^{l}_{i,m} \right] }{ \left\| {}^{t}X^{l-1}_{m} - \left[\sum_{i=0}^{t-1} \left(\sum_{j=0}^{(N^{l-1}-1)} {}^{t}X^{l-1}_{j}\,{}^{t-1}W^{l}_{i,j} \right) {}^{t-1}W^{l}_{i,m} \right] \right\| } \qquad \text{for} \quad m = 0 \ldots {}^{t}N^{l-1} - 1 \qquad (1) $$

The notation for ${}^{t}W^{l}_{n,m}$ is such that the index t refers to the training set, l refers to the layer number, n refers to the node number, and m refers to the interconnect from node n proceeding to layer $l-1$, node m. To calculate this result, first a feed forward is performed which is described by the summation terms contained within the inner parentheses. If the feed-forward output is treated as a feed-back input and the feed-forward input treated as the feed-back output, the same operation can be used to calculate the outer summation. The only remaining operation involves the subtraction of the original input and subsequent normalization. Thus the same hardware can be utilized for most of the training.

Figure 1 shows a graphical representation of the process implemented by Equation 1. Vector **B** represents the already constructed orthogonal vector set. When a training set is input, vector **A**, the projection of vector **A** onto vector **B** is subtracted from vector **A**. The result is the **A'** vector which is the new orthogonal vector which is added to the n-dimensional decision space. If there were more than one initial orthogonal vector this process would be repeated until the input training set was reduced to zero or all of the projections to the orthogonal vectors were subtracted out.

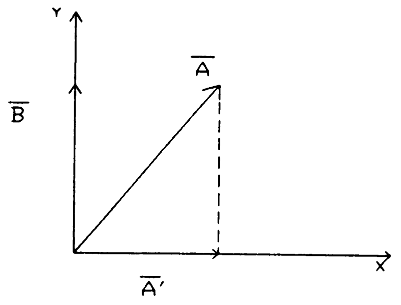


Figure 1. Extraction of orthogonal vector **A'** from **A** after projection to **B** is subtracted.

3 Mapping of Input Into Decision Space

The second layer deviates from the perceptron in that an additional offset is included and each connect has a sigmoid associated with it as computed in Equation 2. There may be supporting evidence that actual neurons have such complex structures [4] although this process is not fully understood at this time. Further it can be shown that the training can be mapped to a perceptron with a slight modification of the sigmoid activation function.

$$\sum_{i=1}^{N^{l-1}} \left\{ \begin{array}{ll} \dfrac{1}{1+e^{\,{}^{l-1}\beta^{l}_{n,i}\left({}^{i}X^{l-1}_{i}-{}^{l-1}\alpha^{l}_{n,i}\right)}} & \text{for } X_i-\alpha_{n,i}<\phi \\ \dfrac{1}{1+e^{\,{}^{l-1}\gamma^{l}_{n,i}\left({}^{i}X^{l-1}_{i}-{}^{l-1}\alpha^{l}_{n,i}\right)}} & \text{for } X_i-\alpha_{n,i}\geq\phi \end{array} \right\} \tag{2}$$

When binary image data is input to the neural network (i.e. each pixel is represented by a 1 or 0) the sigmoid can be linearized without serious degradation to the system. Equation 3 illustrates the forward computations used in the results section. In addition, each term is clamped except for the nearest neighbors.

$$S_n = \sum_{i=1}^{N^{l-1}} \left\{ \begin{array}{ll} \left({}^{i}X^{l-1}_{i}-{}^{l-1}\alpha^{l}_{n,i}\right)\,{}^{l-1}\beta^{l}_{n,i} & \text{for } X_i-\alpha_{n,i}<\phi \\ \left({}^{i}X^{l-1}_{i}-{}^{l-1}\alpha^{l}_{n,i}\right)\,{}^{l-1}\gamma^{l}_{n,i} & \text{for } X_i-\alpha_{n,i}\geq\phi \end{array} \right\} \tag{3}$$

The training is performed algebraically giving equal weight to each of the nearest neighbors and clamping beyond them. After a feed forward cycle is complete, a winner takes all function is used as input to the last layer.

4 Output Mapping

In the examples shown in the results section only a simple OR-ing function is required. For a given desired output a connect is made from the second layer to the third. With the second layer being a winner take all and only one of the outputs are to be active the LMS algorithm reduces to a single pass operation.

5 Results

To verify the accuracy and speed associated with the single pass feed-forward Orthogonal Extraction Training Algorithm (OETA), several tests have been adopted for characterization. Testing was performed over three separate character data sets. The first set included five hand generated characters A through E built on a 7 x 12 binary pixel grid and overlaid and left justified onto a 12 x 12 pixel array (Fig 2). However, both the second and third character sets were derived from standard IBM character patterns based on 7 x 7 binary pixel grids and overlaid into the center of a 12 x 12 pixel array. The hand written set has features that are more distinctive than the IBM character set.

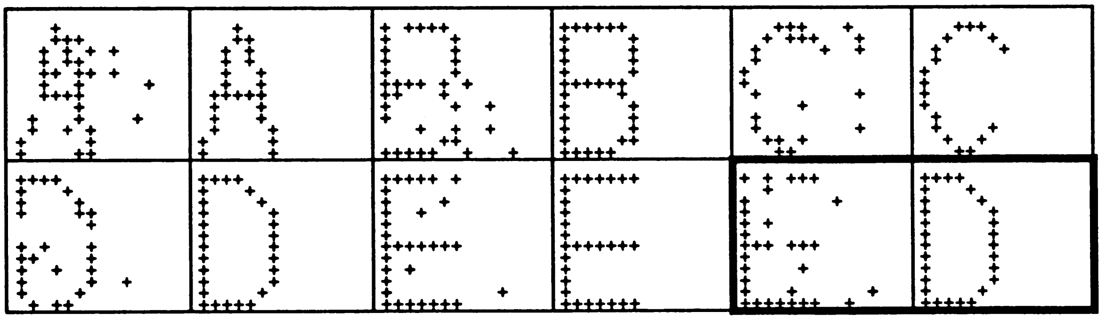


Figure 2. Handwritten character set showing typical inputs with 5% noise and without.

Figure 3. The 'E' character was recognized as a 'D' character. A 2% error rate.

The network trains to each character contained in the specified set. Following training, the test program randomly selects a character in the set for evaluation, adds a fixed percentage of random pixel noise, and then presents the noisy data to the network for evaluation. The test procedure is repeated 200 times for each noise level in 5% increments ranging from 0% to 30%. By the time 30% noise is added the characters are barely recognizable. Recognition success rates are then plotted against the noise level for the purpose of qualitative analysis as shown in Figure 4. Figure 2 illustrates the hand written character set and representative inputs with noise that were recognized correctly. Figure 3 illustrates one sample that failed to be recognized correctly even though the character features were ascertainable. Such failure in recognition occurs about 2% of the time.

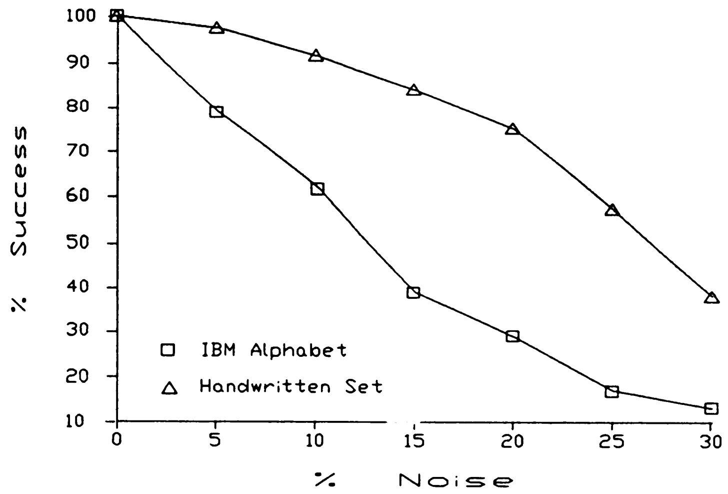


Figure 4. Presentation showing the percent success in recognizing a character as a function to percent noise. The delta marked line plots the results of the handwritten character set and the square marked line plots the results of the IBM character set. It should be noted that the IBM character set is made of fewer pixels than the hand written set. This plot shows that noise does not degrade performance until the character integrity is effected.

Other tests were conducted after training with randomly selected characters from the character set which also included 5% noise. The success rate drops considerably when training is conducted with noisy data. This is in contrast to back propagation trained networks where the recognition usually improves when training with noise [3].

6 Conclusion

Overall this technique gives good results while meeting the constraints outlined in Table I. Additional efforts need to be focused on why the low level error rate persists on good images. Also the technique needs to be evaluated for larger systems where the ratio of input permutations to training sets is much larger. Further evaluation is required to verify how the technique performs when the second layer training is mapped to a perceptron.

7 Acknowledgments

This research was supported in part by grants from PM Trade (contract #N61339-88-G-0002 order 0009), the Institute for Simulation and Training at the University of Central Florida, and the State of Florida High Technology and Industry Council. In addtion, the efforts of Joseph E. Madden are greatly appreciated for his support in producing this paper.

8 References

[1] Hart, John, Lange, David, Meehan, Steve, and Brown, Harold K. (PI), "Quarterly Report on Progress for A Neural Network Database Generator", Report for Project Manager for Training Devices (N61339-88-G-0002 Order 0009), University of Central Florida, Orlando, Florida, November 11, 1988.

[2] Szu, Harold and Scheff, Kim, "Gram-Schmidt Orthogonalization Neural Nets for O.C.R.", International Joint Conference on Neural Networks, IEEE Press, Washington D.C., I(547-555), June 1989.

[3] Gerrity, Francis J., Georgiopoulos, Michael, and Papadourakis, George, "A Study of the Generalization in Multi-Layered Feed Forward Neural Networks", Florida AI Research Symposium, date unknown.

[4] Sejnowski, Terrence, "Neural Computation", Plenary Session for IJCNN, Washington D.C., June 20, 1989.

A Model of the Neural Network Based on the Local Interaction Hypothesis and Two-Stage Modeling of Long-Term Enhancement

Hiroaki Kitano
Center for Machine Translation
Carnegie Mellon University
Pittsburgh, PA 15213 U.S.A.

Abstract

In this paper, we propose a model of the neural network based on physiological studies involving long-term enhancement and local interaction. Modification of weights in the neural network is caused by sustained stimulus, and involves two distinct stages. Each represents Protein Kinase C translocation and altered protein synthesis by messenger RNA. The Local Interaction Hypothesis is modelled to simulate postsynaptic activities of conditioning.

1. Introduction

In this paper, we propose a model of the neural network based on various physiological studies. The focus of the model in this paper is associative learning due to Long-Term Enhancement (LTE). In designing the model, we have taken into account physiological studies of, but not limited to, pyramidal cells in the CA1 region of the hippocampus, because they are identified as the group of cells which accounts for associative learning. Neural activities involve complex biochemical processes, some of which are not yet known, so that complete simulation is beyond our capability at this time. Instead, we take the more simple approach of designing a neural network model taking into account the effects of some major biochemical processes. Since we are interested in associative learning, we have focused on the LTE due to the Protein Kinase C (PKC) translocation, the altered protein sythesis by messenger RNA (mRNA), and the Local Interaction Hypothesis[Alkon, 1989]. In a subsequent section, we describe physiological studies we have taken into account in designing our model. Then, we describe our model.

2. Neural Realities

Neurons are biochemical apparatus. The excitation and inhibition of individual cells and the propagation of electric pulses are due to the biochemical reactions of neural systems. In designing an artificial neural network, physiological studies on how postsynaptic potential is changed and how sensitivities of postsynaptic membrane are changed and sustained are of primary importance, because these directly affect the basic architecture of the model.

Postsynaptic Potential:

Elevation of the postsynaptic potential is caused by an inflow of Ca^{2+} as observed by researchers including [Alkon, 1984]. There are two types of excitatory postsynaptic potential: a fast excitatory postsynaptic potential (fEPSP) and a slow excitatory postsynaptic potential (sEPSP)[1]. The fEPSP is caused by an inflow of ions due to glutamate (GLU), a kind of neurotransmitter, binding Kainate and/or Quisqualate (KQ) receptors to open ion channels[Collingridge and Bliss, 1987]. This process transmits presynaptic potential to postsynaptic potential by changing postsynaptic membrane permeability. The fEPSP is very rapid (on the order of a few milliseconds). The sEPSP is caused when the Mg^{2+} block of the N-methyl-D-aspartate (NMDA) receptor is reduced by depolarization due to a high frequency stimulation, and an influx of Ca^{2+} is observed[Collingridge and Bliss, 1987]. This process activates the Ca^{2+}-dependent process. Also, there is a cyclic AMP (cAMP)-dependent process of sensitivity modulation[Nathanson

[1]We do not discuss slow inhibitory postsynaptic potential (sIPSP) in this paper.

and Greengard, 1977] [Greengard, 1976][Greengard and Kuo, 1970]. However, whether this cAMP-dependent process contributes to persistent change in sensivity is at this point unclear.

Long-Term Enhancement:

We consider presynaptic and postsynaptic reasons for the induction of LTE[2]. The presynaptic explanation is that a sustained increase of neurotransmitter release induces LTE. Postsynaptically, LTE is due to a Ca^{2+} influx through a NMDA-sensitive and voltage-sensitive channel, and maintained by persistent PKC translocation from cytosolic PKC to membrane-associated PKC[Alkon and Rasmussen, 1988][Malinow et. al., 1988]. Elevation of sensitivity by this process starts minutes after the stimuli is given and lasts for weeks. When LTE is sustained for a longer period this is due to the alteration of protein synthesis by mRNA[Nathanson and Greengard, 1977]. There is a hypothesis that this change in the synthesis of protein is due to cyclic-AMP-dependent phosphorylation of nuclear proteins[Greengard, 1976].

Local Interaction:

[Alkon, 1989] proposes a hypothesis that postsynaptic change of sensivity involves extensive local interactions. His hypothesis is based on the observation that membrane-associated PKC increased in CA1 cell bodies, and to a lesser extent in dendrites, immediately after the paired stimuli are introduced, but three days later, PKC is found maximally in dendrites and less in the cell bodies. He further claims that the specific spatiotemporal domain affects promotion of PKC translocation and causes postsynaptic change. The hypothesis claims that, when conditioning occurs, postsynaptic sensitivity modulation is most affected by a nearby site.

3. Models of Associative Learning

In this section, we describe a mathematical model of an artificial neural network based on physiological studies discussed in the previous section. First, we describe a basic model with a simple local interaction. Then, we augment the model and incorporate a configurational factor. Finally, the modelling of LTE is discussed.

3.1. Local Interaction

The Local Interaction Hypothesis is adopted to simulate concentration of PKC in the postsynaptic membrane affected by nearby sites during a specific spatiotemporal window. In implementing the hypothesis, we introduce a *configurational matrix* which defines physical and chemical proximity between synaptic sites. The basic idea is to increase the weights of synapses when there is coincidental stimuli within a specific spatiotemporal window. The configurational matrix defines a spatial region as that where such weight modification can be triggered. The temporal window is defined as the modification which takes place when paired stumili are within a certain time interval. There are two ways to model this hypothesis. First, a basic model assumes that in the presence of paired stumili each weight involved is modified, and the output function employs a conventional threshold function. This model is expressed as equations (1) and (2).

$$o_i = \sum w_{ij}a_j - h \tag{1}$$

$$\Delta w^S_{ij} = \epsilon w_{ij}a_j + \sum \gamma w_{ij}a_j w_{ik}a_k c_{ijk} \tag{2}$$

$$\Delta c^S_{ijk} = \gamma w_{ij}a_j w_{ik}a_k c_{ijk} \tag{3}$$

where $o_i, w_{ij}, a_j, h, \epsilon, \gamma, w^S_{ij}, c_{ijk}$ and c^S_{ijk} are the output level of the i-th node, the weight between the i-th and j-th node, the activation level of the j-th node, the threshold value, the learning parameter for unconditioned stimuli, the learning parameter for conditioned stimuli, the weight between the i-th and j-th node due to PKC translocation, a configurational factor between the i-th and j-th node in the presence of stimuli from the k-th node, and the configuration factor due to the PKC translocation, respectively. Unlike the Hebbian Learning Rule[Hebb, 1949], weight can be changed without firing; it can be changed if stimuli are introduced. The term $\epsilon w_{ij}a_j$ represents a modification of weight whenever the stimuli are introduced. This term involves both presynaptic and postsynaptic change. In the equation (2), the second term of the right hand side shows a local interaction at conditioning. This term represents postsynaptic modification. The weight is strengthened by paired stimuli at nearby sites. The temporal window can be incorporated into the model using equations (4) and (5) by using $\gamma(t)$ which defines a

[2]Morphological explanation is ignored in our model since its nature and comformity to computer simulation is not known.

temporal window of local interaction for the PKC translation.

$$\Delta w_{ij}^S = \epsilon w_{ij} a_j + \sum \int \gamma(t) c_{ijk} w_{ij} a_j w_{ik} a_k dt \quad (4)$$

$$\Delta c_{ijk}^S = \int \gamma(t) w_{ij} a_j w_{ik} a_k c_{ijk} dt \quad (5)$$

Another way to model the hypothesis is to further speculate that, in addition to the basic model, fEPSP can be modulated in the presence of paired stumili.

$$o_i = \sum w_{ij} a_j + \sum \sum w_{ij} a_j w_{ik} a_k c_{ijk} - h \quad (6)$$

$$\Delta w_{ij} = \epsilon w_{ij} a_j \quad (7)$$

$$\Delta c_{ijk} = \gamma w_{ij} a_j w_{ik} a_k c_{ijk} \quad (8)$$

The equation (7) represents a basic modification of synaptic weight where the more the stimuli are given, the stronger the weight will be. This equation applies solely to each synaptic weight. The local interaction is represented in the equation (8). The configurational matrix is modified only when paired stimuli are given. As a result of weight modification, the network will learn association of stimuli, say A and B, where A is the conditioning stimulus and B is a conditioned stimulus which usually flows through the network as simulated in [Vogl et. al., 1989]. After the training, when the stimulus A alone is given, the B will be produced as an output of the network.

3.2. Long-Term Enhancement

We simulate two stages of LTE. We assume that LTE, w_{ij}^S and c_{ijk}^S, are due to the PKC translocation, and the sustained LTE, as represented in w_{ij}^L and c_{ijk}^L, is a result of increased protein synthesis by mRNA altered by PKC and cAMP dependent processes.

$$w_{ij} = w_{ij}^L + w_{ij}^S \quad (9)$$

$$c_{ijk} = c_{ijk}^L + c_{ijk}^S \quad (10)$$

Modification of weight is initially recorded on w_{ij}^S and c_{ijk}^S, and transferred into w_{ij}^L and c_{ijk}^L, respectively.

$$w_{ij}^L = \zeta \int w_{ij}^S dt \quad (11)$$

$$c_{ijk}^L = \eta \int c_{ijk}^S dt \quad (12)$$

ζ and η are parameters for converting PKC-based LTE into mRNA-based LTE. Although PKC-based LTE may decay as a result of dispersion, mRNA-based LTE is long lasting. We assume that the mRNA-based LTE is weakened by cell and dendrital tree loss in the cell population. Equation (13) and (14) roughly simulate loss of cells and dentrital trees in the cell population.

$$w_{ij}^S(t) = w_{ij}^S(t_0) e^{-(t-t_0)} \quad (13)$$

$$c_{ijk}^S(t) = c_{ijk}^S(t_0) e^{-(t-t_0)} \quad (14)$$

We expect that this two-stage modeling of LTE will provide our model with a capability to learn only common features of the data set and ignore uncommon features. Although the possibility of making generalizations from data by this type of learning scheme is not evaluated in our model, we assume that the two-stage modeling provides some filtering capabilities.

4. Discussions and Conclusions

In this paper, we proposed a model of an artificial neural network based on the Local Interaction Hypothesis and the two-stage modeling of LTE. The Local Interaction Hypothesis is incorporated into the model using the configurational matrix and a temporal function of weight modulation. The two-stage modeling simulates the lasting period of LTE due to PKC translocation and altered protein production by mRNA, and decay of LTE due to chemical dispersion and the loss of cell bodies and dendritic trees in the cell population.

As pointed out in [Alkon, 1989], a learning rule based on the Local Interaction Hypothesis should be different from Back Propagation[Rumelhart, 1986] and the Hebbian Learning Rule[Hebb, 1949]. We believe this is a reasonable conclusion when modelling the pyramidal cells of the hippocampus because neither Back Propagation nor the Hebbian Learning Rule conform to biological observations of actual neurons in the CA1 region of the hippocampus. Back Propagation assumes a reverse flow of information which causes change in the synaptic weight. Such reversed flows in real axons are unrealistic in a real neural system[Crick, 1989]. The Hebbian Learning Rule increases the connections of neurons that are fired at the same time. However, in hippocampal cells, there is evidence that cells do not follow the Hebbian Learning Rule. Instead, it could seem that local interaction may be a dominant learning rule in the CA1 of the hippocampus.

There are several biochemical processes our model has ignored. First, we have not modeled inhibition by an antagonist[Morris et. al., 1986]. Blockage of NMDA sites with aminophosphonovaleric acid (AP5), which is a NMDA antagonist, does not affect synaptic transmission, but prevents the induction of LTE. Such a role of an antagonist needs to be incorporated in a future model. Neurogenesis or *neural darwinism* [Edelman, 1987] is an important theme in learning, but is not discussed in this paper, mainly due to the fact that we are not sure about the mechanism and phenomena of neurogenesis in the hippocampus. Recent discovery regarding the RNA transport in dendrites [Gordon-Weeks, 1988] has not been incorporated in our model because the detail of the phenomena are not well understood. Application of our model to the Frequency Modulation Neural Network [Kitano and Tomabechi, 1989][Tomabechi and Kitano, 1989] is another future issue.

We need to explore the mathematical and experimental properties of the model. [Vogl et. al., 1989] proposed a model of an artificial neural network based on the biological observations by Alkon, and claim that their network learns more efficiently than Back Propagation. Since our network shares basic learning features with their network, investigation of the efficiency of learning is one important topic in the evaluation of our model. A computational experiment to evaluate the learning capacity of our model is now in progress.

Acknowledgements

The author would like to thank members of Carnegie Mellon community for fruitful discussions. Lyn was, again, a patient proofreader of my paper.

References

[Alkon, 1989] Alkon, D.L., "Memory Storage and Neural Systems," *Scientific American*, July, 1989.
[Alkon, 1984] Alkon, D.L., "Calcium-Mediated Reduction of Ionic Currents: A Biophysical Memory Trace," *Science*, Vol. 226, pp1037-1045, 1984.
[Alkon and Rasmussen, 1988] Alkon, D.L. and Rasmussen, H., "A Spatial-Temporal Model of Cell Activation," *Science*, Vol. 239, pp998-1005, 1988.
[Collingridge and Bliss, 1987] Collingridge, G.L. and Bliss, T.V.P., "NMDA Receptors - Their Role in Long-Term Potentiation," *Trends in Neuroscience*, Vol. 10, No. 7, 1987.
[Crick, 1989] Crick, F., "The Recent Excitement about Neural Networks," *Nature*, Vol. 337, pp129-132, 1989.
[Edelman, 1987] Edelman, G.M., *Neural Darwinism: The Theory of Neural Group Selection*, New York, Basic Books, 1987.
[Gordon-Weeks, 1988] Gordon-Weeks, P.R., "RNA Transport in Dendrites," *Trends in Neuroscience*, Vol. 11, No. 8, 1988.
[Greengard, 1976] Greengard, P., "Possible Role for Cyclic Nucleotides and Phosphorylated Membrane Proteins in Postsynaptic Actions of Neurotransmitters," *Nature*, Vol. 260, pp101-108, March 11 1976.
[Greengard and Kuo, 1970] Greengard, P. and Kuo, J.F., "On the Mechanism of Action of Cyclic AMP," *Adv. Biochem. Psychopharmacol.* 3:287-306, 1970.
[Hebb, 1949] Hebb, D.O., *The Organization of Behavior: A Neuropsychological Theory*, New York:Wiley, 1949.
[Kitano and Tomabechi, 1989] Kitano, H. and Tomabechi, H., *The Frequency Modulation Neural Network Theory*, Manuscript, Carnegie Mellon University, 1989.
[Malinow et. al., 1988] Malinow, R., Madison, D.V. and Tsien, R.W., "Persistent Protein Kinase Activity underlying Long-Term Potentiation," *Nature*, Vol. 335, pp820-824, 1988.
[Morris et. al., 1986] Morris, R.G.M., Anderson, E., Lynch, G.S. and Baudry, M., "Selective Impairment of Learning and Blockage of Long-Term Potentiation by an N-methyl-D-aspartate Receptor Antagonist, AP5," *Nature*, Vol. 319, pp774-776, 1986.
[Nathanson and Greengard, 1977] Nathanson, J.A. and Greengard, P. "Second Messengers in the Brain," In *The Biology of the Brain*, Llinas, R.R. (ed.), Freeman, 1977.
[Rumelhart, 1986] Rumelhart, D.E., "Learning Internal Representation by Error Propagation," In *Parallel Distributed Processing*, Rumelhart, D.E. and McClelland, J.L. (Eds.), 1986.
[Tomabechi and Kitano, 1989] Tomabechi, H. and Kitano, H., "Beyond PDP: the Frequency Modulation Neural Network Architecture," In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-89)*, 1989.
[Vogl et. al., 1989] Vogl, P.T., Alkon, D.L. and Blackwell, K.T., "Dynamically Stable Associative Learning (DYSTAL): A Biologically Motivated Artificial Neural Network," In *Proceedings of the International Joint Conference on Neural Networks*, 1989.

Tree Net: A Dynamically Configurable Neural Net

John A. Nevard
Department of Mathematics
University of California, Los Angeles
Los Angeles, CA 90024

1. Description of the Algorithm

We consider the basic pattern recognition problem of constructing a machine or algorithm to partition a given set of patterns into two previously specified disjoint subsets. We use the notation

$$I = \{-1, 1\}, \qquad \mathcal{P} = I^b, \qquad S \subset \mathcal{P}, \qquad n = |S|, \qquad S = S_A \cup S_b, \qquad S_A,\ S_B \text{disjoint},$$

and

$$f : \mathcal{P} \rightarrow I \text{ satisfies } f(x) = \begin{cases} -1 & x \in S_A \\ 1 & x \in S_B \end{cases}$$

so that $\mathcal{P}$ is the pattern space, S a given subset of patterns, and f is the characteristic function for S in $\mathcal{P}$. Then, an effective realization of f by a network of "neurons" is sought. We will denote such a network by $N(f)$. Here, the simplest model of a neuron, namely that of McCulloch-Pitts [1], will be sufficient, so that a neuron ν with fan-in k may be represented as the function $\nu : \mathbf{R}^k \rightarrow I$ given by

$$\nu(x_1, \ldots, x_k) = H(\sigma(x) - \theta),$$

where

$$\sigma(x) = \sum_{i=1}^{k} x_i w_i, \quad H(x) = \begin{cases} -1 & x \leq 0 \\ 1 & x > 0, \end{cases}$$

and $w_1, \ldots, w_k$, and θ are the adjustable weights and threshold of the neuron. By "effective realization" we mean a learning procedure that, given $f|S$, efficiently constructs a reasonably sized network which implements f on S.

Intuitively, at least, the demands for efficient construction and reasonable size are at odds with each other: A network implementing f, having $n(b+1)$ connections is trivial to construct, while, for most f's,the problem of finding the smallest network implementing f is, very probably, impossibly difficult. Thus, a learning procedure that could balance these two requirements might be inherently more efficient than many of the learning schemes in use where the architecture of the net is fixed before the difficulty of learning a specific problem is known.

The procedure is based on the idea that while $f|S$ may be a difficult problem, it is always possible to write $S = S_C \cup S_D$ ($S_C \cap S_D = \emptyset$), and usually easier to solve the problems of implementing $f|S_C$ and $f|S_D$ separately. Let

$$g(x) = \begin{cases} -1 & x \in S_C \\ 1 & x \in S_D \end{cases}$$

and suppose that, somehow, we have available $N(g)$, $N(f|S_C)$ and $N(f|S_C)$. Let $s : I^3 \rightarrow I$ be the function

$$s(x_1, x_2, x_3) = \begin{cases} x_2 & \text{if } x_1 = -1 \\ x_3 & \text{if } x_1 = 1 \end{cases}$$

and let $M = N(s)$, a net of fixed size (a version having two hidden nodes and six connections is easy to construct) which we may assume to be given. Then $N(f)$ can be constructed as shown in figure 1. Although all the nets except M operate in parallel (conceptually), the structure can be seen as a tree, with $N(g)$ acting as a parent node to the two subtrees $N(f|S_C)$ and $N(f|S_D)$. It will be useful here to make the usual distinction between internal nodes and leaf nodes; the algorithm will construct them quite differently, as detailed below.

As it stands, we now have a template for a class of learning procedures which can be filled in to yield an exact algorithm by specifying how $N(g)$, $N(f|S_C)$ and $N(f|S_D)$ are to be constructed. Unfortunately, given g, $f|S_C$ and $f|S_D$, our task appears hopeless, because there is no reason to suppose that $N(g)$ is any easier to construct than $N(f)$. To avoid this impasse, we replace $N(g)$ in figure 1 by $N(?)$, i.e., a random network, which partitions S into two subsets which we continue to call S_C and S_D, and implicitly defines the function it computes, namely g. We may well be able to do better than the random network $N(?)$, and we will address this below, in a preliminary way. For now, $N(?)$ will almost certainly have divided the problem into the two strictly smaller problems of computing $N(f|S_C)$ and $N(f|S_D)$.

Since these are smaller problems, we could apply the above procedure recursively to each of these problems, and end up with a net which is indeed $N(f)$. The serious drawback is that it will have $O(n)$ connections, and, at least from a priori considerations, will be virtually incapable of generalization. Instead of a completely recursive approach, then, at each stage a "local" learning algorithm is applied, i.e. to the problem of constructing $N(f|S_C)$ and $N(f|S_D)$. If, after a certain number of presentations $t = t(|S_C|, b)$ the algorithm has not succeeded, then the main procedure is applied recursively, partitioning S_C and S_D and continuing. This is a flexible scheme, since the choice of local learning algorithm at each stage is essentially arbitrary, as is the function $t(|S_C|, b)$. (For example, the purely recursive scheme, considered above, fits into this framework by choosing $t(|S_C|, b) = 0$ if $|S_C| > 1$.) As mentioned above, we can view the net as a tree, and it is now clear that internal nodes act to partition the patterns, while external nodes do the actual classifying.

The choice of the perceptron algorithm [2] [3] used here as the local learning algorithm at each stage was made to keep the size of the final net as small as possible, and to reduce the number of intermediate decisions. For instance, Back Propagation [4] fits into the above framework easily; the McCulloch-Pitts neurons, chosen only for simplicity, can be replaced by differentiable neurons. Then, however, at each stage, the architecture of the subnets must be chosen by some criteria, and rarely is a minimal net size achieved. Perceptron learning, on the other hand, trains a single neuron, and has the well-known property that if a one-neuron solution exists, it will be found [2]. On a purely practical level, although Perceptron learning is notorious for slowness, its inner loop is so simple that it often ends up outperforming more sophisticated algorithms whose convergence properties are not so well understood.

An optimal method for constructing $N(?)$ would have to determine a partition of S into $S_C \cup S_D$, where $f|S_C$ and $f|S_D$ are easy to learn, and then proceed to actually construct a net which effected the given partition. This is more difficult than the original problem—instead, we will settle for an $N(?)$ which partitions S into approximately equal-sized subsets, $|S_C| \approx |S_D|$, with no other constraints on how S_C and S_D are chosen. Then, we may define $N(?)$ to consist of the single neuron represented by $\nu(x) = H(w \cdot x - \theta)$, where $w \in S$ is chosen randomly and θ is adjusted to be the median of $\{w \cdot x | x \in S\}$. Clearly, $N(?)$ will partition S into two sets of approximately the same size. Geometrically,

$$S_C = \{\, x \in S \mid \|x - w\| \geq r \,\},$$

$$S_D = \{\, x \in S \mid \|x - w\| < r \,\},$$

where $r = \sqrt{2(b - \theta)}$.

The net can be trained incrementally, as the number of patterns grows. Suppose the net $N_1 = N(f|S_1)$ has been constructed, and we wish to construct $N_2 = N(f|_{S_1 \cup S_2})$. If N_1 consists of one node, then we merely continue training with the perceptron algorithm (or whatever local learning algorithm is being used) using $S_1 \cup S_2$. On the other hand, if N_1 has more than one node, $S_1 \cup S_2$ will be split into two subsets, and the splitting will continue down the tree until a leaf node is reached. At this point, the local learning algorithm continues at the leaf node, applied to the particular subset of $S_1 \cup S_2$ that has filtered down from the splitting nodes. Since the splitting nodes' weights are fixed after the first time they are adjusted, the tree may become unbalanced using this technique; in practice, however, this was not a problem. For instance (see below), only once did a tree trained incrementally on a total of 3000 patterns split the patterns as many as three times.

When the algorithm is working on a leaf node, and the number of presentations of patterns reaches $t(|S|, b)$ without successfully separating the patterns, the algorithm discards the weights of that node, constructs a partition node in its place, allocates two additional nodes which can be thought of as the children of the partition node, and appends the small fixed-size net M, so that its inputs are the outputs of the three nodes. The patterns are partitioned by the partition node, and the appropriate subsets are then used to train the two child nodes.

2. An Example: Handwritten Character Recognition

As an example intermediate between "toy" problems and practical applications, a net was built to recognize the handwritten block capital letters A through Z. For ease of entry (using a mouse), the letters were first input as 60×68 bit-images, and then scaled to 15×17 images. Thus, for this example, $\mathcal{P} = I^{255}$. The patterns were then shifted so that their centers-of-mass coincided with the center of the 15×17 rectangle.

Since classification of letters is a multiway rather than binary decision, we need a "forest" in place of a single tree. Here, 26 subnets, operating (conceptually) in parallel, were constructed using the algorithm described in the previous section, with the i^{th} net trained to return 1 if the pattern was identified as the i^{th} letter of the alphabet, and -1 otherwise. Two criteria were used for judging if the net's response was correct: A pattern was considered an "exact match" if the appropriate subnet returned 1, and all other subnets returned -1. Alternately, it was a "best guess" if the appropriate subnet responded more strongly than all other nets (even if that strongest response were negative).

Because of the nature of the algorithm, learning a given set of patterns was not considered complete until the resulting tree achieved 100% exact matches. For simplicity, $t(|S|, b) = 20000$ was used. A set of 1311 patterns was used to build the first tree, requiring 83 minutes on a Macintosh Plus before all patterns were correctly identified. The incremental building process described above was used to continue the learning, producing 11 trees in all, the last having been trained on 2984 patterns. 520 new patterns were then used to test the generalization capabilities of the trees. As figure 2 shows, the percentage of exact matches rose from about 63% for the first tree, to almost 77% for the last tree. The percentage for best guesses, although considerably better than that for exact matches, rose more slowly, from 82% for the first tree to 88% for the last tree.

An encouraging feature of the algorithm is that the size of the net seems to grow slowly, although more data is needed before anything definitive can be said. Here, for instance, the total number of nodes (distributed among the 26 subtrees) for the first tree, trained on 1311 patterns, is 76, whereas the last tree, trained on 2984 patterns, has 98 nodes. The distribution of those 98 nodes among the 26 subtrees is also interesting, the subtrees for certain letters having only one node and thus operating as perceptrons, while most letters require one splitting node and two perceptron nodes. The letter B (presumably hard to distinguish from E's and R's and to a lesser extent A's) ended up with seven nodes, three splitting nodes and four perceptron nodes. The algorithm, then, seems to have the desirable capability of recognising learning difficulties, and allocating more resources where they are needed.

3. Conclusion

Recent results have shown that learning to classify a given set of patterns using a net with fixed architecture is an NP-complete problem. It seems inevitable then, that practical learning algorithms will have to be capable not only of adjusting weights in a given architecture, but of adjusting the architecture of the net, if the problem seems to demand it. The method presented above is a beginning attempt that has the virtues of simplicity, flexibility and quite high efficiency. The node-adding criterion used here, of defining $t(|S|, b)$ as a constant, can doubtless be improved, but even as it stands, the method's efficiency is such that nets having 25000 connections can be built on a machine like the Macintosh Plus in about two hours.

4. Bibliography

[1] McCulloch, W. S. and W. Pitts (1943) "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bull. of Math. Biophys.* **5**, 115–133.

[2] Rosenblatt, F. (1959) "Two Theorems of Statistical Separability in the Perceptron," in Mechanisation of Thought Processes. *Proceedings of a symposium held at the Natural Physical Laboratory, November 1958.* **1**, 421–456. London: HM Stationery Office

[3] Minsky, M. and S. Papert (1969) Perceptrons. Cambridge, MA: MIT Press

[4] Rumelhart, D. E., G. E. Hinton and R. J. Williams (1986) "Learning Internal Representations by Error Propagation," *Parallel Distributed Processing, Rumelhart and McClelland, eds.*, 318–362. Cambridge, MA: MIT Press

Figures

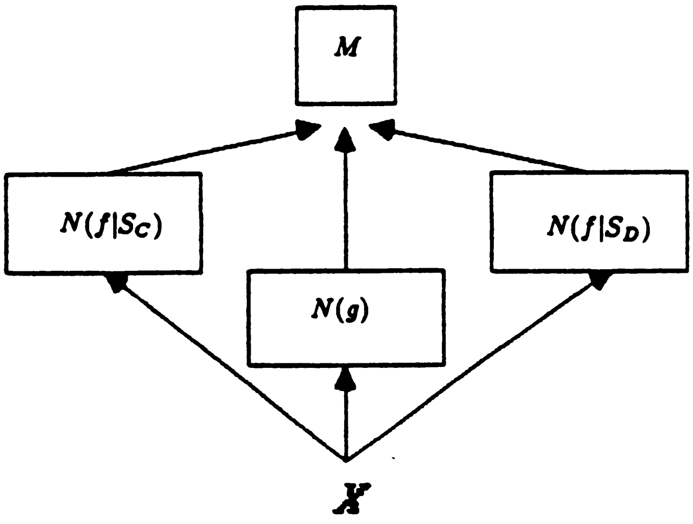


Figure 1

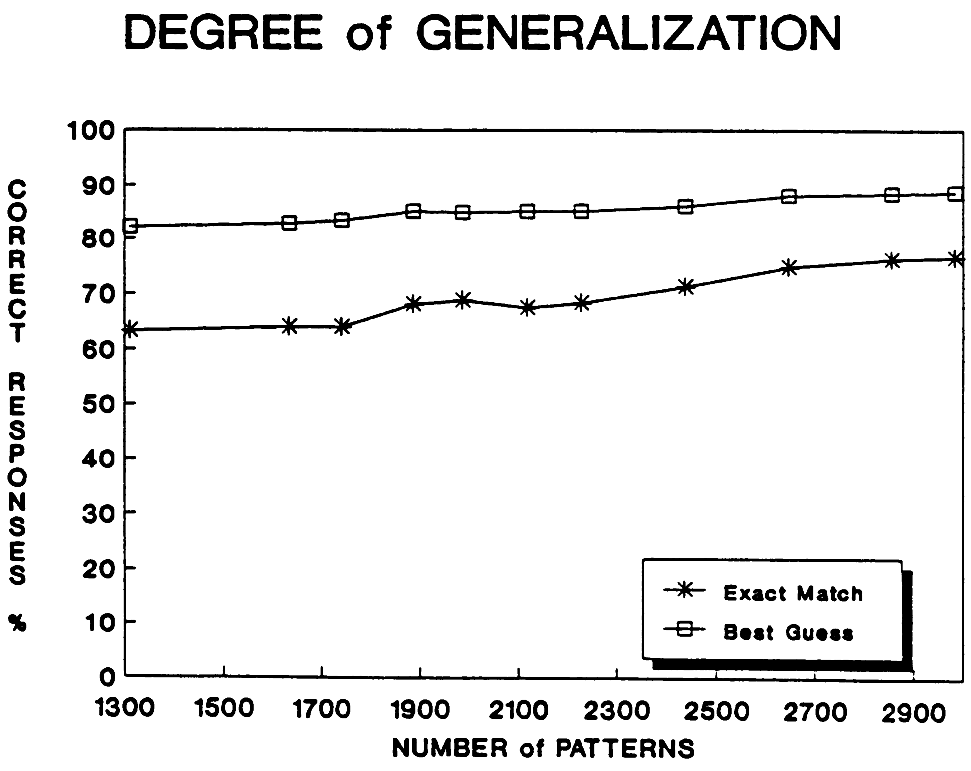


Figure 2

INFORMATION STORAGE MATRIX NEURAL NETWORKS

R. L. Waterland and N. Samardzija
Engineering Physics Laboratory, E. I. du Pont & Co. (Inc.),
PO Box 80357,
Wilmington, DE 19880-0357

Abstract

This paper presents a new neural network model that incorporates feedback. The model resembles a binary Hopfield net but requires neither symmetric connections nor orthogonal stored memories. We have developed a complete theory of the fixed points of the model and their basins of attraction. A novel feature of the model is direct control of the speed of convergence to stored memories and flexibility in tailoring the basins of attraction.

Introduction

Connectionist models feature an interconnected structure, expressed as a connectivity matrix, imposed upon a set of elementary neurons. Typically, each neuron applies a clipping- or sigmoid-type nonlinear transformation to a weighted sum of its inputs and adapts to its environment according to a learning rule. Most algorithms [1,2,3] are based on this description, with the main distinction being the learning rule imposed.

We choose to divide learning rules into two classes, local and global. Local learning is based upon an iterative procedure while global learning is based upon a non-iterative rule. An example of global learning is the Hopfield outer product rule [3].

In this paper a new algorithm based on a global learning rule is presented. The method describes a dynamical system with feedback which is not restricted to the symmetric connectivity matrix of the Hopfield structure. The model incorporates a discrete time, finite step architecture that operates on binary strings, and can easily be implementated in VLSI hardware.

The Network Architecture

Assume there are n elementary neurons, with the strength of the connection from the i-th output neuron to the j-th input neuron given by component a_{ij} of a real nxn connectivity matrix, A. The state of a neuron is represented by a real number in the range [-1.0,1.0] and the collective state of the system of neurons is an n-component vector in the real vector space, $\Re^n$. Matrix A is selected to store memories in the network, each memory being an n-bit binary string containing symbols 1 and -1. These memories are elements of the code C_n which contains 2^n n-bit elements each with a representative in $\Re^n$, e.g. the string 11-1 $\in C_3$ has a representative $X = [1.0,1.0,-1.0]^T$ in $\Re^3$. Each element of C_n defines a vertex of the Hamming cube in $\Re^n$.

The dynamical system presented relates vectors X(k-1) and X(k) that lie within or on the surface of the Hamming cube. Specifically, the dynamics takes a vector X(k-1) and maps it into a vector X(k) according to the prescription

$$X(k) = q(A\, X(k-1)) \qquad (1)$$

where A is the connectivity matrix and q is a clipping-type nonlinear vector function. If y_i is the i-th component of a vector $Y \in \Re^n$, then $q(Y) = [q_1(y_1),q_2(y_2),...,q_n(y_n)]^T$ with

$$\begin{aligned} q_i(y_i) &= y_i && \text{if } |y_i| \leq 1 \\ &= \text{sign}(y_i) && \text{if } |y_i| > 1 \end{aligned} \qquad (2)$$

For a given connectivity matrix A, the mapping (1) is iteratively applied to an initial vector $X(1) \in C_n$ to produce a sequence of iterates {X(1), X(2), X(3),...}. In the next section we will show how to encode memories in A so that this sequence of iterates is guaranteed to converge to one of the memories.

Learning and the Information Storage Matrix

We select m target vectors $Y_1, Y_2, Y_3, ..., Y_m \in C_n$ with $m \leq n$ requiring them to be linearly independent. These vectors span an m-dimensional subspace of $\Re^n$ called the target space. The question is: how does one store information in and extract information from such a space?

Augment the m target vectors Y_i with (n-m) vectors $Z_{m+1}, Z_{m+2}, \ldots, Z_n \in \Re^n$, called the slack vectors. The Z_i must not be elements of the code C_n and the set $Y_1, \ldots, Y_m, Z_{m+1}, \ldots, Z_n$ must form a basis for $\Re^n$. The target and slack vectors are arranged as the columns of an nxn non-singular matrix $T = [Y_1, \ldots, Y_m, Z_{m+1}, \ldots, Z_n]$. Define a real nxn diagonal matrix $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_m, \theta_{m+1}, \theta_{m+2}, \ldots, \theta_n)$ and form the matrix product

$$A = T \Lambda T^{-1} \tag{3}$$

A and Λ are related by a similarity transform; the column vectors of T are the eigenvectors of A and the diagonal elements of Λ are the associated eigenvalues [4], i.e.

$$A T = T \Lambda \tag{4}$$

In fact, given T and Λ, equation (4) can be solved directly for A by using the Gauss-elimination algorithm or by applying the δ-rule of perceptron learning.

There is an additional requirement on the eigenvalue matrix Λ. The m diagonal elements of Λ corresponding to the Y_i and the (n-m) elements corresponding to the Z_i must satisfy the conditions,

$$\begin{aligned} |\lambda_i| &> 1.0 \quad i=1,m \\ |\theta_i| &< 1.0 \quad i=m+1,n \end{aligned} \tag{5}$$

This eigenvalue assignment establishes a saddle topology in $\Re^n$. Any matrix A satisfying these requirements is called an Information Storage Matrix and may be used as a connectivity matrix of the neural network.

Operation of the Network

Each initial state of the system is a binary string which has a representative X(1) in $\Re^n$. Applying relation (1) iteratively to X(1) produces a sequence of states each lying within or on the surface of the Hamming cube. It can be shown [5] that this state sequence must evolve towards the target space, i.e. the iterates of any initial state must move towards the desired memories.

The outlined procedure requires the initial state to evolve on a saddle in $\Re^n$ with 'downhill' directions pointing directly at the stored memories and 'uphill' directions everywhere else.

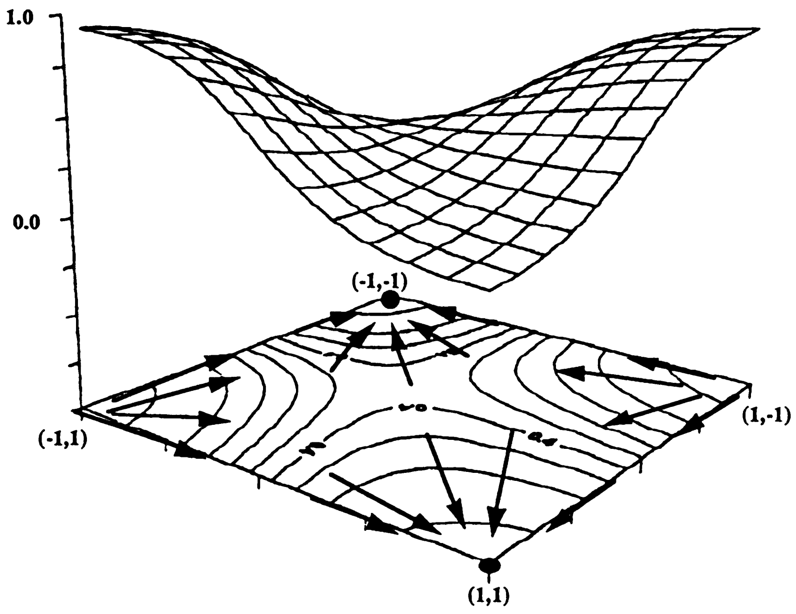


Figure 1: A combined 3-d and contour plot of a saddle in $\Re^2$. Stored memories, represented by black circles, lie at the vertices (-1,-1) and (1,1). Dynamical flow is represented on the contour plot by the arrows. The flow is always towards the memories

Figure 1 illustrates the type of surface upon which a state evolves. The only stable fixed points of the dynamical system lie at the vertices of the Hamming cube corresponding to the stored memories.

An Example: a 1-bit Feature Extractor

Consider the complementary 4-bit code D_4 represented by the eight real vectors, $[1.0,1.0,1.0,1.0]^T$, $[1.0,1.0,1.0,-1.0]^T$, $[1.0,1.0,-1.0,1.0]^T$, $[1.0,1.0,-1.0,-1.0]^T$, $[1.0,-1.0,1.0,1.0]^T$, $[1.0,-1.0,1.0,-1.0]^T$, $[1.0,-1.0,-1.0,1.0]^T$ and $[1.0,-1.0,-1.0,-1.0]^T$. Choose $\mathbf{Y}_1 = [1.0,1.0,-1.0,-1.0]^T$ and $\mathbf{Y}_2 = [1.0,-1.0,-1.0,1.0]^T$ as target vectors and $\mathbf{Z}_3 = [0.0, 0.0, 1.0, 0.0]^T$ and $\mathbf{Z}_3 = [0.0, 0.0, 0.0, 1.0]^T$ as slack vectors. In this case the matrix T is given by

$$T = \begin{pmatrix} 1.0 & 1.0 & 0.0 & 0.0 \\ 1.0 & -1.0 & 0.0 & 0.0 \\ -1.0 & -1.0 & 1.0 & 0.0 \\ -1.0 & 1.0 & 0.0 & 1.0 \end{pmatrix} \tag{6}$$

Take the eigenvalue matrix to be $\Lambda = \text{diag}(\lambda_1, \lambda_2, 0.5, 0.5)$ with $|\lambda_1|, |\lambda_2| > 1$. In the particular case, $\lambda_1 = 3.0$ $\lambda_2 = 2.0$ we have

$$A = \begin{pmatrix} 2.5 & 0.5 & 0.0 & 0.0 \\ 0.5 & 2.5 & 0.0 & 0.0 \\ -2.0 & -0.5 & 0.5 & 0.0 \\ -0.5 & -2.0 & 0.0 & 0.5 \end{pmatrix} \qquad [7]$$

representing a non-symmetric connectivity pattern in which each neuron connects to itself. Applying the procedure to all elements of the code D_4 produces the following basins of attraction,

$$\text{Basin of } \mathbf{Y}_1 \leftarrow \left\{ \begin{matrix} \text{Code elements} \\ [1.0, & 1.0, & 1.0, & 1.0]^T \\ [1.0, & 1.0, & 1.0, & -1.0]^T \\ [1.0, & 1.0, & -1.0, & 1.0]^T \\ [1.0, & 1.0, & -1.0, & -1.0]^T \end{matrix} \right. \qquad \left. \begin{matrix} \text{Code elements} \\ [1.0, & -1.0, & 1.0, & 1.0]^T \\ [1.0, & -1.0, & 1.0, & -1.0]^T \\ [1.0, & -1.0, & -1.0, & 1.0]^T \\ [1.0, & -1.0, & -1.0, & -1.0]^T \end{matrix} \right\} \rightarrow \text{Basin of } \mathbf{Y}_2$$

are attracted to target $\mathbf{Y}_1 = [1.0,1.0,-1.0,-1.0]^T$ are attracted to target $\mathbf{Y}_2 = [1.0-1.0,-1.0,1.0]^T$

The neural network is operating as a 1-bit feature extractor - it separates the code according to the sign of the second bit.

This is an elementary Information Storage Matrix Network. Other types we have considered include multi-bit feature extractors, Boolean logic elements and cascades of elementary networks. All are possible, and can be directly implemented with present-day VLSI hardware.

Learning in the Network

The learning parameters in the proposed model are the slack vectors $\mathbf{Z}_i$ and the eigenvalues λ_i and θ_i. These parameters can be used to tailor the domains of attraction of the model so that a particular function is executed [5].

The effect of increasing the λ_i is generally to change the mean rate of convergence to the desired solutions. For example, consider the 10-bit complementary code D_{10}, with two targets $\mathbf{Y}_1$ and $\mathbf{Y}_2$ and eight slack vectors, $\mathbf{e}_3$ to $\mathbf{e}_8$ where $\mathbf{e}_j$ is the j-th column vector of the 10x10 identity matrix. Take $\theta_i = 0.5$ for the slack vectors and vary λ_1 and λ_2 in the range 1.1 to 3.0. For each pair of values (λ_1, λ_2), apply the method to all 512 elements of D_{10} and compute the mean number of iterations required for convergence to the appropriate target. Figure 2 is a plot of the mean number of iterations vs. (λ_1, λ_2). If one eigenvalue is held fixed while the other is increased the mean is a decreasing function. When both λ_1 and λ_2 are close to unity, the mean becomes large although still is bounded by

4. When both λ_1 and λ_2 are larger than 2.0, the mean number of iterations is 1, i.e. every element of the code converges in 1 step! However, if the eigenvalues are made too large numerical instabilities may begin to appear. We have found that eigenvalues less than 5.0 give good results.

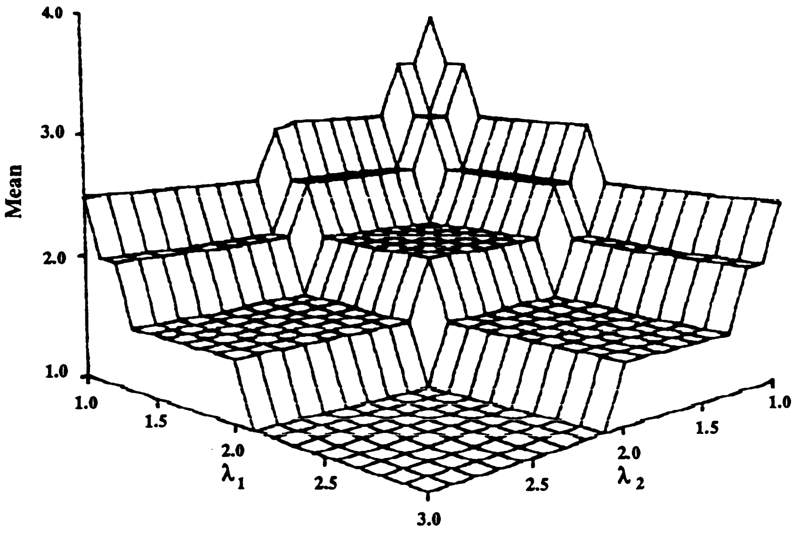


Figure 2: Mean number of iterations for convergence of all 512 elements of D_{10} as a function of λ_1 and λ_2

The eigenvalues control the slope of the various sections of the saddle in $\Re^n$. In contrast, the slack vectors are the means of moulding the saddle. If more slack vectors are included, there is more freedom to shape the domains of attraction, but there is a trade-off between the number of slack vectors and the capabilities of the model. When T is determined solely by target vectors, no learning is possible through the slack vectors. In contrast, with only a few target vectors there are correspondingly few possible decisions for the network to make.- We have addressed these limitations elsewhere [5].

Conclusions

The model presented can perform a wide variety of useful operations often applied in signal processing and control. In particular, the algorithm is well suited for implementation of Boolean operations. The network is guaranteed to converge to stored memories for all initial states. Construction of the connectivity matrix requires $O(n^3)$ floating point operations. A useful feature is that the network's execution speed can be controlled directly.

References

[1] Rumelhart, D. E., Hinton G. E. and Williams R. J., "Learning representations by back-propagating errors", Nature, 323, pp. 533-536 (1986); Pineda F. J., "Generalization of back-propagation to recurrent neural networks", Phys. Rev. Lett., 59(19), pp.2229-2232 (1987).

[2] Anderson, J. A., "Cognitive and psychological computation with neural models", IEEE Trans. System, Man and Cybernetics, SMC-13(5), pp. 799-815 (1983); Golden R. M., "The 'Brain-State-in-a-box' model is a gradient descent algorithm", J. Math. Psychology, 30, pp. 73-80 (1986); Rumelhart, D. E., Hinton G. E. and McClelland J. L., Parallel Distributed Processing, Vol. 1, pp. 66-68, MIT Press (1986).

[3] Hopfield, J. J., "Neural networks and physical systems with emergent collective computational abilities", Proc. Natl. Acad. Sci. USA, 79, pp. 2554-2556 (1982); Hopfield J. J., "Neurons with graded response have collective computational properties like those of two-state neurons", Proc. Natl. Acad. Sci. USA, 81, pp. 3088-3092 (1984).

[4] Gantmacher, F. R., The Theory of Matrices, Vol. 1, Chelsea, New York, 1977.

[5] Samardzija, N. and Waterland, R. L., submitted for publication.

NEURAL NETWORKS IN STATISTICAL CLASSIFICATION

Dr. Andrew K. C. Wong (Member IEEE)
Dept. of Systems Design
University of Waterloo
Waterloo, ON N2L 3G1
E-mail: akcwong@ever.waterloo.EDU

John. O. Vieth (Member IEEE)
Dept. of Systems Design
University of Waterloo
Waterloo, ON N2L 3G1
E-mail: vieth@ever.waterloo.EDU

1. INTRODUCTION

Both Discrete-Valued Data Clustering Algorithm (DECA) [1] and feature space transformation (FST) methods [2,3] evolve from the early work involving discrete valued data analysis. These methods, based on low order approximations of high order probability and statistical pattern recognition concepts, have been introduced to overcome major difficulties encountered in clustering and classification of both ordered (ordinal) and unordered (nominal) discrete-valued data. They both exhibited favorable results in experiments involving stochastically simulated data as well as actual clinical data for medical diagnosis [4].

Many of the earlier classification systems based on probability and information measures were not sufficiently tolerant to noise scattered throughout the entire data set and exhibited poor performance under such conditions. To overcome this problem, the concept of Event Covering has been introduced [5,6] which is capable of ignoring events that are statistically irrelevant for classification. By Event Covering we refer to the process of selecting only those subsets of joint events defined in an incomplete probability space that are relevant for use in the particular classification decision. The event covering process resembles association of events on the basis of statistical weights, with class discrimination based on the magnitude of the sum of weights of the active associations.

A refined approach to Event Covering has been introduced for inductive learning of classification rules and/or patterns from time-dependent or time-independent data that is subject to noise and uncertainty [7]. Experimental results indicate that this classification scheme is very well suited to real world applications. Unlike conventional techniques in Artificial Intelligence, these involve weighted decision rules which are difficult to accommodate under currently established architectures. However, there do exist some alternative architectures, such as Artificial Neural Networks (ANNs) under which this latest technique can become very useful.

2. LEARNING CLASSIFICATION RULES

Consider a problem domain consisting of entities representing objects or events and a set of classes into which these can be divided. Consider also that an entity can belong to only one class. Given a set of training data, probabilistic inference techniques can be used to inductively acquire a set of classification rules. To illustrate this, suppose the training data contains L entities that represent a sample space S. Suppose also that each of these can be assigned to a known class $C = c_p$ where $p = 1,2,3\ldots P$. Suppose that N distinct attributes A_j, $j = 1,2,3\ldots N$ can be defined to describe any entity in the sample space. Each attribute can take a value which is either ordinal or nominal such that:

$$val_j \in domain(A_j) = \{v_{jk} | k = 1,2,3\ldots K_j\}$$

Thus any entity can be represented by an n-tupple consisting of a set of attributes with particular values.

To simplify network implementation, we will represent the entities in a binary form. The number of bits required to represent A_j will be K_j. We may describe any entity in the sample space S by a tupple $X = \{ x_1, x_2, x_3 \ldots x_{NK} \}$ where $NK = \Sigma_j K_j$. Let us also define $x_i \in (0,1)$, where $x_i = 1$ represents the case where a value v_{jk} is observed in the entity and $x_i = 0$ otherwise. We may also describe the class to which an entity belongs by the p-tupple $C = \{ c_1, c_2, c_3 \ldots c_P \}$.

A supervised learning process can be applied when the entities in the training set have a-priori determined class membership. Using a widely accepted scheme such as Error Back-Propagation, the training process would consist of several iterations of comparison between the input n-tupple and the desired output class representation, while incrementally adjusting the "weights" in a given network of arbitrary structure. We propose a non-iterative approach to deriving a set of "weights" that can be instantiated in a rigorously defined network structure.

We begin the training process by examination of each of L entities in the training set, to count the following quantities:

opk : number of n-tupples in class c_p with attribute A_j having value v_{jk}
ok : total number of n-tupples with attribute A_j having value v_{jk}
op : total number of n-tupples in class c_p
epk : expected number of n-tupples in c_p with v_{jk} when A_j and c_p are independent.

For each p = 1,2,3... P, there are NK possible "opk"s and "epk"s, N possible "ok"s and a single "op".

Having determined these numerical quantities, we will use a statistical technique to determine which of the outcomes of the jth attribute (v_{jk}) are actually correlated with class c_p. In other words, which of the bits x_i in the nk-tupple X are relevant for classification. We use the adjusted residual [7] in the form:

$$dpk = (opk - epk) / \sqrt{(epk*(1-op/M)(1- ok/M))}$$

where $M = \Sigma_{pk}\ opk \le L$ due to possible missing data and $epk = ok*op/M$. Instances of x_i and c_p for which the adjusted residual has a magnitude of 1.96 or greater (95th percentile of a normal distribution) are assumed to have genuine correlation and are thus valid for classification.

Using only attribute values which are relevant for classification, it is possible to derive a set of weighted decision rules such that the hypothesis with the greatest associated weight is selected. We define the *weight of evidence* provided by x_i in favour of the entity (nk-tupple) being a member of the class c_p as opposed to not being a member of class c_p as follows [7]:

$$W(C=c_p / C \neq c_p | x_i) = I(C=c_p : x_i) - I(C \neq c_p : x_i)$$

where $I(C=c_p : x_j)$ represents Mutual Information [2,8]. It is clear that the weight of evidence depends only on the relative frequency of selected events in the training set and is not dependent on any assumptions as to the underlying probability density function for the data.

We can express the *weight of evidence* in terms of the cardinal quantities that we have defined:

$$W(C=c_p / C \neq c_p | x_i) = \log (opk*(M-op) / op*(ok-opk))$$

This is a logarithmic function with the property that its value decreases to zero as the interdependence between the class and the value of the attribute decreases. The weight of evidence decays rapidly as values and classes are less correlated. The classification process is accomplished by finding the maximum, over all classes (c_p), of :

$$W(C{=}c_p/C{\neq}c_p|X) = \Sigma_i W(C{=}c_p/C{\neq}c_p|x_i) = wc_p$$

We will refer to this formulation as the affinity of the input vector X to the class c_p

3. NETWORK IMPLEMENTATION

In figure 3-1 we illustrate an exemplary implementation of an Artificial Neural Network (ANN) that is programmed using the *weights of evidence.*

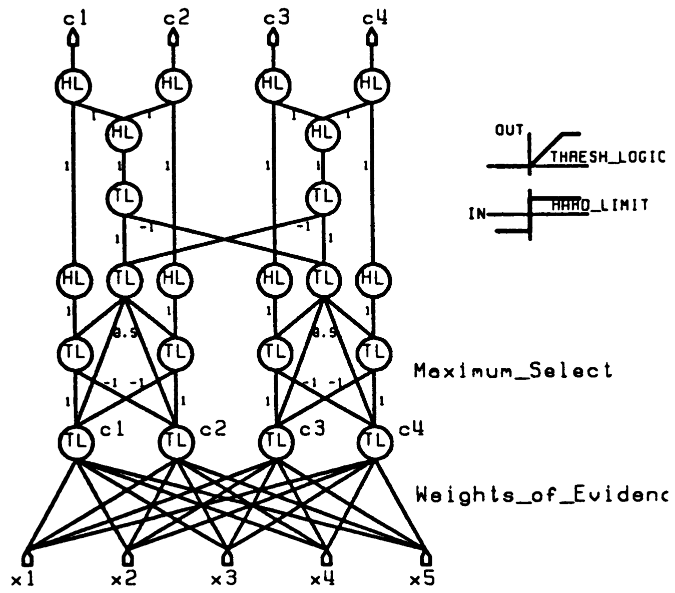


Figure 3-1 Network Topology

The entity or object to be classified, described by an nk-tupple X, is input to the network via x1 through x5. The first layer, programmed with normalized weights of evidence, evaluates the relative affinity of the entity to each class. These weights are normalized such that $0 \leq wc_p \leq 1$. The subsequent feed forward layers, adapted from [9,10], determine which entity to class affinity is greatest. The output of the network is the p-tupple C (c1 through c4) representing the most likely class for the entity.

The network implementation presented here is capable of only fisrt-order associations. For problems in which classes are not linearly separable, another layer of neurons can be added at the input to allow for higher order associations. We have been investigating this idea, however we consider these developments to be beyond the scope of this introductory paper.

4. EVALUATION AND CONCLUSIONS

In order to evaluate the performance of our classifier, we have chosen a medical diagnosis example. This involves real data recorded from patient examinations [11]. Each patient record consists of 12 attributes, each having between two and four unique values. Each patient has been diagnosed by a physician as belonging to one of four disease groups. It is important to note that a substantial number of records are incomplete and others had been incorrectly completed, introducing noise into the data. We randomly selected 75% of the records as a training set and used the remaining 25% to test trained network. We repeated the set selection, training and testing proceedure 10 times and averaged the results (see table 4-1 under OURS). The "Failure to Classify" category represents the case where no output was active, indicating an ambiguous choice between equal affinities.

As a benchmark comparison, an independent researcher in our laboratory [12] implemented a Back-Propagation network and trained it on the same medical data. She used a similar set selection, training and testing scheme. Classification was considered correct when the only output active above the 0.9 threshold represented the a priori class for the entity. Any other cases were considered incorrect. The average of the 10 trials are presented in table 4-1 under BPN. After trying several variations, we decided that this is the best that can be done with Back-Propagation learning for this example problem.

TABLE 4-1 EXPERIMENT RESULTS

Class	Correct		Incorrect		Reject	
	OURS	BPN	OURS	BPN	OURS	BPN
Chest (1)	100%	85%	0%	15%	0%	NA
Cardiac (2)	96%	84%	4%	16%	0%	NA
Abdominal (3)	87%	64%	10%	36%	3%	NA
Neurological (4)	96%	87%	0%	13%	4%	NA

Given the results of these experiments as well as speed of training and rigorously defined topology, we conclude that this statistically derived ANN implementation is worthy of further consideration and application.

5. REFERENCES

1. Wong, A. K. C. and Wang, D. C. C. (1979). "DECA: A Discrete Valued Clustering Algorithm", IEEE Trans. Patt. Anal. Mach. Intell., PAMI-1, 4, 342-349.
2. Wang, D. C. C. and Wong, A. K. C. (1979). "Classification of Discrete Data with Feature Space Transformation", IEEE Trans. Auto. Control, AC-24, 3, 434-437.
3. Chiu, D. K. Y. and Wong, A. K. C. (1984). "A Probabilistic Inference System", Proc. IEEE Conf. Pattern Recognition, 303-306.
4. Chiu, D. K. Y. and Wong, A. K. C. (1986). "Synthesizing Knowledge: A Cluster Analysis Approach using Event-Covering", IEEE Trans. Sys. Man & Cyber., SMC-16, 2, 251-259.
5. Wong, A. K. C. and Chiu, D. K. Y. (1987). "An Event-Covering Method for Effective Probabilistic Inference", Pattern Recognition, 20, 2, 245-255.
6. Wong, A. K. C. and Chiu, D. K. Y. (1987). "Synthesizing Statistical Knowledge from Incomplete Mixed-Mode Data", IEEE Trans. Patt. Anal. Mach. Intell., PAMI-9, 6, 796-805.
7. Chan, K. C. C. and Wong, A. K. C. (1988). "Learning From Examples in the Presence of Uncertainty", Proc. Intl. Comp. Sci. Conf. '88 AI Theory & Applications, Hong Kong. 369-376.
8. Osteyee, D. B. and Good, I. J. (1974). "Information, Weight of Evidence, the Singularity between Probability Measures and Signal Detection", Springer-Verlag, Berlin.
9. Lippmann, R. P. (1987). "An Introduction to Computing with Neural Nets", IEEE ASSP Mag., April 1987.
10. Martin, T. (1970). "Acoustic Recognition of a Limited Vocabulary in Continuous Speech", Ph.D. Thesis, Univ. of Pennsylvania.
11. Pao, Y. H. and Hu, C. H. (1984). "Processing of Pattern-based Information: Part I: Inductive Inference Methods suitable for use in Pattern Recognition and Artificial Intelligence", In Tou, J. T., Advances in Information Systems Sciences, vol. 9, Plenum Press.
12. McAndless, E. A. (1989). Univ. of Waterloo, Dept. Systems Design Eng., Private Communication.

EEROR FUNCTIONS TO IMPROVE NOISE RESISTANCE AND GENERALIZATION IN BACKPROPAGATION NETWORKS

Javier R. Movellan
University of California at Berkeley
e-mail: Movellan@garnet.berkeley.edu

This paper explores the relationship between the error-function used in BP and the statistical properties of the learned solutions when the teachers are contaminated by noise. If we know precisely the form of the noise distribution, the best error function is the negative likelihood. For instance, when the noise is Gaussian, the sum of squared errors, is the most efficient, when Laplace, the absolute error is best, and when Bernoulli, the cross-entropy error is the most efficient. In most practical situations we do not know precisely the form of the underlying noise distributions, and therefore we need error functions that perform well under a wide range of noise (robust error functions).

A backpropagation network (BP) can be seen as a function $\hat{y} = \eta\ (x\ ;\ w)$ that relates inputs (x) to output unit activations ($\hat{y}$). In this paper we use the "hat notation", common in statistics, to emphasize that BP networks are statistical estimators. The objective in BP is to estimate a "teacher function", $y_i = \mu(x_i)$. In most applications, this teacher function is probabilistic or contaminated by noise. This can be modelled by adding by adding a random vector ϵ_i, to the teacher function: $y_i = \mu(x_i) + \epsilon_i$, where the noise vector ϵ_i is independent identically distributed (iie) with probability distribution Fx_i, symmetric with zero mean.

Let us define the following:

1- *Sampling distribution of the solutions : The probability distribution of the solutions that we would expect to obtain if we drew an infinite number of samples from the contaminated teacher function and calculated the learned solutions for each sample.*

2- *Mean squared error of the sampling distribution of the solutions (MSE_T): The average expected value of the squared differences between the learned solutions and the expected values of the teachers. Notice that if the learned solutions are unbiased, MSE_T becomes the variance of the sampling distribution of the solutions. MSE_T can be used to measure noise resistance. Learning is resistant to a type of noise if the learned solutions are not very affected by the noise contamination of the teachers, or where the MSE_T is small. Analogous definitions can be used to study the sampling distribution of the generalizations over a set of inputs that did not appear in the training sample.*

BP learning is an statistical estimator. In statistics there are three major classes of estimators [1]. "L-estimators" are based on linear combinations of order statistics, (e.g., the trimmed mean), "R- estimators" on rank tests (e.g. the Hodges-Lehmann estimates), and "M-estimators" are defined by the optimization of an error function. BP learning is a generalized M-estimator. Instead of estimating points, it estimates contingencies.

3- *The generalized M-estimates, $\hat{y}(x) = \eta(x;\hat{\omega})$ for $\mu(x)$, based on the error function ρ and the sample S, are given by the values of ω that minimize*

$$\sum_{j=1}^{o} \sum_{i=1}^{n} \rho(\ y_{i\,j}\ ;\ \hat{y}_{i\,j}\ )\quad .$$

These estimates are determined implicitly by the set of differential equations

$$\sum_{i=1}^{o}\ \sum_{i=1}^{n} -\ \Psi(\ y_{i\,j}\ ;\hat{y}_{i\,j}\ )\ \frac{\partial\ \hat{y}_{i\,j}}{\partial\ \omega}\ = 0\ ,$$

with

$$\Psi = -\frac{\partial\, \rho(y_{ij} : \hat{y}_{ij})}{\partial\, \hat{y}_{ij}} .$$

We refer to this derivative as the "Ψ-function" or marginal error function. This function can be seen as an attentional filter. It modulates the influence of an exemplar on the learned solution. The learning rule is the gradient descent solution to the above equations,

$$w_{(t+1)} = w_{(t)} + \epsilon\, \Psi_{(t)} \frac{\partial\, \hat{y}}{\partial\, w(t)} ,$$

where t is the iteration index, and ε is the stepsize of the descent. Notice that to implement the gradient descent rule we only need to know the Ψ-function. Instead of proposing a ρ-function and calculating its derivative, we can directly propose a Ψ-function, which will implicitly define an error function. Figure 1 *shows some possible error functions for M-estimation. Their formulas are in the appendix. We were particularly interested in Tukey's Ψ-function. This function mapps a human attention phenomena, known as the discrepancy effect* [2]. *Normal individuals orient more readily to stimuly that are moderately discrepant from expectations. Higly predictable, and higly unpredictable events fail to elicit orienting.*

FIGURE 1

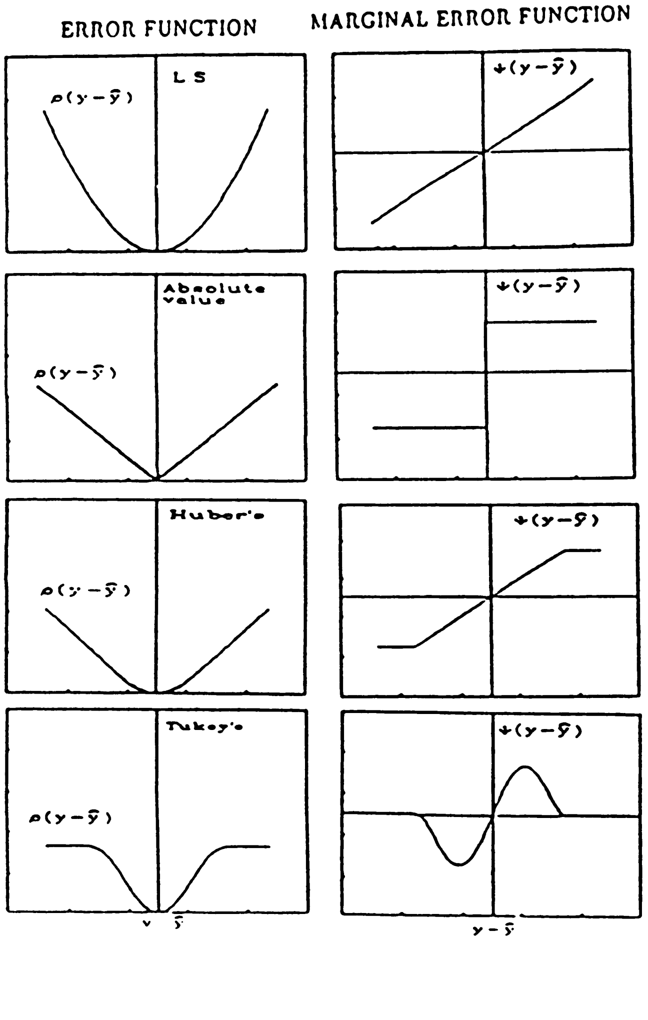


FIGURE 2
THE BALLISTIC PROBLEM

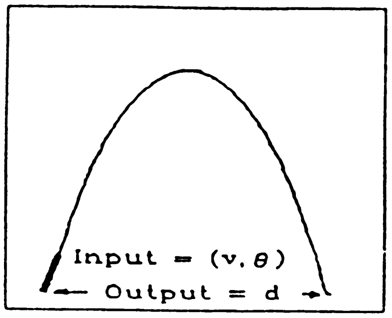
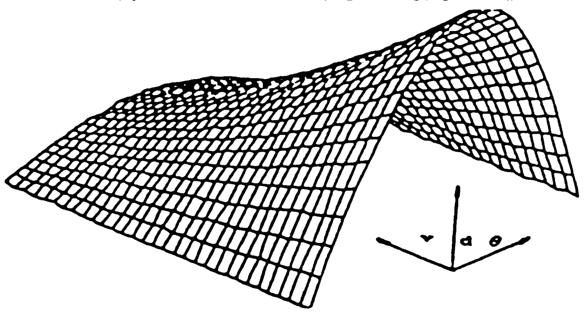


FIGURE 3
THE TEACHER FUNCTION FOR THE BALLISTIC PROBLEM



Simulations

The purpose of these simulations is to compare the error resistance and generalization properties of BP learning with different error functions. Commonly an estimator is considered robust if it performs well, (e.g., has small MSE_T and small MSE_G) on the following three benchmark distributions: Gaussian, Contaminated Gaussian, and Slash. The following ρ-functions were investigated: Least Squares; Absolute value; Huber's; Tukey's biweight. These error functions and the corresponding Ψ-functions are shown in Figure 1 *and in the appendix. In addition we investigated the performance of least squares compounded with weight decay, (e.g., the weights of each unit decay exponentially each trial). Several investigators claim that the use of weight decay in BP improves generalization. Weight shrinkage, as well as robust error functions indirectly improve generalization by resisting noise.*

The architecture used was the logistic 3*-layers, network (input-hidden-output) with* 6 *hidden units. The problem was a version of the ballistic prediction task . Figure* 2 *shows schematically the task. The network is presented with the initial velocity ,v, and angle, θ, and it must predict where the projectile will fall. The uncontaminated teacher function is the solution to the classical Newtonian equations of motion. Figure* 3 *shows the shape of the uncontaminated teacher function. The network had to learn* 25 *input combinations, (e.g., initial speed and angle conditions). The teacher function determines the uncontaminated teachers, (e.g., where the projectile falls). Before presentation to the network, these teachers were contaminated with one of the three benchmark noises. The network was allowed to learn until stopping criterion was attained . The learned solutions were then compared to the uncontaminated teachers. We also tested solutions to a set of* 36 *generalization points, that is,* 36 *input combinations for which the network had not been trained. The sampling-learning-testing process was repeated* 30 *times for each of the* 3 *by* 5 *noise v.s. learning rules.*

Table 1

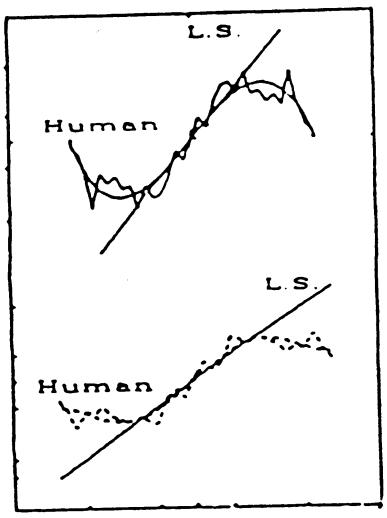
ROBUSTNESS OF THE ERROR FUNCTIONS

	SE	WD	AE	HUBER'S	TUKEY'S
ROBUSTNESS OF RESISTANCE	17%	46%	44%	50%	63%
ROBUSTNESS OF GENERALIZATION	50%	65%	75%	51%	89%
ROBUSTNESS OF SPEED	64%	85%	43%	74%	66%

SE is the squared error function
WD is squared error with weight decay
AE is absolute error function

Figure 4

Human Ψ – Function



The MSE of the sampling distribution of the solutions for a training and a generalization set were estimated. A common measure of robustness is the worst case scenario for the three *benchmark noise conditions. Table 1 shows robustness of resistance to noise, of generalization, and of speed for the 5 error functions. The details of how these indexes were calculated are in [2]. Clearly BP with the squared error function is not robust. Weight decay was very beneficial for noise resistance, generalization, and especially for speed of convergence. Absolute error was more resistant and produced better generalizations than squared error, but was slower. Huber's function was commensurate to squared errors plus weight decay. The best performance was obtained with Tukey's function, which greatly improved robustness of noise resistance, and generalization.*

Robust error functions and human estimation

Our goal was to estimate the Ψ-function used by humans in a simple estimation task. The task was to fit a horizontal line to an array of points. This can be modeled with the simplest BP net, e.g., a single unit with linear activation function and a constant non-zero input. Our approach was based on the fact that for this problem, the Ψ- function is proportional to the influence function. The details of the study are in [2]. Figure 4 shows the results for two subjects. If subjects were minimizing the squared error, the Ψ-function would be linear. It can be seen that the Ψ-functions are non-linear and of the robust to noise type.

Appendix

Define $u = \frac{e}{c\,S}$, *where* $e = y - \hat{y}$, *c is a constant, and S is a "scale parameter". In Huber's function* $c=2$, *in Tukey's* $c= 9$. *The scale parameter for each unit was calculated using* $S_{t+1}= S_t + .01\ (abs(e) - S_t)$.

Huber's Ψ-function is:

$$\Psi(u)= u \text{ ; when } abs\ (u) < 1;\ \Psi(u) = 1; \text{ when } abs(u) > 1 .$$

Tukey's function is:

$$\Psi(u)= u\ (1-u^2)^2 \text{ ; when } abs\ (u) < 1;\ \Psi(u) = 0; \text{ when } abs(u) > 1$$

REFERENCES

[1] *Goodall, C.(1985): M-Estimators of Location: An Outline of the Theory. in Hoaglin, D.C.; Mosteller, F. & Tukey, J.: Exploring data, tables, trends, and shapes, New York, John Wiley.*

[2] *Rodriguez-Movellan, JR (1989) Computational aspects of contingency detection: New options from connectionism. Doctoral Dissertation. Department of Psychology, University of California, Berkeley.*

Incremental Backpropagation Learning from Novelty-Based Orthogonalization

Ken Otwell

Martin Marietta Laboratories
1450 South Rolling Road
Baltimore, Maryland 21227

Abstract

Current neural network training techniques for hetero-associative pattern learning require a complete set of exemplars to be presented cyclically until the network weights converge to a common solution. In contrast, the recurrent novelty filter, a varient auto-associative system, can asymptotically converge to each pattern in sequence without disrupting performance on prior ones. In this paper we illustrate how the incremental orthogonalization properties of the recurrent novelty filter can be integrated with backpropagation of error in a feedforward system to provide incremental learning in hetero-associative systems as well.

Introduction

Weight adjustment based on backpropagation of error is an effective procedure for training complex multidimensional stimulus/response associations in feedforward networks [1]. The basic learning algorithm, popularized as the "generalized delta rule," [6] implements a gradient-descent procedure whose convergence depends on the cyclic presentation of a complete set of associations. Theoretically, response errors are accumulated for an entire presentation cycle, or "epoch," prior to weight adjustment; however, in common practice small adjustments are often made after presentation of each association with minor deterioration of optimal convergence.

In contrast, the novelty filter provides a mechanism for a single layer of nodes to habituate to known activation patterns and thus transmit only what is "novel" in any new pattern [2,3]. In effect, a novelty filter memorizes a given set of activation patterns and is thus able to generate an orthogonal projection from new patterns. In fact, the recurrent or feedback version of the novelty filter is "a continuous counterpart of the Gram-Schmidt [orthogonalization] algorithm," [4] and is thus incremental, i.e., it can be trained on each pattern in isolation until it converges (asymptotically) without disrupting the performance on previously learned patterns in the case that the pattern vectors are linearly independent. Even when linear dependence is present, the training algorithm tends to minimize the RMS error during crosstalk.

We have previously reported how the nonrecurrent, nonincremental version of the novelty filter can accelerate standard backpropagation learning [5]. In this paper, we illustrate how the recurrent version of the novelty filter can be combined with backpropagation to provide a mechanism for incremental learning of pairwise pattern associations. This is accomplished by creating a novelty filter within the input to each layer of the feedforward network and using the novelty term in place of input activation in the backpropagation training equations. In other words, after the feedforward network has converged to a stimulus/response association during incremental training, the novelty filter at each layer is trained on its input activation pattern. During training of subsequent associations, the novelty filter provides only the orthogonal components of the input vectors for feedforward weight adjustment. The unfiltered activation values are utilized in all cases for feedforward activation transfer. With these modifications, backpropagation can be used to train stimulus/response associations in sequence, as they are discovered, rather than concurrently as in current practice. In the remainder of the paper we define the integrated network equations and present the results from several simulations.

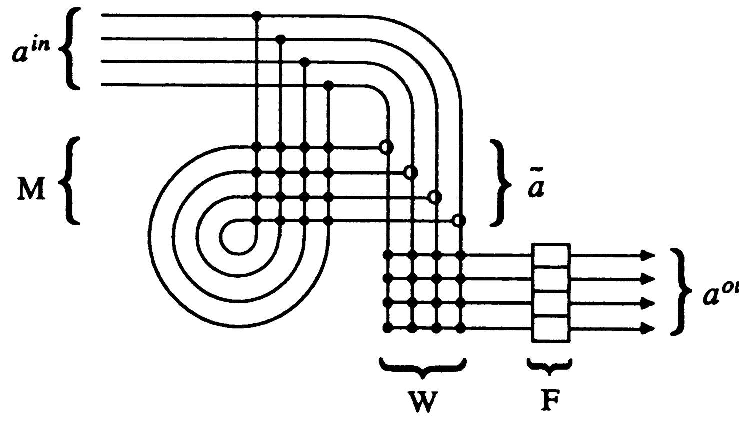


Figure 1. System Model for Single Layer with Novelty Filter

Orthogonalized Backpropagation

We use a two-layer fully connected feedforward network in our idealized backpropagation model. Each layer consists of 1) an input vector, taking elements from the reals between 0 and 1, exclusive; 2) a recurrent orthogonalizing filter on the input vector; and 3) an activation transformation function to generate output. (Many authors include the input buffer and refer to the network as a three-layer system.) A system model for a single layer with 4 inputs and 4 outputs is illustrated in Fig. 1. Feedforward activation transfer is governed by the common sigmoidal function:

$$a_k^{out} = f(\sum_i w_{ik} a_i^{in}) = [1 + \exp(-\sum_i w_{ik} a_i^{in})]^{-1} \tag{1}$$

where a_i^{in} is the activation value on input line i (the output of node i in the previous layer), each w_{ik} is an element of the feedforward weight matrix W, f is the sigmoidal "squashing" function, and a_k^{out} is the output activation on line k. Output activations are thus also real-valued quantities between 0 and 1, exclusive. The novelty term for each element of the input vector is governed by the following recursive equation:

$$\tilde{a}_i = a_i^{in} + \sum_j u_{ij} \tilde{a}_j \tag{2}$$

where $\tilde{a}_i$ is the "novelty" recursively generated from a_i^{in} and the weight matrix M with elements u_{ij}. M converges to produce a zero novelty vector for each input vector in sequence by the following equation:

$$\partial u_{ij}/\partial t = -\alpha \tilde{a}_i \tilde{a}_j. \tag{3}$$

For ease of digital simulation we use the *overall* novelty transfer matrix Φ which can be derived, in matrix notation, as follows:

$$\tilde{x} = x + \mathrm{M}\tilde{x} = (\mathrm{I} - \mathrm{M})^{-1} x = \Phi x \tag{4}$$

where $\tilde{x}$ is the novelty vector resulting from activation vector x. Thus, instead of utilizing training equation 3 in our simulations, we compute Φ directly by the Gram-Schmidt orthogonalization algorithm:

$$\Phi_l = \Phi_{l-1} - \tilde{x}_l \tilde{x}_l^T / \|\tilde{x}_l\|^2. \tag{5}$$

The "memoryless" Φ_0 is thus an identity matrix, whereas the matrix M_0 would contain all zeros.

Table 1. Percentage Correct Training and Recall for Incrementally Trained XOR

Number of Hidden Nodes	2	8	16	24
Percent Correct (1000 trials each)	0.0	33.1	82.7	99.6

Table 2. Percentage Correct Training and Recall for Incrementally Trained Blocks World State/Legal-Action-Set Pairs

Number of Hidden Nodes	30	60
Percent Correct (50 trials each)	8.0	42.0

To train associations in sequence, "as they are encountered," the generalized delta rule is modified to produce the "incremental delta rule:"

$$\Delta w_{ik} = \eta \delta_k \tilde{a}_i \tag{6}$$
$$\delta_k^{last} = a_k^{out}(1 - a_k^{out})(t_k - a_k^{out}) \tag{7}$$
$$\delta_k^{butlast} = a_k^{out}(1 - a_k^{out})\sum_j \delta_j w_{kj} \tag{8}$$

where t_k is the target value at output node k, δ_k is the error at node k, and η is the gain.

Simulation Results

We have tested our simulation extensively on two disparate problems, the XOR ("exclusive-or") and a larger problem drawn from the blocks-world planning domain. The XOR problem is a minimal non-linear pairing that demonstrates the robustness of the approach while highlighting its primary shortfall: many "superfluous" output lines (or hidden nodes) are often required in the first layer to ensure success. Table 1 illustrates the percentage of successful incremental training trials for different sized hidden layers. One bias input line was provided for each layer with a constant value of 0.9. An error range of ± 0.05 around target activations of 0.1 and 0.9 were used in all trials.

The number of iterations required to train each association increases with the number of previously trained ones due to the decrease in the available representation space and thus the decrease in the magnitude of the novelty vectors. This may be compensated for by increasing the gain as a function of time. We used an initial gain of 5 and increased it by a factor of 1.5 between examples in the reported XOR trials. We also discovered that the range of the initial random weights has a tremendous impact on success; we used evenly-distributed random values between -5 and +5 in these trials. The two-bit XOR patterns were trained in increasing numerical order requiring training cycles ranging from 1 to 20 on the first association to over 200 on the fourth for the 24 hidden node case. (A limit of 500 cycles was imposed.) As a control, we also attempted to incrementally train the XOR problem using the standard generalized delta rule, which always resulted in zero percent correct recall from prior learning for even much larger sized hidden layers.

The blocks world problem was designed to generate the set of legal actions from each legal domain state. Each input line to the first layer corresponded to a state proposition and each output line from the second layer corresponded to an action proposition (local coding). We used a 3-block, 1-table, and 1-arm version of the blocks world, giving 16 inputs, 18 outputs, and 22 legal states, and thus 22 pattern associations to be learned. The results are presented in Table 2.

Discussion

To our knowledge this is the first reporting of incrementally trained hetero-associative learning in neural networks. Further analysis is required to fully understand the behavior and limitations of this approach. An obvious problem is the large number of nodes required to ensure success -- much more than n nodes are required to learn n associations. This might be due to the interaction of the error surfaces from successive associations. It appears that once an error minimum has been discovered for one association, a "valley minimum" must exist in its error surface for the system to follow when converging to a minimum in the error surface of the next association, and overlapping valley minima of those two must be followed for the next one, etc. The large number of hidden nodes seems to provide the wrinkled error surface required for following the overlapping valley minima for many successive pairings. On the other hand, the results reported above are slightly pessimistic since many of the failures were only marginally outside of the error tolerance.

The catastrophic failures that did occur were of two types: *saturation* and *erasure*. In saturation failures, the first layer developed linearly dependent output activation patterns for multiple associations and the novelty at the second layer went to zero, preventing convergence for some associations. In all cases, because the number of associations was greater than the number of input lines, the first layer novelty went to zero before all associations were presented. However, since the feedforward activation is unaffected by the novelty filter, the first layer was frequently able to continue generating linearly independent output vectors despite not being able to adjust its own weights. Thus, convergence of later associations depended solely on weight changes in the second layer. In erasure failures, the training of later associations erased the learning from earlier ones. In these cases the system was able to converge on each association in sequence, but could not correctly recall earlier ones. During erasure failures the error was frequently observed to increase before finally converging, which always indicated that earlier learning was being erased or "forgotten." The likelihood of an association being erased appeared to decrease with the recency of its being learned; however, at this time no analysis has been performed for confirmation.

Perhaps larger systems or systems trained on patterns with greater regularity will be capable of a greater degree of generalization for compatible, if not predictable, new associations -- and thus will develop shorter valley minima to traverse. A potential, only slightly anthropomorphic explanation of erasure comes readily to mind: the system based the earlier associations on incorrect features which were later contradicted. To quote Robert Hecht-Nielsen: "Clearly, more research into error surfaces is needed [1]."

References

[1] Hecht-Nielsen, R., "Theory of the Backpropagation Neural Network," in *Proc. International Joint Conference on Neural Networks,* Vol. I, pp. 593-605, June 18-22, 1989.

[2] Kohonen, T., and E. Oja, "Fast Adaptive Formation of Orthogonalizing Filters and Associative Memory in Recurrent Networks of Neuron-Like Elements," *Biological Cybernetics* 21, pp. 85-95, 1976.

[3] Kohonen, T., **Self-Organization and Associative Memory,** Chapt. 4, Springer-Verlag, New York, 1984.

[4] Oja, E., "$\mathcal{S}$-Orthogonal Projection Operators as Asymptotic Solutions of a Class of Matrix Differential Equations, *SIAM Journal of Mathematical Analysis,* Vol. 9, No. 5, pp. 848-854, October 1978.

[5] Otwell, K., "Accelerating Back-Propagation Learning with Novelty-Based Orthogonalization." In *Proceedings of the IASTED International Symposium -- Expert Systems and Neural Networks,* August 16-18, 1989.

[6] Rumelhart, D.E., Hinton, G.E., and Williams, R.J., **Parallel Distributed Processing: Explorations in the Microstructure of Cognition,** Vol. 1, Chapt. 8, J.A. Feldman, P.J. Hayes, and D.E. Rumelhart, eds. The MIT Press, Cambridge, MA, 1986.

Backpropagation Improvements Based on Heuristic Arguments

Tariq Samad
Honeywell SSDC
1000 Boone Ave. N.
Golden Valley, MN 55427

1. Introduction

The backpropagation learning rule (Werbos, 1974; Rumelhart, Hinton and Williams, 1985) for multi-layer feedforward networks is a gradient descent procedure for minimizing a squared error criterion. The rule is defined as:

$$\Delta w_{ij} = \eta\, o_i \delta_j \tag{1}$$

$$\delta_j = \begin{cases} o_j'(t_j - o_j) & \text{for output units } j \\ o_j' \sum\limits_k w_{jk}\delta_k & \text{for hidden units } j \end{cases} \tag{2}$$

where w_{ij} is the weight from unit i (the "source unit") to unit j (the "destination unit"). Other terminology is as in (Rumelhart, Hinton and Williams, 1985). Usually, sigmoid activation functions are used with backpropagation:

$$o_j = \frac{1}{1 + e^{-net_j}} \tag{3}$$

While Eqs. (1-2) are derived mathematically, they can be explained on the basis of heuristic arguments. Such an explication can provide an intuitive understanding of backpropagation. Moreover, as this paper shows, it can help identify variations and extensions of backpropagation that are superior in a number of ways to the original rule—faster learning rates can be achieved, derivative computations can be dispensed with, and simpler unit activation functions can be used.

2. Backpropagation Heuristics

Several heuristics, implicit in Eqs. (1-2), are given below:

H1. If the error in the destination unit is positive (negative), the weight should be increased (decreased).

H2. The magnitude of the weight update should be proportional to the error in the destination unit.

H3. The magnitude of the weight update should be proportional to the value of the source unit.

H4. The magnitude of the error assigned to a unit should be inversely proportional to its degree of saturation.

H5. The assignment of a unit j*'s error to unit* i *should be proportional to the error in* j *and to the magnitude of the weight* w_{ij}.

H1, H2 and *H5* (a credit assignment heuristic) are obvious. *H3* is a consequence of the requirement that weight changes caused by one training pattern not interfere strongly with the response of the network to other training patterns. The effect of a weight update is proportional to the source unit value. A large update to a weight w_{ij} will affect response to network inputs that produce a high value for unit i more than to inputs that produce a low value for i. If the weight update is inversely proportional to the source unit value at the time of the update, there will be a high degree of interference between different training examples. Heuristic *H4* is useful because changing the output of a unit that is close to saturation (i.e., near 0 or 1) requires extremely large weight changes. The derivative of a unit output with respect to its net input is an indication of how distant the current value of the unit is from saturation.

3. Variations and Extensions

Eqs. (1-2) are just one possible implementation of *H1-H5*. Good results have been obtained with a number of alternative implementations. In addition, the heuristics can be refined. For example, note that the justification given for *H4* is relevant only for hidden units; output unit values must be driven to their target values (within some predetermined tolerance) irrespective of their degree of saturation. An alternative to *H4* is *H4'*:

H4'. The magnitude of the error assigned to a hidden unit should be inversely proportional to its degree of saturation.

There is another reason for scaling errors by derivatives in the original backpropagation rule: it implies a simulated-annealing-like weight update procedure in that error terms, and therefore weight changes, are larger initally than later on in the training process. *H4'* does not capture this benefit for weights to output units.

The credit assignment heuristic can also be refined. The assignment of a unit j's error to another unit should depend on how many different units source unit j. The greater the fan-in of unit j, the less the error that should be attributed to any particular source unit.

H5'. The assignment of a unit j*'s error to unit* i *should be proportional to the error in* j *and to the magnitude of the weight* w_{ij}*, and inversely proportional to the fan-in of unit* j.

Implementing these heuristics requires modifying slightly both the weight update rule and the error term formulae:

$$\Delta w_{ij} = \eta \, o_i \, \delta_j / N_j \tag{4}$$

$$\delta_j = \begin{cases} 0.25\,(t_j - o_j) & \text{for output units } j \\ o_j' \sum\limits_k w_{jk} (\delta_k / N_k) & \text{for hidden units } j \end{cases} \tag{5}$$

where N_j is the fan-in of unit j. The factor 0.25 (the maximum value of o_j') tends to make the magnitudes of the error terms for hidden and output units somewhat equivalent. The Results section shows simulation results for Eqs. (4-5) for a variety of problems. Substantial improvements over Eqs. (1-2) are consistently obtained.

Next, let us consider an alternative measure of unit saturation. Instead of o_j' in Eq. (2) or Eq. (5), we use the following function:

$$f(o_j) = \begin{cases} o_j & 0.0 \leq o_j \leq 0.25 \\ 0.25 & 0.25 \leq o_j \leq 0.75 \\ 1.0 - o_j & 0.75 \leq o_j \leq 1.0 \end{cases} \tag{6}$$

This function has a value of 0.25 when a unit has an output in its "central" region, and drops linearly elsewhere. The fact that the derivative need not be computed implies that unit activation functions that are not continuously differentiable can be used. For example, a linear threshold function can be used instead of Eq. (3):

$$o_j = \begin{cases} 1.0 & net_j \geq 4.0 \\ 0.125 \times net_j & -4.0 \leq net_j \leq 4.0 \\ 0.0 & net_j \leq -4.0 \end{cases} \tag{7}$$

This is significantly easier to compute than the sigmoid. The next section shows simulation results using Eqs. (6-7) (with Eqs. [4-5]). Again, the results are significantly better than for the original rule. The results for sigmoid units and with derivative computation are better overall, but for at least some applications, not by a large enough margin to compensate for the added computation.

It should be noted that we have only tried one possible linear threshold activation function, and only one possible heuristic measure of unit saturation. Further research may well uncover better functions yet. At any rate, the above results demonstrate that neither derivative computation nor continuously differentiable activation functions are necessary for backpropagation learning.

4. Results

Simulation results are tabulated below for a number of problems using the original backpropagation learning rule and the two extensions discussed above. In all simulations, no momentum term was used, initial weights were randomly assigned in $(-1.0,+1.0)$, and an output unit value was interpreted as a 1 (0) if it was above 0.75 (below 0.25). Weights were updated after every training example. The backpropagation extensions differed in two other ways from the original rule. First, weight adjustments were made even for network responses that were correct (by the above criterion). Thus, 1.0 and 0.0 were used as the actual t_j values in Eq. (2). Second, "expected source values" were used (Samad, 1988, 1989). 10 trials were performed for each experiment. The number of converged trials and the average number of iterations through the training set (over the converged trials) are reported. The results shown are the best ones obtained for each problem/rule.

It should be noted that the extensions discussed here require little or no additional computation or memory (indeed, they may result in some savings). This is in contrast to some of the other extensions to backpropagation that have recently been reported (Jacobs, 1988; Fahlman, 1988). These other extensions can be used in concert with the ones described in this summary for presumably faster learning yet.

Further details on rules and experiments can be found in (Samad, 1989).

Net Structure and Problem	Equations Used					
	1, 2, 3		3, 4, 5		4, 5, 6, 7	
	#Conv.	Average	#Conv.	Average	#Conv.	Average
4-2-4, Encoder/Decoder	10	157	10	14	10	19
8-3-8, Encoder/Decoder	10	215	10	36	10	53
16-4-16, Encoder/Decoder	10	242	10	49	10	55
10-5-10, Encoder/Decoder	10	258	10	17	10	18
2-2-1, XOR	8	573	10	71	9	406
3-4-1, Parity3	9	522	10	327	9	479
10-10-10, 10 Binary Random Associations	10	176	10	56	10	72

References

Fahlman, S.E. (1988). Faster-learning variations of back-propagation: an empirical study. *Proceedings of the 1988 Connectionist Models Summer School*, D. Touretzky, G.E. Hinton, and T.J. Sejnowski (Eds.). Morgan Kaufmann Publishers.

Jacobs, R.A. (1988). Increased rates of convergence through learning rate adaptation. *Neural Networks,* Vol. 1, No. 4, pp. 295-308.

Rumelhart, D.E., G.E. Hinton, and R.J. Williams (1985). *Learning Internal Representations by Error-Propagation.* ICS Report 8506, Institute for Cognitive Science, UCSD, La Jolla, Ca.

Samad, T. (1988). Back-propagation is significantly faster if the expected value of the source unit is used for update. *Neural Networks*, Vol. 1, Supp. 1. (Abstracts of the First INNS Meeting.)

Samad, T. (1989). *Backpropagation Extensions.* Honeywell SSDC Technical Report, 1000 Boone Avenue North, Golden Valley, MN 55427. (In preparation)

Werbos, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* Ph.D. Thesis, Harvard University Committee on Applied Mathematics.

LEARNING COMPLEX MAPPINGS BY STOCHASTIC APPROXIMATION

D. Sbarbaro
PJ Gawthrop
Department of Mechanical Engineering
The University , GLASGOW G12 8QQ
Scotland / U.K

ABSTRACT

In this paper, an alternative to the Back Propagation Algorithm, based on the stochastic approximation approach, is presented. This is applied to learn complex mappings between variables, like coordinate transforms and inverse dynamic robot estimation . Several simulations show that the algorithm achieves good results in few iterations.

1. Introduction

One of the more important capabilities of a Neural Network (NN) is the learning of a nonlinear mapping between inputs and outputs. Some authors have treated the problem as a discrete system (Albus, 1977) (MacClelland,1988) , and others has shown that the network can produce a mapping among continuous signals (Psaltis, 1988) (Josin ,1988)(Bassi,1989). It is thus possible to capture the structure and parameters of a nonlinear dynamic system in a single and simple representation. This kind of nonlinear identification is very important because most systems found in practice are nonlinear to some extent .

In general, nonlinear processes can only be adequately characterized over the whole operating range by nonlinear-models (Unbehauen,1988). The choice of the model structure is one of the major difficulties in dealing with identification of nonlinear systems ; this choice is vitally important since this will influence its application in prediction and control.

In this paper, a new algorithm to learn a mathematical mapping that represents a nonlinear dynamical system is presented . It is shown that this algorithm can approximate a function Y=F(X)+e given a set of examples [Y,X] , where e is the residual error. Several simulation showing the performance of the algorithm are presented .

2. Stochastic approximation algorithm

The network is considered as a nonlinear function described by

$$Y_n=F(\theta^o,X_n)$$

where

θ^o is the nominal parameter vector, X_n is a set of input vector and Y_n a set of output vector and

F is a nonlinear function composed of sigmoid functions

Considering a Taylor expansion around θ^O

$$R_n=h_n^T\theta \tag{2}$$

$$h_n=\dot{F}(\theta^o,X_n)=\frac{\partial F}{\partial\theta^o} \text{ and } R_n=Y_n-[F(\theta^o,X_n)-\dot{F}(\theta^o,X_n)^T\theta^o]$$

The exponential weighted least square estimate for θ is given by the equations (Albert, 1967).

$$\theta_{n+1}=\theta_n+a_n[Y_n-F(\theta_n,X_n)] \tag{3}$$

$$a_n=B_n^+h_n$$

where

B_n^+ is the pseudoinverse of B_n

$$B_n=\lambda B_{n-1}+h_nh_n^T \tag{4}$$

The algorithm involves linearization , and so the approximation error is smaller near to convergence. For this reason, old information corresponding to estimates of θ far from the convergence point is discounted according to the exponential forgetting factor λ

In practice, problems of illconditioning can arise , for example , when certain units are saturated. In order to avoid these effects on the inversion of B_n ,a pseudo inverse (Moler ,1977) is taken.

When an input pattern is applied to the network, the activation of each unit is determined using the sigmoid activation function

$$O_j=f\,(\sum_k w_{ji}\,O_i+w_{j0}) \quad \text{where} \quad f\,(x)=\frac{1}{1+e^{-x}}$$

To apply the algorithm it is necessary calculate h_n .

for an output unit $,\frac{\partial F}{\partial w_{ij}}=d_{ji}=O_j(1-O_j)O_i=\delta_{ji}\,O_i$ (5)

for a internal hidden unit $,\frac{\partial F}{\partial w_{ij}}=d_{ji}=O_j(1-O_j)\sum_k \delta_k\, w_{kj}\,O_i=\delta_{ji}\,O_i$ (6)

the summation is over all nodes in the layer above node j , then

$$h_n = \left[d_{10}\;\; d_{11}\ldots\ldots\ldots\ldots d_{Nn}\right]^T$$

where N is the number of units and n is the number of inputs of the last unit.

The algorithm can be summarized as

step 1 : Set weights and offsets to small random values

step 2 : Present an input and desired outputs

step 3 : Calculate actual outputs and derivatives by equations (5) and (6)

step 4 : Adapt matrix , equation (4)

step 5 : Change weights, equation (3).

step 6 : if (residual error)^2 < ε then stop

step 7 : repeat by going to step 2

Steps 4 and 5 are not found in the classical Back Propagation algorithm.

3. Some examples

3.1. Coordinate transform

For a perfect manipulator the cartesian endpoint position is given by

$$x=L_1cos\,(\theta_1)+L_2cos\,(\theta_1+\theta_2) \quad , \quad y=L_1sin\,(\theta_1)+L_2\sin(\theta_1+\theta_2)$$

where

$[x,y]$ position in the plane ; L_1,L_2 length of the segments and θ_1,θ_2 joint angles

These simulations show that it is possible to produce this kind of transformation using the structure shown in figure 3.1.1. The evolution of the error measured in meters is shown in figure 3.1.2.

3.2. Inverse model estimation of a cylindrical robot

The cylindrical robot consists of 3 differential ordinary equations: a rotation θ , a vertical translation z , and a radial translation r. The dynamical model of this robot is (Tourassis ,1987)

$$D\,(q\,)\ddot{q}+H\,(q\,,\dot{q}\,)=F\,(t\,)$$

the coordinate vector $q=[\theta \quad z \quad r\,]^T$

the inertial matrix $D\,(q)=diag\,[J+j\,(r) \quad M \quad (m_R+m_L)]$

the coupling vector $H(\dot{q},q)=[\frac{\partial j(r)}{\partial r}\dot{r}\dot{\theta} \quad Mg \quad \frac{-1}{2}\frac{\partial j(r)}{\partial r}\dot{\theta}^2]$

the external joint forces/torques $F(t)=[F_\theta \ F_z \ F_r]$

where $j(r)=(m_R+m_L)r^2-m_R R$, $\frac{\partial j(r)}{\partial r}=2(m_R+m_L)r-m_R R$

and the parameters are m_R , the mass of the radial link , m_L , the mass of the payload which is concentrated at the tip of the radial link, M , vertically translated masses , R , the length of the radial link, and J , the constant inertia of the vertical column

This relatively simple model preserves all the inherent coupling and nonlinear characteristics of a robot dynamic.

In the simulation only the nonlinear relation between θ and r is considered.

From the structure of the robot, the following dependencies are known

$$F_r=f(\ddot{r},\dot{r},\dot{\theta},r) \quad \text{and} \quad F_\theta=g(\ddot{\theta},\dot{r},\dot{\theta},r)$$

with this knowledge we reduce the system to NNs with 4 inputs rather than a NN with 2 outputs and 6 inputs.To estimate the inverse dynamic of the robot, the scheme shown in figure 3.1.0 was set up.

The NN was trained using a training region composed of several trajectories generated using cubic polynomial in the first quadrant. Then the system was simulated using the NN and given a reference trajectory .

The figures 3.2.2 a) and b) , show the evolution of the errors for a NN with 15 units during the training stage . The results of the simulations are shown in figure 3.2.3 , each surface representing the error against the initial coordinate for the estimation of the torque and final position.

4. Conclusions

A new algorithm that adjust the weights of a NN acting as a nonlinear mapping function has been proved .This algorithm uses a pseudoinverse to avoid illconditioning.

The results show a satisfactory performance of the algorithm as well as its fast convergence.

The NN produces a good generalization (interpolation) in the region that was trained. Outside this region this characteristic is lost.

More work is necessary in order to know the number of units required to match a certain structure.

5. References

1 A.E Albert and L.A Gardner, *Stochastic approximation and nonlinear regression*, The MIT pres, 1967.

2 J.L MacClelland and D.E Rumelhart, *Exploration in Parallel Distributed Processing*, The MIT press, 1988.

3 C. Moler , J. Little and S. Bangert, *PRO-MATLAB user guide*, The Math Works Ltd, 1987.

4 V.D. Tourassis and C.P Neuman *Robust nonlinear feedback control for robotic manipulators* , IEE Proc. Pt D , vol 132, No 4 ,134 - 143.

5 M. Kortmann and H. Unbehauen, *Two algorithms for model structure determination of nonlinear dynamics systems with applications to industrial process*, IFAC Symposium on identification on system parameter estimation(preprints), vol 2 (1988), 939-946.

6 D. Sbarbaro , *Learning internal representation using stochastic approximation*, Control Eng. Report 89.4, University of Glasgow, 1989.

7 L. Josin, *Neural Space generalization of a topological transformation*, Biological Cybernetics, 59, 283 - 290.

8 D. Psaltis , A. Sideris and A Yamamura, *A multilayered neural network controller*, IEEE Control System Magazine, April 1988,17 -21.

9 J.S Albus, *Data storage in the cerebellar model articulation controller (CMAC)*, Trans. of the ASME, J of dynamic system , measurement and control, sept 1975, 228-233.

10 D. Bassi and G.A Bekey *Decomposition of neural networks model of robot dynamics : A feasibility study* In W.Webster editor , Simulation and AI, vol 20 ,8 - 13.

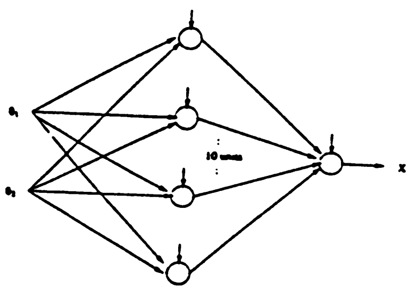


figure 3.1.1 general structure

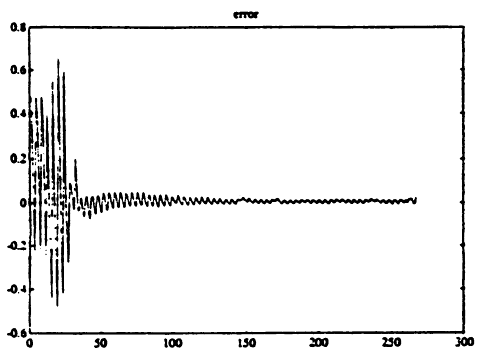


figure 3.1.2 Evolution of the error

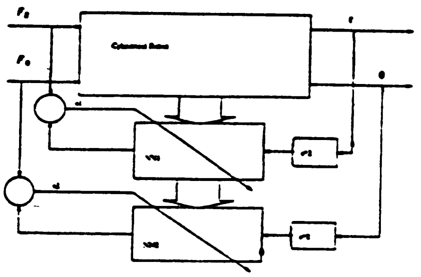
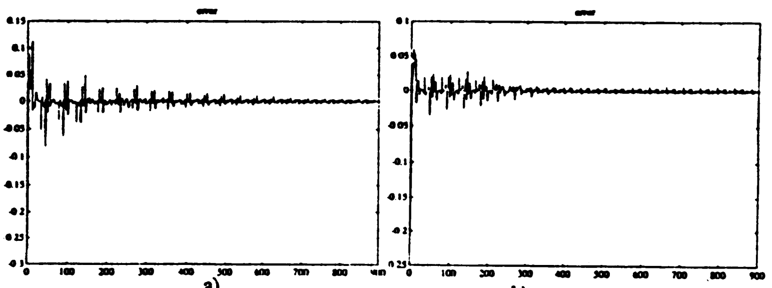
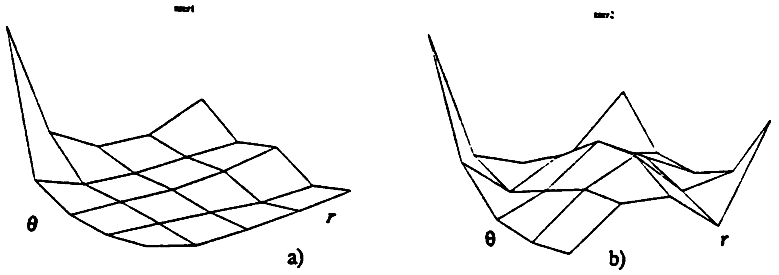


figure 3.2.1 Structure to estimate the inverse dynamic of a cylindrical robot



a) b)

figure 3.2.2 Error for a 15 hidden units NN
a) error between real external joint torque and estimate
b) error between real external joint force and estimate



a) b)

figure 3.2.3 Error for different staring point , and fixed ending point
a) sum square error for external joint torque
b) sum square error for external joint force

BACK-PROPAGATION LEARNING WITH COARSE QUANTIZATION OF WEIGHT UPDATES

P.A. Shoemaker
M.J. Carlin
R.L. Shimabukuro
Naval Ocean Systems Center
San Diego, CA 92045-5000
USA

With the extensive work and advances in connectionist or neural-network-like computational models in recent years, interest has arisen in implementation of such models in large-scale analog integrated circuitry. Implementation of modifiable weights is an important step which is required if a programmable or adaptive network is to be built, and a number of workers have reported efforts toward this end in analog circuitry [1,3-5,8,9]. However, iterative "learning" algorithms simulated on digital computers typically specify graded weight update values which can be represented to a high degree of precision, and it may be very difficult (and costly in terms of silicon) to compute and impose such weight updates with anything close to the same precision when efforts are undertaken to implement analog "learning" networks. Such anticipated difficulties suggest investigation of adaptive algorithms which might be inherently better suited for implementation.

Accordingly, we have considered learning procedures in which weight modifications during each update are very coarsely quantized, into two or three states. Such an approach may be of use, for example, when weights in circuitry are represented in an analog manner in the charge domain. This is the case with floating-gate MOS devices, which have been investigated for nonvolatile storage and representation of weight values [3,4,9], and with dynamic memory employing MOS capacitors [1,5], which have been used in conjunction with CCD-like structures to move charge about [8]. In both cases, the total charge integrated onto a floating gate or capacitor plate (or the difference in charge on two such structures) is used to represent a weight value in the circuitry. Moving a fixed quantum of charge onto or off of such a structure might be accomplished with simple switches and fixed current or voltage pulses, and would be far easier to do in parallel across an entire network than imposing charge increments with a continuous range of values and a different value at each weight circuit.

A precedent for the use of coarsely quantized weight updates is found in the work of Peterson and Hartman [6], who have studied the so-called "mean field theory" learning algorithm, a deterministic approximation to the Boltzmann machine [2]. They report that quantizing weight updates into two states improves performance of the algorithm on problems involving extensive sets of training data. Weight modifications across the network are all of the same magnitude and retain only the sign of the desired update as it would ordinarily be applied.

The learning rules which we have examined are simple variants of the well-known back-propagation algorithm [7]. The "neuron" outputs and back-propagated "delta" terms are computed in the usual way in a three-layer, feedforward network, and are used to decide which of the two or three possible increment values are to be applied to each weight. Each learning procedure is iterative and uses a training set with updates applied on a pattern-by-pattern basis, in strict analogy with standard back-propagation. The learning rules have been evaluated in sets of simulations and performance has been compared with that of standard back-propagation. In all cases, momentum or other "memory" terms were omitted from the learning rules and learning parameters were held fixed throughout each learning trial in our initial studies. Three small benchmark problems were used in these evaluations. One consisted of mapping sixteen orthogonal 32-component vectors to a one-of-sixteen output code, a second was based upon a training set composed of 40 pairs of arbitrarily chosen 15-bit vectors, and a third constituted identification of the parity of a set of fourteen four-bit binary numbers. Bipolar rather than truly binary "neuronal" activation functions and input vectors were used in the networks. Convergence was defined such that each output component for each training pattern was required to deviate from its target value by less than ten percent of the total output range.

Weight and bias updates were quantized into two states according to the rule

$$\begin{aligned} \Delta W_{ij} &= \eta \, \mathrm{Sgn}(\delta_i O_j) \\ \Delta B_i &= \eta \, \mathrm{Sgn}(\delta_i) \, , \end{aligned} \tag{1}$$

where δ_i is the delta term associated with the ith unit, O_j is the output of the jth unit, ΔW_{ij} is the update to the weight connecting the jth to the ith unit, ΔB_i is update to the adjustable bias or offset of the ith unit, and η is a positive constant. Consistent convergence could be obtained with this rule on the problem with the orthogonal set of input training vectors (which was computationally the easiest of the three), in a total number of iterations roughly comparable to standard back-propagation. Convergence could also be obtained on the parity problem, but it was typically more than an order of magnitude slower than in the fastest standard back-propagation trials. Convergence was not generally obtained for the arbitrary bit-string mapping problem.

A second approach was taken in which updates were quantized into three states according to the rule

$$\begin{aligned}\Delta W_{ij} &= \eta \ \mathrm{Sgn}(\delta_i O_j) && (|O_j| \geqslant \varepsilon_1 \text{ and } |\delta_i| \geqslant \varepsilon_2)\\ &= 0 && (|O_j| < \varepsilon_1 \text{ or } |\delta_i| < \varepsilon_2)\\ \Delta B_i &= \eta \ \mathrm{Sgn}(\delta_i) && (|\delta_i| \geqslant \varepsilon_2)\\ &= 0 && (|\delta_i| < \varepsilon_2)\ ,\end{aligned} \qquad (2)$$

where ε_1 and ε_2 are positive constants. By the inequalities in (2), the quantities ε_1 and ε_2 define a "dead zone" for each weight in which no update occurs, as well as regions in which either positive or negative increments are made.

Simulations with this learning rule showed that it is capable of convergence upon all three benchmark problems, for significant ranges of the parameters η, ε_1, and ε_2. Coarse searches were performed for both standard back-propagation and this "trinary" rule to identify parameter values and hidden layer size for which convergence seemed to be both rapid and consistent. Parametric studies were then performed in which the learning rate η and the number of hidden units were varied individually about this point, for both standard and trinary rules. In addition, the parameter ε_2 was varied for the trinary rule, and ε_1 was set to 0.33. The other parameters assumed values in the ranges $\eta = 0.005$ to 0.15 and $\varepsilon_2 = 0.005$ to 0.12 for the trinary rule, and $\eta = 0.005$ to 0.2 for standard back-propagation. A set of five learning trials, each starting from a different initial random weight set, was performed at each set of parameter values. This study demonstrated that the trinary rule converges in significantly fewer iterations than standard back propagation on all three problems. Convergence times for the two rules in the best-performing networks differed by a factor of three to four for the orthogonal input vector and arbitrary bit-string mapping problems, and four to ten for the parity problem.

Over the ranges of parameters studied, more individual instances of non-convergent learning trials occurred for the trinary rule than for standard back-propagation, particularly for large values of the ε_2 or η parameters or in networks with small hidden layers. However, most of these non-convergent cases occurred because the networks reached terminal states in training in which all delta terms fell into the dead zone which specifies zero weight update. This problem is readily circumvented by initiating the learning trial with a smaller value of ε_2, or by reducing ε_2 adaptively during training. This has been demonstrated in further simulations.

We have also investigated in additional simulations the properties of the learning rule subject to some constraints which might be expected in real implementations operating on real data. The response of networks to noisy data has been examined by corrupting the input training patterns with random, zero-mean, normally-distributed additive noise of different variances. It has been found that networks trained with such noise imposed upon the exemplar patterns misclassify patterns with novel noise far less often than do networks trained upon the uncorrupted data. We have also investigated the effects of imposing random offsets upon the delta terms computed by the network. If the delta feedback system were implemented in an analog circuit, such offsets could never be completely eliminated. Results show that the learning rule is tolerant of offsets whose standard deviation is a significant fraction of the ε_2 parameter. Convergence typically slows as the variance is increased, but occurs fairly consistently until the standard deviation reaches values near $\varepsilon_2/2$ or greater. Another constraint which might be expected in analog hardware is soft limiting or saturation of the weights. As a weight grows in magnitude it would become more difficult to make an increment of the same sign as the weight, due, for example, to opposing potential if accumulated charge represented the weight value. We have simulated this effect in conjunction with the trinary rule, and found that convergence is not necessarily precluded or slowed.

Acknowledgments

This work was supported by the Office of Naval Technology under program element 62234N, project RS34M40.

References

[1] Furman, B., & Abidi, A. (1988) Neural Networks 1, Sup. 1, p. 381.

[2] Hinton, G.E., & Sejnowski, T.J. (1986) In Parallel Distributed Processing, Explorations in the Microstructure of Cognition, Vol. 1, pp. 282-317. D.E. Rumelhart & J.L. McClelland, Eds. MIT Press, Cambridge.

[3] Holler, M., Tam, S., Castro, H., & Benson, R. (1989) In Proceedings, International Joint Conference on Neural Networks 1989, Washington, Vol. II, pp. 191-196. IEEE, New York.

[4] Hu, V., Kramer, A., & Ko, P.K. (1988) Neural Networks 1, Sup. 1, p. 385.

[5] Kub, F., Moon, K., & Mack, I. (1989) In Proceedings, International Joint Conference on Neural Networks 1989, Washington, Vol. II, p. 614. IEEE, New York.

[6] Peterson, C., & Hartman, E. (1989) Explorations of the mean field theory learning algorithm. Neural Networks, in press.

[7] Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986) In Parallel Distributed Processing, Explorations in the Microstructure of Cognition, Vol. 1, pp. 318-362. D.E. Rumelhart & J.L. McClelland, Eds. MIT Press, Cambridge.

[8] Schwartz, D.B., Howard, R.E., & Hubbard, W.E. (1989) IEEE Journal of Solid-State Circuits 24, pp. 313-319.

[9] Shoemaker, P.A., & Shimabukuro, R. (1988) Neural Networks 1, Sup. 1, p. 409.

CONNECTIONIST PUSHDOWN AUTOMATA THAT LEARN CONTEXT-FREE GRAMMARS

G.Z. Sun, H.H. Chen, C.L. Giles, Y.C. Lee and D. Chen

Laboratory for Plasma Physics Research,
Department of Physics and Astronomy
and
Institute for Advanced Computer Studies
UNIVERSITY OF MARYLAND,COLLEGE PARK,MD 20742

Recently, much efforts have been made to construct neural network models of sequential machines (1), (2), (3), (4), (5). Most of the existed models use recurrent neural network (or time-delayed feedback network) to simulate the internal state transitions. The basic structure of neural network models of state machine is introduced and studied in reference (5). Using second order connection weights, the recurrent formula of the state transitions can be written

$$S_i^{t+1} = g\Big(\sum_{j,k} W_{ijk} S_j^t I_k^t + \theta_i\Big) \tag{1}$$

where S_i^t is the activity of the i_{th} neuron at time step t, I_k^t is the k_{th} component of input pattern at time step t and g can be *sigmoid* function $g(x) = 1/(1 + exp(-x))$. The task of the model is to train the weights W_{ijk} from the classification information of a given set of training patterns such that the errors would be minimum and the formed state machine can be used as a classifier for new patterns. This process is actually the grammatic inference in terms of neural network methodology. The work done in (5) has shown successes of this model in dealing with regular grammars. The trained neural network state machine can recognize testing sequences of any length correctly.

In this paper we would like to extend this neural network model to deal with context-free grammars, for example , to learn the parenthesis balance checking or the $a^n b^n$ grammar. The difficulty is that, in order to learn context-free grammars, one needs external stack memories in addition to the state transition rules. Otherwise, we can at most train the network to recognize the sequential patterns up to a certain length, and this length is proportional to the network size. Although in practice we have to have a stack of finite length, after the whole system is trained this length should be adjustable in order to fit the different lengths of testing patterns. In this sense a state machine coupled with a stack (i.e. a pushdown automata) can learn context-free grammars, but a state machine itself can not. Although some researchers claimed that they had trained the neural network machines that learned context-free grammars, what they really did was to train the network to learn a subset of a context-free grammar (containing patterns of limited length). After training, the results can not be generalized to arbitrarily long sequences.

An interesting work of William and Zipser (3), using similar technique, tries to couple an external tape to the connectionist state machine to learn a parenthesis balance checking problem. However, in their scheme the transition behavior of the tape and the state machine and their coupling are all given in advance. Therefore, they are only simulating a known Turing machine. No grammatic inference task was attempted. In grammatic inference we do not have informations about the particular transition behavior of the target machine, the only thing we know is the classification information of training patterns.

The purpose of this paper is to develop a neural network model of pushdown automata for the task of grammatic inference. For this purpose, we need to couple a stack to a state machine. The major outstanding problems are: 1) how to construct this stack memory, 2) how to couple this stack memory to the state machine and 3) how to determine the objective function such that its optimization will lead to a self-organization of the entire system based only on the knowledge of classification information of the training examples.

Our the connectionist pushdown automata are composed of two parts: a neural network state machine and a continuous valued stack. The state machine part is an extended version of the finite state machine as in (5). It consists of a group of neurons $S_i, i = 1, 2, ..., N_S$, representing the internal state, another group of neurons $I_i, i = 1, 2, ..., N_I$, as input lines and some other neurons $R_i, i = 1, 2, ..., N_R$, as readings from stack, and still some other neurons $A_i, i = 1, 2, ..., N_A$ representing the output actions (push or pop the stack). The state neurons are connected by one time step delayed feedback weights. At the same time, second order (and also third order for some cases) feedforward connection weights between the state neurons, input symbols, stack readings and output actions would implement the state transition and the stack actions. The recurrent updating formula can be written

$$S_i^{t+1} = g\Big(\sum_{j,k} W^s_{ijk} S^t_j (R^t \oplus I^t)_k + \theta^s_i\Big); \; and \; A_i^{t+1} = f\Big(\sum_{j,k} W^a_{ijk} S^t_j (R^t \oplus I^t)_k + \theta^a_i\Big) \qquad (2)$$

where $(R \oplus I)_k = R_k(if\ 1 \leq k \leq N_R)\ or\ I_k(if\ N_R + 1 \leq k \leq N_I + N_R)$ and $f(x) = 2g(x) - 1$. From this formula it is seen that given initial state S^1, stack reading R^1 and first input symbol I^1, the new state S^2 and action A^2 are generated. After action A^2 is performed on the external stack, we read from the top of stack with a new reading R^2. Using R^2 together with S^2 and I^2, we can repeat the procedure until the end of the input string.

The stack part is simply a normal stack but with continuous value of memory and actions. We will see that this continuity is necessary for the gradient descent learning algorithm. For simplicity, we first consider the simplest action of deterministic pushdown automata. We assume three types of actions so that only one action neuron is needed: *push* ($A > 0$), *pop* ($A < 0$) and *do nothing* ($A = 0$). We also assume that each time we push only the input symbol into the stack. In this case we can use the same representation for stack readings R^t and input symbols I^t (so that $N_R = N_I$).

To illustrate how the continuous stack works, we consider a binary input string "aababe". We use three neurons and unary representation: I=(1,0,0), (0,1,0) and (0,0,1) for string symbols "a", "b" and end symbol "e". Initially, stack is empty, so the reading is $R^1 = (0,0,0)$, and $I^1 = (1,0,0)$ represents the first input symbol "a". Suppose that we use four neurons to represent internal states and assign initial state to be $S^1 = (1,0,0,0)$. After an iteration of Eq.(2), new state S^2 and new action A^2 are obtained. If the action output is $A^2 = 0.6$, we push a symbol "a" with length=0.6 into the stack. It can be represented as $(0.6,0,0)$ and the next reading R^2 would be $(0.6,0,0)$. If $A = -0.3$ it pops an empty stack. When this happens we need an learning algorithm to deal with it. After several pushes and pops, the stack memory stores the continuous symbols and may look like (from bottom to top): (0.32,0,0), (0.2,0,0), (0,0.7,0) and (0.4,0,0). Each time we read the stack from the top with depth=1. So, the next reading would be $R^t = (0.4, 0.6, 0)$. If the action $A^{t+1} = -0.86$, we pop (0.4,0,0) and (0,0.46,0) from the stack, this leaves the stack with (0.32,0,0), (0.2,0,0) and (0,0.24,0). The next reading should be $R^{t+1} = (0.52, 0.24, 0)$ and so on.

The design of this model is mainly determined by the following considerations. Firstly, in the limit of saturation (neuron outputs approach 1 and 0, or 1 and -1 for action neuron), this continuous stack should approach the normal discrete stack behavior. Secondly, before the saturation limit, the analog values of representations can have a meaningful probabilistic interpretation. For example, $R^t = (0.1, 0.89, 0.01)$ means that the reading from the top of the stack at time t is not one discrete symbol, the probabilities to be symbols (1,0,0),(0,1,0) and (0,0,1) are 0.1, 0.89 and 0.01 respectively. When the stack length is less than 1, the reading may be $R^t = (0.1, 0, 0)$, this means that the probability to be (1,0,0) is 0.1 and the probability to read empty stack is 0.9. Thirdly, whenever the connection weights W_{ijk} have an infinitesimal change, the changes of states, stack storage, actions and stack readings should all be infinitesimal. This continuity is a crucial condition for the gradient descent learning algorithm to be meaningful and is also the main reason for us to employ the continuous stack instead of the discrete stack. It is seen from the above model of stack that all of these requirements are satisfied.

For optimization, the objective function we chose is the scalar error measure of the end state and the stack length. According to the theory of pushdown automata, either the end state or the stack length alone can be sufficient criterion to determine the acceptance of input strings. We used the combination of the two in the hope of speeding up the learning process. We tried different error measure functions. One of those that numerically work well is defined as

$$er = \begin{cases} 0 & \text{if } target = 0 \text{ and } S^T_{N_S} \geq L^T \\ target - (S^T_{N_S} - L^T) & otherwise; \end{cases} \tag{3}$$

where *target* = 1 for legal string and *target* = 0 for illegal string, $S^T_{N_S}$ and L^T are respectively the output value of the $(N_S)_{th}$ state neuron and the total stack length at time T (the end of input string). L^T can be evaluated recursively through

$$L^{t+1} = L^t + A^t. \tag{4}$$

The total error function to be minimized is the summation of er^2 over all training patterns. In testing or classification later on, the quantity $v = S^T_{N_S} - L^T$ will be evaluated at the end of each sequence (usually we feed an end symbol to the network), an input pattern is classified as legal string if $v > 0.5$, otherwise illegal. It can be seen that for legal patterns (target=1) minimizing the error will lead to the case where the last state neuron output to be 1 and the stack length to be zero. Similarly, for illegal patterns, minimizing the error will decrease the state neuron output and increase the stack length, but if the value of v is already negative, we do not need to make corrections.

By imposing the "on-line" learning algorithm the weight perturbations (predicted errors) are propagated forward (stored in a stack-like matrix memory) though the recurrent formula and the final correction is made at the end of input patterns. These recurrent relations can be derived in terms of chain rule of the derivative of the error function. They are

$$\frac{\partial S^{t+1}_{i'}}{\partial W_{ijk}} = h_{i'}(S^{t+1}_{i'})\{\delta_{ii'} S^t_j (R^t \oplus I^t)_k + \sum_{j'=1}^{N_S} \sum_{k'=1}^{2N_I} W_{i'j'k'} (R^t \oplus I^t)_{k'} \frac{\partial S^t_{j'}}{\partial W_{ijk}} + \sum_{j'=1}^{N_S} \sum_{k'=1}^{N_I} W_{i'j'k'} S^t_{j'} \frac{\partial R^t_{k'}}{\partial W_{ijk}}\} \tag{5}$$

where for simplicity we combined the notations of $S^t_{i'}$ and A^t in one, *i.e.* the $(N_S+1)_{th}$ component of S^t is A^t, so that function $h_i(x)$ represents derivatives $g'(x)$ for i=1 to N_S and $f'(x)$ for $i = N_S+1$; similarly we combined W^s and W^a in one such that W_{ijk} represents W^s_{ijk} for i=1 to N_S and W^a_{jk} for $i = N_S+1$.(note that we assumed $N_A = 1$ and $N_R = N_I$). Up to now the recursion is not complete until we express $\frac{\partial R}{\partial W}$ by $\frac{\partial S}{\partial W}$. Since the current stack reading depends on the whole history of the stack, no simple recurrent relation can be found. But, after some trainings when action values are large enough (> 0.5 for example), each reading R^t may not contain much information of the past. To the first order approximation, we write

$$\frac{\partial R^t_{k'}}{\partial Wijk} = \sum_{t'=1}^{t} \frac{\partial R^t_{k'}}{\partial A^{t'}} \frac{\partial A^{t'}}{\partial Wijk} \simeq \frac{\partial R^t_{k'}}{\partial A^t} \frac{\partial A^t}{\partial Wijk} = (\delta_{k' r^t_1} - \delta_{k' r^t_2}) \frac{\partial A^t}{\partial Wijk} \tag{6}$$

where r^t_1 and r^t_2 are the ordinal numbers of neurons that represent the top and the bottom symbols respectively in reading R^t, for example, after the execution of the action A^t the stack is (from bottom to top): (0.2,0,0), (0,0.7,0) and (0,0,0.15), then we have $r^t_1 = 3$ and $r^t_2 = 1$. Now, the weights update formula is

$$\Delta W_{ijk} = \eta \cdot er \cdot \left(\frac{\partial S^T_{N_S}}{\partial W_{ijk}} - \frac{\partial L^T}{\partial W_{ijk}}\right) \tag{7}$$

At last, to treat the case of popping empty stack, we make correction of weights to increase the stack length if the input pattern is legal, otherwise we do not bother.

We have simulated the neural network pushdown automata numerically to learn several context-free grammars. The results are encouraging. For the parenthesis balance checking, we used three state neurons. The fifty training patterns include all possible strings up to length four and some longer strings up to length eight. After twenty sweeps of training, the network formed a perfect pushdown automaton. This can be seen either from the state and stack analysis or from the numerical test with novel patterns, no recognition errors have been found for all possible strings up to length twenty and some strings of length around one hundred. The basic behavior of this pushdown automata is that the state part do not need to do anything, only one state is needed in addition to the start state (1,0,0) and end states. The learned transition diagram is shown in *Fig*.1, where the notation (a,b,c) beside the arrows indicates that this transition occurs when the input symbol is "a", the reading from the stack is "b" and the action neuron has output "c".

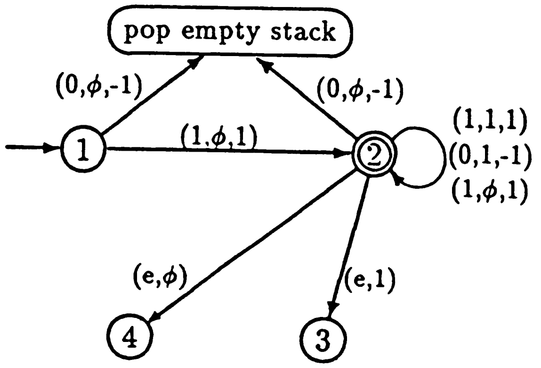


Fig. 1. Learned neural network pushdown automaton for parenthesis balance checking where the numerical results for states (1), (2), (3) and (4) are (1,0,0), (0.9,0.2,0.2), (0.89,0.17,0.48) and (0.79,0.25,0.70). State (1) is start state. State (4) is legal end state. Before feeding end symbol a legal string must end at state (2) with empty stack

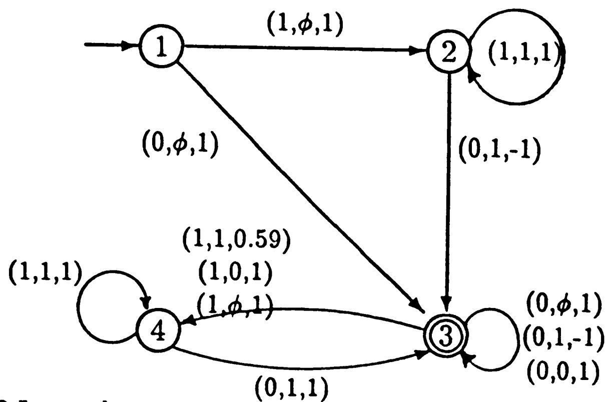


Fig.2.Learned neural network pushdown automaton for 1^n0^n grammar, where the four states (1), (2), (3) and (4) are (1, 0, 0, 0), (0.96, 0.10, 0.12, 0.16, 0.68), (0.99, 0.99, 1.0, 1.0, 0.99), (0.05, 0.01, 0.01, 0.60, 0.92). End states are not shown. Before feeding end symbol, state (3) is the right final state for legal string.

We also successfully trained 1^n 0^n grammar. Five state neurons are used. We first chose 27 short strings to form the training set. After 100 sweeps of training we tested and found 6 recognition errors up to length eight. We added these patterns to the training set and trained another 100 sweeps. Then we tested again and found 8 errors for up to length nine. After this procedure had been repeated five times, the neural network pushdown automata was almost prefect. We tested all 2097150 patterns up to length twenty and some patterns up to length 160 and found no recognition errors. Using quantization of the neuron output values, a perfect pushdown automata can be seen and the transition diagram is shown in *Fig*.2.

Further numerical and analytical studies are being undertaken. We believe that this successful model of pushdown automata demonstrated the power of connectionist strategy.

Acknowledgement
This work is supported by AFOSR and NSF.

References

(1). Robert B. Allen,*Adaptive Training for Connectionist State Machines*, paper presented at ACM Computer Science Conference, Louisville, Feb. 1989.

(2). M.I. Jordan, *Attractor Dynamics and Parallelism in a Connectionist Sequential Machine*, Proceedings of the Cognitive Science Society, 531-546, Amherst, Aug. 1986.

(3). R.J. William and D. Zipser, *A Learning Algorithm for Continually Running Fully Recurrent Neural Networks*, ICS Report 8805, Oct. 1988.

(4). D. Servan-Schreiber, A. Cleeremans and J.L. McClelland, *Encoding Sequential Structure in Simple Recurrent Networks*, paper presented at IEEE Conference on Neural Information Processing System, Denver, Colorado(1988).

(5). Y.D. Liu, G.Z. Sun, H.H. Chen, C.L. Giles and Y.C. Lee, *Inductive Inference and Neural Network State Machine*, submitted to IEEE Conference on Neural Information Processing System, Denver, Colorado(1989). See also the paper *Grammatic Inference and Neural Network State Machine* in IJCNN-90-Washington D.C..

MULTIPLE THRESHOLD PERCEPTRON USING GAUSSIAN FUNCTION

Kaveh Ashenayi*, Heng-Ming Tai*, Mohammad R. Sayeh**, Mohammad T. Mostafavi***
Department of Electrical Engineering
*The University of Tulsa
Tulsa, Ok 74104-3189
**Southern Illinois University at Carbondale
Carbondale, IL 62901-6603
***Department of Computer Science
The University of North Carolina at Charlotte
Charllote, North Carolina 28223

ABSTRACT

A new multi-threshold perceptron capable of handling both binary and analog input is presented and discussed. The modified perceptron replaces the sigmoid function with the Gaussian function. The modified perceptron is used to solve the XOR problem. It requires fewer number of iterations to converge to a solution than that of a multi-layer network using back propagation.

INTRODUCTION

An artificial Neural Network (NN) is a computational structure based upon simplified models of human brain [1]. The hallmark of NN is massive parallelism and interconnectivity between a large number of relatively simple processing units, often called neurons. Neural networks have been utilized to solve problems ranging from pattern recognition/classification to security assessment for electric power systems [2-4].

The single-layer perceptron is one of the first NN developed. It is capable of handling both binary and analog inputs [5]. The original perceptron convergence theorem and weight adjustment procedures were developed by Rosenblatt (see reference 5). It was shown that using the procedure developed by Rosenblatt single-layer perceptron can only classify linearly separable input patterns [6]. Therefore, problems such as XOR problem where the input patterns can not be classified into two distinct groups can not be solved using single-layer perceptron [7,8]. This limits utility of single-layer perceptron. One possible solution is to use multi-layer perceptrons.

Using multi-layer perceptron architecture and "Back Propagation (BP)" [9] many shortcomings of the single-layer perceptron can be overcome. This procedure is effective and allows for efficient use of multi-layer perceptrons. But the procedure does not guarantee convergence to the global minima at all times. Also, it requires a large number of training iterations in order to converge (see reference 9).

Because of the problems associated with BP [10] it is of interest to modify the weight adjustment procedure and/or the model developed by Rosenblatt to enable single-layer perceptron to solve problems such as XOR problem. In this paper a modified perceptron is presented. The modified perceptron is a multiple threshold perceptron which is capable of solving problems such as XOR problem. Unlike the previous efforts [11] in developing multiple threshold perceptron our perceptron is capable of handling both binary and analog input. Also, the procedure requires fewer number of iterations (compared to BP) and based on our experience it always converges to global minima.

Proposed Modified Perceptron

As it was stated a single-layer perceptron utilizing Rosenblatt's procedure is only capable of classifying input patterns into one of two categories. Therefore, an alternate approach is needed if the input patterns are as shown in Figure 1.

As is evident, if we are to correctly classify patterns of Figure 1, we require at least two hyperplanes in order to divide the two dimensional space of Figure 1 into three distinct regions. Therefore, the modified perceptron must be able to form two decision boundaries (see Equation 1). Figure 2 illustrates decision boundaries in the two-dimensional space.

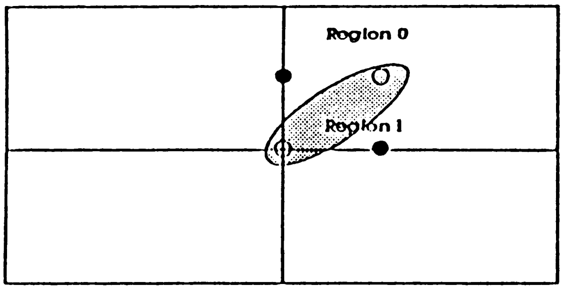


Figure 1. Example of Input Patterns That are not Linearly Separable

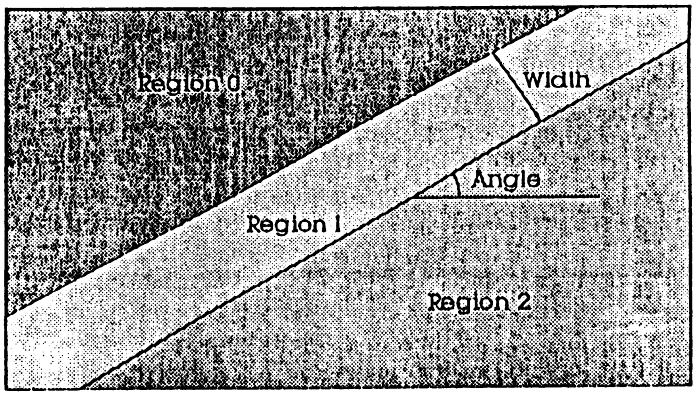


Figure 2. Decision Boundaries Required

$$y_1 = \theta_1/W_2 - (W_1/W_2)\,X$$
$$y_2 = \theta_2/W_2 - (W_1/W_2)\,X \qquad (1)$$

where W_i represents weight, θ_i represents thresholds of function, and X represents the input pattern

Note that both decision boundaries have the same slope. However, they have different intersection points with the vertical axis.

Two values are very useful when studying a perceptron with two hyperplanes. These are width and angle (see Figure 2). These variables provide a measure of performance that can be used for comparison purposes. Slope of the decision boundaries is equal to tangent of "angle" (see Equation 2). When W_2 is negative we must add 180^o to the value obtained.

$$\text{angle} = \tan^{-1}(W_1/W_2) \qquad (2)$$

The variable "width" represents the distance between the two decision boundaries. Using simple geometry (see Figure 2) it can be shown that the variable width is equal to

$$\text{width} = (\theta_2 - \theta_1)/(W_1^2 + W_2^2)^{0.5} \qquad (3)$$

There are a number of functions that satisfy the requirements specified. Gaussian, double Sigmoid, and some piecewise linear functions all satisfy these requirements. The double sigmoid function correctly forms the required decision boundaries. However, it is unstable when the input patterns are not symmetric about a hyperplane passing through the origin. Figure 3 shows both symmetrical and non-symmetrical patterns in the two dimensional space. The piecewise linear functions can not be used because their derivative is not defined for all values of input.

The Gaussian function presents the best choice in that its derivative is defined for any possible value of input. Also, it is capable of correctly classifying patterns that are not symmetric about a hyperplane passing through the origin.

The modified procedure utilizes the Gaussian function f_G, given below, in place of the sigmoid function used by Rosenblatt.

$$f_G(\alpha-\Theta) = \{1/[\sigma(2\pi)^{0.5}]\}\{EXP[-0.5[(\alpha-\mu)/\sigma]^2-\Theta]\} \qquad (4)$$

where

σ = variance of the Gaussian function and μ = mean of the Gaussian function

After each iteration the weights are adjusted as follows

$$w(t+1) = \begin{cases} w(t) + \eta(d - y(t))x(t) & \text{if slope of } f_G \text{ is positive} \\ w(t) - \eta(d - y(t))x(t) & \text{if slope of } f_G \text{ is negative} \end{cases} \quad (5)$$

SIMULATION RESULTS

A computer program was developed to simulate the behavior of the modified perceptron. Results of the simulations indicate that use of Gaussian function as the non-linearity function allows the single-layer perceptron to correctly classify three distinct classes of input patterns.

Using learning rate (η) of 0.5 both BP and the modified perceptron were used to solve the XOR problem. The modified perceptron converged to the global minima in less than 60 iterations while BP required more than 550 iterations.

Figures 4 and 5 illustrate results obtained using the simulation program. The program was used to simulate a network utilizing the modified perceptron to solve the XOR problem of Figure 1. Figure 4 demonstrates the number of iterations required as a function of σ; and Figure 5 shows variations in the number of iterations required as a function of η. As is evident the network is capable of solving the XOR problem more efficiently compared to BP.

REFERENCES

1. S. Borman, "Neural Network Applications in Chemistry Begin to Appear," Chemical & Engineering News, pp. 24 - 28, April 24, 1989.

2. K. Ashenayi, S. Singh, and M. R. Sayeh, "Pattern Classification Using Associative Memory," paper presented at 31 St. Midwest Symposium on Circuits and Systems, Aug. 11-12, 1988.

3. H. M. Tai and T. -L. Jong, "Information Storage in High-Order Associative Memory With Unequal Neural Activity," paper presented at 1989 International Joint Conference on Neural Networks, Washington DC, June 18-22, 1989.

4. D. J. Sobajic and Y. H. Pao, "Artificial Neural-Net Based Dynamic Security Assessment for Electric Power Systems," IEEE Transactions on Power Systems, Vol. 4, No. 1, Feb. 1989, pp. 220 - 228.

5. R. Rosenblatt, Principles of Neurodynamics, Spartan Books, New York, 1959.

6. H. D. Block, "The Perceptron: a Model for Brain Functioning. I," Reviews of Modern Physics, 34, pp. 123 - 135, 1962.

7. M. L. Minsky and S. A. Papert, Perceptrons, Expanded Edition, MIT Press, Cambridge, Mass., 1988.

8. R. P. Lippmann, "An Introduction to Computing With Neural Nets," IEEE ASSP Magazine, pp. 4 - 22, April 1987.

9. D. E. Rumelhart, J. L. McClelland, and The PDP Research Group, Parallel Distributed Processing Explorations in the Microstructures of Cognition Vol. 1: Foundations, MIT Press, Cambridge, Mass., 1988.

10. M. L. Brady, R. Raghavan, and J. Slawny, "Back Propagation Fails to Separate Where Perceptron Succeed," IEEE Transactions on Circuits and Systems, Vol. 36, No. 5, pp. 665 - 674, May 1989.

11. R. Takiyama, "Multiple Threshold Perceptron," Pattern Recognition, Vol. 10, pp. 27 - 30, 1978.

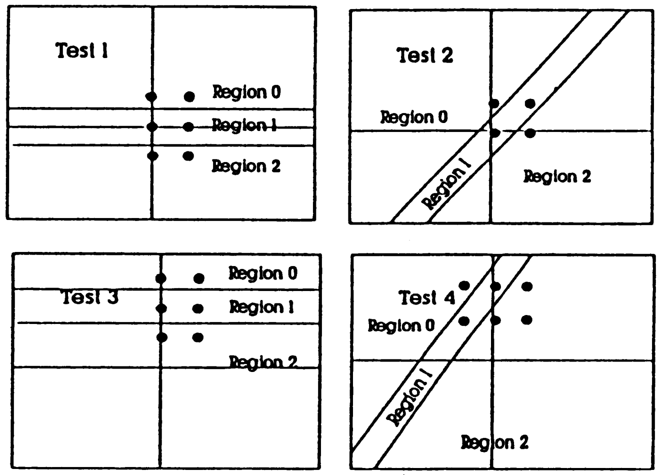


Figure 3. Symmetric and Non-Symmetric Input Patterns

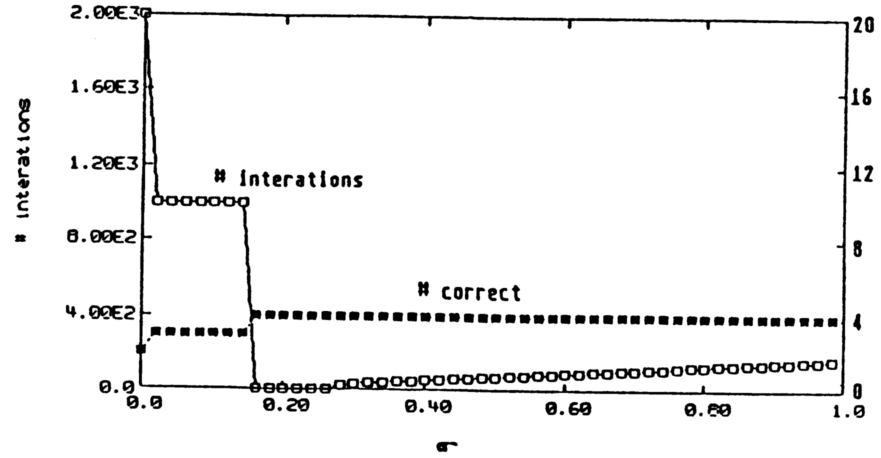


Figure 4. Number of Iterations Required to Solve XOR Problem as a Function of σ

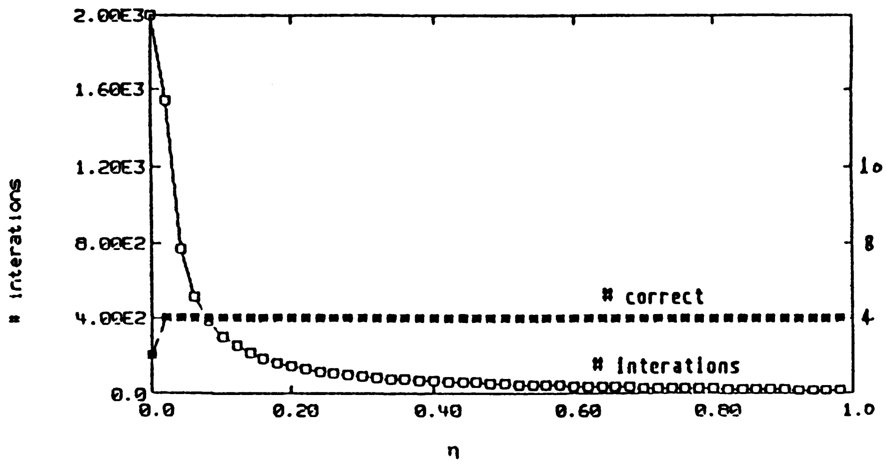


Figure 5. Number of Iterations Required to Solve XOR Problem as a Function of η

A Hybrid Algorithm for Finding the Global Minimum of Error Function of Neural Networks

Norio Baba
Faculty of Engineering, Tokushima University, 770, JAPAN

Abstract: Recently, back-propagation method has often been applied to adapt artificial neural networks for various actual pattern classification problems. However, one of the most important limitations of this method is that it sometimes falls into a local minimum of the total error function of neural network.

In this paper, a hybrid algorithm which combines the back-propagation method with the random optimization method of Matyas (its modified algorithm) is proposed to learn the weights and parameters involved in a neural network. It is shown by several computer simulation results that the proposed hybrid algorithm can be successfully utilized in order to find the global minimum of the total error function of neural networks in a comparatively small number of steps.

1. INTRODUCTION

In recent years, neural network computing has been studied quite extensively by many researchers and various fruitful results have been obtained. In particular, the back-propagation method (BP method) proposed by Rumelhart et al is one of the most stimulating products and has given great impact to the development of this area.

However, this method has also several problems to be solved. Among those, the following problem might be particularly important: "It sometimes falls into a local minimum of the total error function."

In the first part of this paper, we propose a hybrid algorithm which combines the BP method with the random optimization method of Matyas (its modified random optimization method) in order to find the global minimum of the total error function in a small number of steps.

In the latter half of this paper, several computer simulation results using this hybrid algorithm for pattern classification problems are given.

2. HYBRID ALGORITHM FOR FINDING A GLOBAL MINIMUM OF THE TOTAL ERROR FUNCTION

In this section, we propose a new combined algorithm of the BP method and the random optimization method in order to find a global minimum of the total error function $E(w)$ in a small number of steps.

First, let us briefly refer to the BP method and the random optimization method.

2.1 BACK-PROPAGATION METHOD

Recently, Rumelhart and McClelland [1] have proposed the BP method as a new algorithm for finding weights of multi-layered network. This method has attracted great numbers of researchers working in the field of computer, artificial intelligence, and neuro-science. It has been applied to various interesting actual porblems. However, the BP method has several problems to be solved. One of the most important problems is the potential for falling into a local minimum of the total error function $E(w)$.

2.2 RANDOM OPTIMIZATION METHOD

It is well known that the random optimization method of Matyas [2] ensures convergence to a global minimum with probability 1 on the compact set [3],[4],[5]. This algorithm can be described as follows:

STEP 1. Select an initial point $w^{(0)}$ in the search domain X and let $k = 0$. Let M be the total number of steps.

STEP 2. Generate Gaussian random vector $\xi^{(k)}$. If $w^{(k)} + \xi^{(k)} \in X$, go to STEP 3. Otherwise, go to STEP 4.

STEP 3. If $E(w^{(k)} + \xi^{(k)}) < E(w^{(k)})$, let $w^{(k+1)} = w^{(k)} + \xi^{(k)}$.
If $E(w^{(k)} + \xi^{(k)}) \geqq E(w^{(k)})$, let $w^{(k+1)} = w^{(k)}$.

STEP 4. If $k = M$, stop the total calculation. If $k < M$, let $k = k+1$ and go to STEP 2.

In 1981, Solis & Wets [5] proposed the modified random optimization method in order to find a global minimum of the objective function in a small number of steps. Since this method differs from the original random optimization method only in STEP 3, we abbreviate another part of steps.

MODIFIED RANDOM OPTIMIZATION METHOD (STEP 3)

(i) If $E(w^{(k)} + \xi^{(k)}) < E(w^{(k)})$,

let $w^{(k+1)} = w^{(k)} + \xi^{(k)}$ and $b^{(k+1)} = 0.4\xi^{(k)} + 0.2b^{(k)}$.

(ii) If $E(w^{(k)} + \xi^{(k)}) \geqq E(w^{(k)})$,

let $w^{(k+1)} = w^{(k)} - \xi^{(k)}$ and $b^{(k+1)} = b^{(k)} - 0.4\xi^{(k)}$.

Otherwise, let $w^{(k+1)} = w^{(k)}$ and $b^{(k+1)} = 0.5b^{(k)}$ ($b^{(0)} = 0$)

2.3 HYBRID ALGORITHM

We shall propose a new hybrid algorithm that makes use of both the merits of the random optimization method and the back-propagation method. An outline of this hybrid algorithm is described in Figure 1. Let us explain it briefly. First, parameter training is carried out using the BP method. When the decrease of the value of the total error function becomes smaller than a specified value $\varepsilon 1$, we change the overall descent algorithm from the BP method to the random optimization method of Matyas (modified random optimization method) in order to prevent it from falling in to a local minimum of E(w). If the decrease of the value of the total error function becomes larger than a specified value $E(w^{(k)})G$, we change the overall descent algorithm from the random optimization method to the BP method. The same procedures are repeated several times. When the total number of steps exceeds a specified number M, the overall calculation is stopped. [6]

3. COMPUTER SIMULATION RESULTS

In this section, two computer simulation results are presented in order to demonstrate the effectiveness of our proposed hybrid algorithm.

Example 1: In this example, we consider a rather simple pattern classification problem in which 64 training patterns consisting of all of the possible combinations of "0" and "1" are the inputs of the neural network as shown

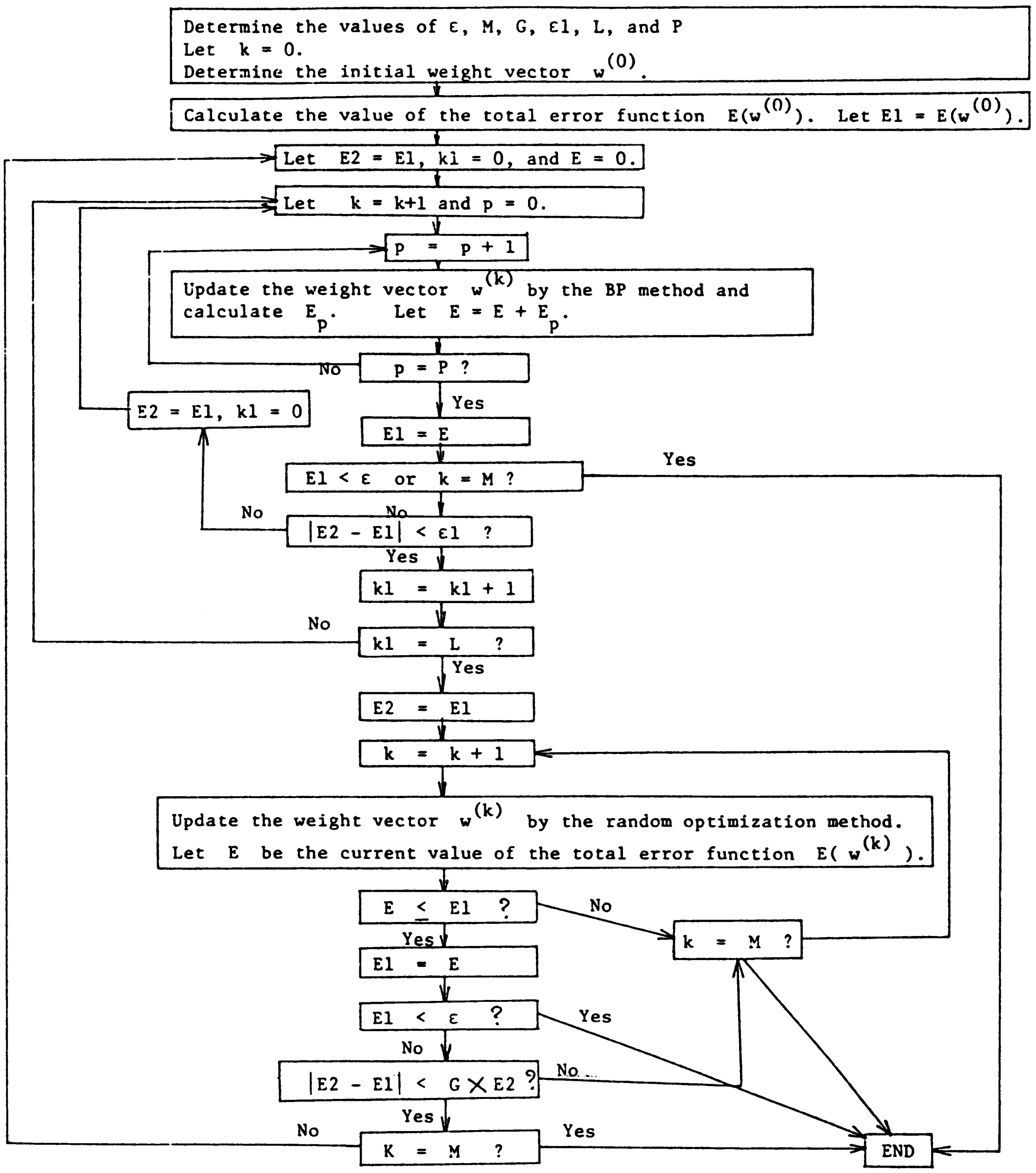

Determine the values of ε, M, G, ε1, L, and P
Let k = 0.
Determine the initial weight vector w^(0).
Calculate the value of the total error function E(w^(0)). Let E1 = E(w^(0)).
Let E2 = E1, k1 = 0, and E = 0.
Let k = k+1 and p = 0.
p = p + 1
Update the weight vector w^(k) by the BP method and calculate E_p. Let E = E + E_p.
p = P ?
No
Yes
E2 = E1, k1 = 0
E1 = E
E1 < ε or k = M ?
Yes
No
|E2 - E1| < ε1 ?
Yes
k1 = k1 + 1
k1 = L ?
E2 = E1
k = k + 1
Update the weight vector w^(k) by the random optimization method.
Let E be the current value of the total error function E(w^(k)).
E ≤ E1 ?
k = M ?
E1 = E
E1 < ε ?
|E2 - E1| < G × E2 ?
K = M ?
END

Figure 1

in Figure 2. The objective in this example is to train weights of the neural network in such a way that the network responds with the output "1" when the numbers of the input are even or zero or else it responds with the output "0". Here, the value 0.1 and 0.5 have been used for parameter η of the BP method and the value 0.01 and 0.005 have been used for the variance of ξ of the random optimization method. Figure 3 shows the computer simulation results.

Example 2: In this simulation, we consider prediction of SO_2 density at noon in Tokyo using the informations obtained at 10 a.m. In particular, our objective is to construct a neural network which emits output "1" (alarm) when the SO_2 density exceeds 8 pphm and emits "0" when the SO_2 density does not exceed 8 pphm. In order to construct such a neural network, we have carried out parameter training using the data obtained over the previous two weeks in order to forecast SO_2 density daily for one week in the future. Since we have not

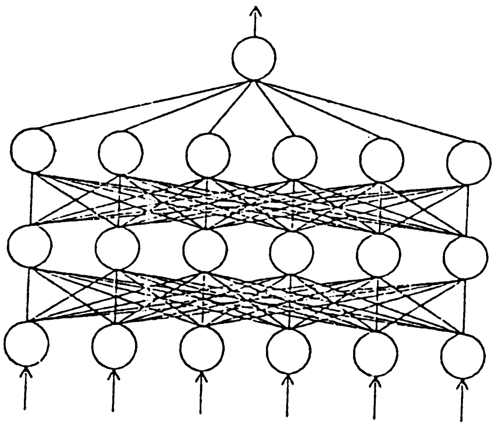


Figure 2

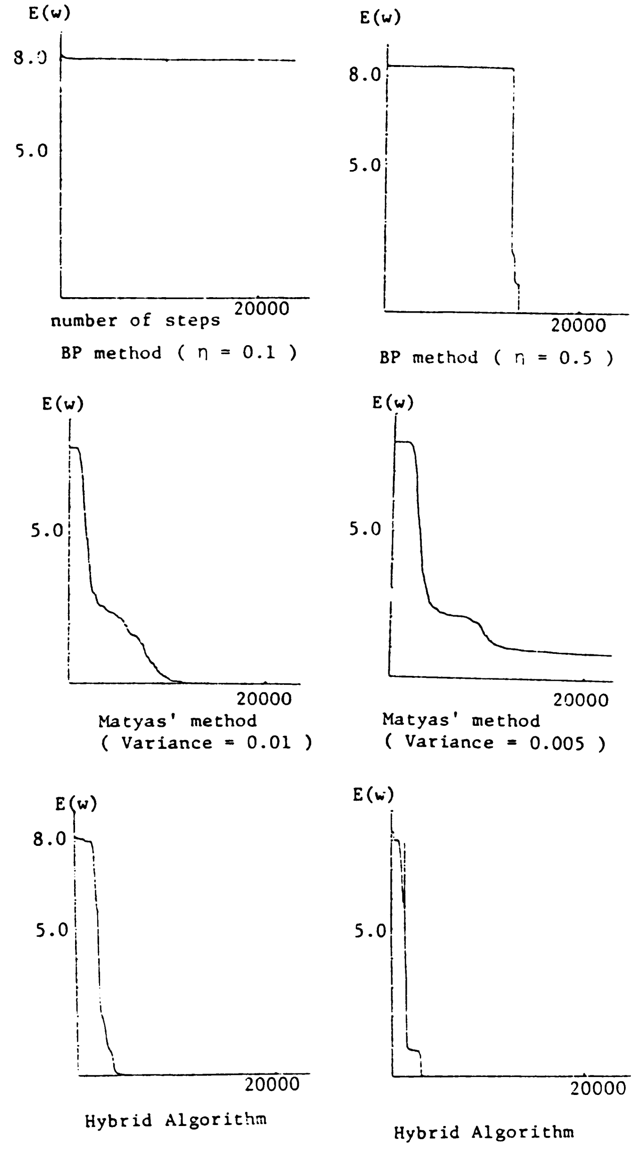


Figure 3

enough space in this paper, we abbreviate details of the simulation results. They will be presented in my talk at the conference.

Acknowledgement

The author would like to thank Mr. T. Totori for his kind assistance in preparing the manuscript.

References:

[1] D.E. Rumelhart and J.L. McClelland, Editors, Parallel Distributed Processing, MIT Press, 1986.
[2] J. Matyas, Automation & Remote Control, Vol. 26, pp. 246-253, 1965.
[3] N. Baba et al, Information Sciences, Vol. 13, pp. 159-166, 1977.
[4] N. Baba, JOTA, Vol. 33, pp. 451-461, 1981.
[5] F.J. Solis & J.B. Wets, Mathematics of Operations Research, Vol. 6, pp. 19-31, 1981.
[6] N. Baba, Int. J. Control, Vol. 37, pp. 929-942, 1983.
[7] N. Baba, Neural Networks, Vol. 2, 1989 (to be published)

AUTOMATIC EVOLUTION OF NEURAL NET ARCHITECTURES

A.W. Bailey
Physical Sciences Inc.
Research Park, P.O. Box 3100
Andover, MA 01810

ABSTRACT

Algorithms have been developed for self-constructing feed-forward neural nets. The network produced in an example problem is compared with a conventionally constructed net. A fast back-propagation training algorithm is discussed.

1. INTRODUCTION

Conventional neutral nets use an architecture that is static. Modification of the net during training is limited to alteration of the synaptic connection weights and the neuron thresholds. While it has long been noted by workers in automata theory that neural nets can, in principle, be self-constructing,[1] in practice design of an optimal net configuration for a given problem has been a trial and error process. In a back propagation feed-forward net, a number of layers are chosen, each with a number of neurons and with some degree of synaptic interconnection. Typically, a number of neurons are assigned to three or four layers and each neuron output is fed to all neurons in forward layers. If the configuration proves inadequate, a new configuration with more hidden layer neurons is tried. For small problems with a relatively few inputs and a modest number of neurons, this trial and error procedure might be acceptable. For complex problems, this procedure is unacceptable. For example, machine vision with high-resolution video would involve hundreds of thousands of inputs. Full synaptic connection to all forward neurons would require billions of synaptic connections. The synaptic connection tree must be limited to a more reasonable size. The development of an optimal net can be automated and incorporated as part of the training procedure of the net, thereby avoiding a trial and error procedure.

A neural net can be grown by adding and removing synapses and neurons until the net works as desired. This technique allows optimized neutral nets to be created whose complexity is matched to the nature and amount of the training data. One technique for doing this has been described by Tenorio and Lee.[2] Herein is described a different, independently formulated, technique.

2. THEORETICAL APPROACH

It is assumed that we are dealing with a conventional, non-time dependent, feed-forward neural net. Two terms can be defined to guide the evolution of the net. These will be called the synapse "importance,"

$$I_{ki} = \frac{1}{2} \sum_{P} \sum_{D} \left(\frac{\partial D_{pj}}{\partial W_{ki}} \right)^2$$

and the synapse "effectiveness,"

$$\varepsilon_{ki} = \sum_{p} \frac{\partial}{\partial W_{ki}} \left(\sum_{j} D_{pj}^{2} \right) ,$$

Important synapses are those that strongly affect the pattern errors, D_{pj}, when their weights are altered. Effective synapses are those that strongly alter the global error when their weights are altered. An important synapse may not be effective due to cancellation of terms: increasing the synapse weight may decrease the absolute value of some deviations and increase the absolute value of others. Effective synapses will be the ones that are most effective for training the net and thus are desirable. Synapses that are both ineffective and unimportant are candidates for removal from the net if this can be done without an excessive degradation of the training state of the net resulting in a large increase in the total error. The increase in the global error from an alteration of the net by adding or removing components will be referred to as the "trauma". These new parameters, "importance", "effectiveness", and "trauma", provide the basis for the construction of neural net evolution algorithms.

A synapse would be a desirable addition if it would have a large effectiveness and so aid in training the matrix to its desired state. The effectiveness is well defined and non-zero for synaptic connections between existing neurons even when those connections are not currently part of the net. These "virtual" synapses are candidate for inclusion in the net.

A synaptic connection with a high importance but a low effectiveness is one for which an alteration of the weight lowers the error of some patterns but increases it for others. Such a situation implies that correlation of the signal with that from another important synaptic connection could produce a significant correlation.

A synapse is a candidate for deletion if both its importance and its effectiveness are low and the trauma to the net is not excessive. The trauma to the net can be minimized by increasing the self-activation of forward neurons to reflect the average input signal from the deleted synapse.

A neuron is ineffective as a switch when it is either always on or always off, regardless of the input pattern. In this case, the importance and effectiveness of the synapses leading to the neuron will become small and they can be deleted. The neuron will then have a constant output that is set purely by the neuron self-activation, regardless of input pattern. The neuron can be deleted and the lost current compensated by altering the self activation of the forward neurons.

3. FAST BACK PROPAGATION TRAINING TECHNIQUES

It has been noted by researchers that training can be accelerated by associating a "momentum" with the changes in the weight coefficients. This allows the direction of change of the weights to cancel where different patterns direct the weight changes in different directions and reinforce in the direction of common improvement.[3] This technique reduces the time to train an XOR from many thousands of iterations down to a few hundred.

This concept can be expanded by forming an analogy to a damped oscillator:

$$\ddot{w} + b\dot{w} + kw = 0 \quad .$$

The weight, w, is subject to a damping force, $b\dot{w}$ proportional to its velocity and to a restoring (learning) force, kw, proportional to the difference between the best estimate of the proper weight and the current weight. In order to average over all the values of a training set (i.e., a low-pass filter), we must have $b < 1/n$, where n is the number of distinct patterns in the training set. $b = 1/(2n)$ is a good choice and agrees well with the observation that retaining 90 percent of the old momentum helps to efficiently train an XOR gate $(n = 4)$. The restoring force, kw, should be set to be slightly higher than critical damping, $k = (b/2)^2$, for fastest convergence. Using this technique it has been possible to reduce the time to train an XOR down to as little as 74 training set iterations. This compares favorably with much more computationally expensive schemes such as quasi-Newton methods.[4]

4. EXAMPLE PROBLEM

A simple trial was conducted of these algorithms by growing a neural net to map the digits 0 through 9 as described by the seven segment calculator display to the binary representations of the numbers (Figure 1). A simple FORTRAN neural net code was written for the test, using fast back propagation and computing the importance and effectiveness of the synaptic links. After every 50 iterations of the training set, the state of the net was investigated and neurons and synaptic links were added or deleted based on the importance, effectiveness, and trauma of the alterations.

The simulation was started with no hidden units and the input and outputs fully connected. The simulation ended with a fully trained four layer neural net with 7 hidden units and 73 synaptic links, as shown in Figure 2. The second layer grew first, then the third. If this layer by layer growth proves to be the rule, it may imply easier training for multilayer nets, as the layer nearest the outputs trains fastest. The system converged to a global error of below 0.01 after a total of 604 iterations of the training set.

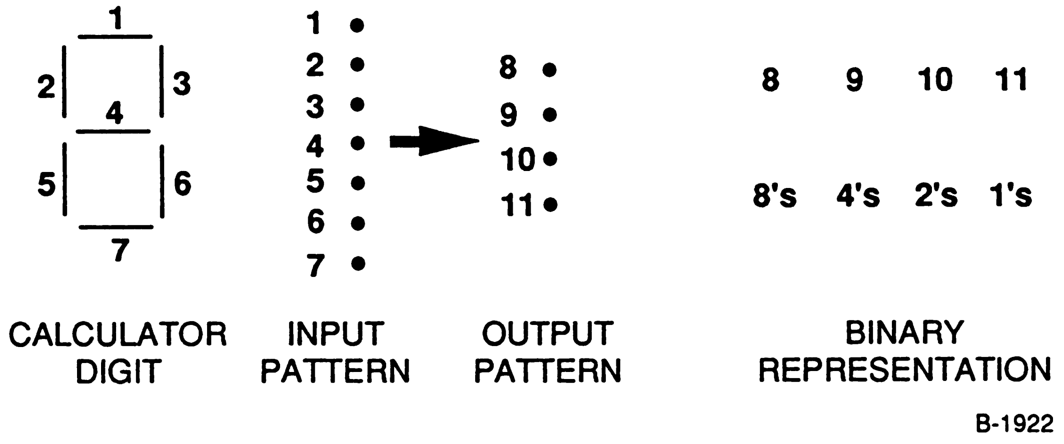


Figure 1. The Seven Lines Used to Represent a Digit on a Hand Calculator Are Used as Inputs to the Net Which Produces the Binary Representation of That Number

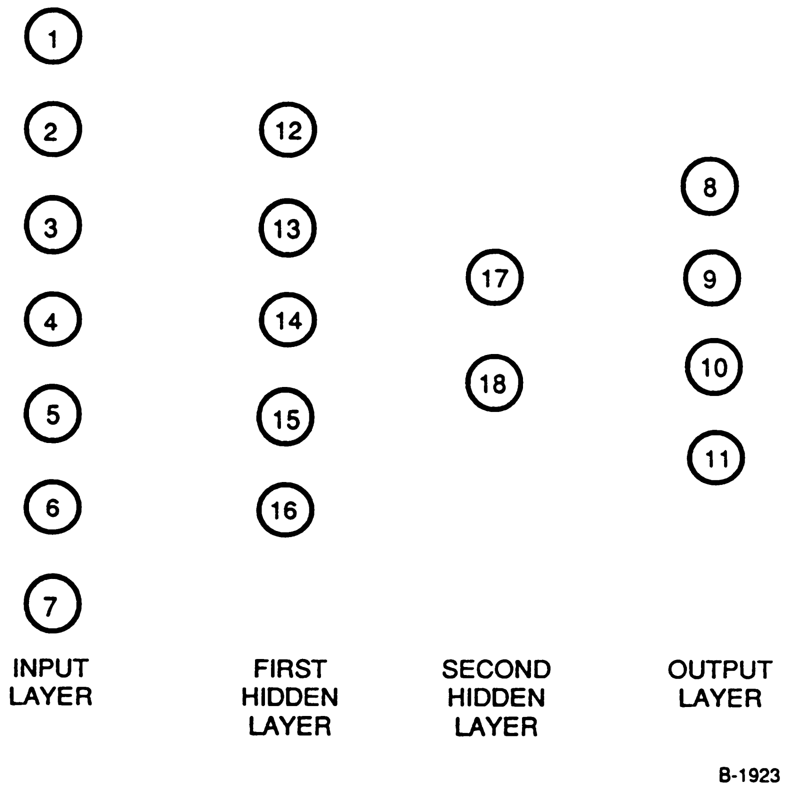


Figure 2. Final Net Configuration. Neuron numbers are assigned in order of their creation.

A fully connected three layer net with 7 hidden neurons and 105 synaptic links was simulated for comparison. It converged to the same level of error in 490 iterations of the training set. The total computational work was slightly less for the growing net, as it had fewer connections, particularly in the early stages of the simulation.

5. CONCLUSIONS

Automatic evolution of neural nets based on synapse and neuron effectiveness and importance is practical. It allows easy generation of optimized multilayer back propagation feed-forward networks.

6. REFERENCES

1. L.P.J. Weelenturf, "An Automate-Theoretical Approach to Developing Learning Neural Networks," Cybernetics and Systems, Vol. 12, 1981, p. 179-202.

2. M.F. Tenorio and W.T. Lee, "Self Organizing Neural Networks for the Identification Problem," Neural Information Processing Systems, 1988, p. 57-64.

3. D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation," in Parallel Distributed Processing, Vol. I, p. 318-362, ed. by D.E. Rumelhart, J.L. McClelland, MIT Press, 1986.

4. R.L. Watrous, "Learning Algorithms for Connectionist Networks: Applied Gradient Methods for Nonlinear Optimization," IEEE First International Conference of Neural Networks, Vol. II, p. 619-627, San Diego, 21-24 June 1987, ed. by. M. Caudill, C. Butler.

Optimization Methods for Back-propagation: Automatic Parameter Tuning and Faster Convergence.

Roberto Battiti
Computation and Neural Systems Dept.,
158-79 California Institute of Technology
Pasadena CA 91125
Computer address: `roberto@hamlet.caltech.edu`

Abstract

Standard back-propagation learning (BP) is known to have slow convergence properties. Furthermore no general prescription is given for selecting the appropriate learning rate, so success is dependent on a trial and error process. In this work a well known optimization technique (conjugate gradient with inexact linear searches) is employed to speed up convergence and to select parameters. The strict locality requirement is relaxed but parallelism of computation is maintained, allowing efficient use of concurrent computation. While requiring only limited changes to BP, this method yields a speed-up of one or two orders of magnitude for medium-size networks.

Comparisons are done with BP using optimal parameters and with a version of BP employing learning rate adaptation. This last method is in itself interesting, since it converges in a number of iterations close to that of optimized BP, with no need for parameter optimization.

1 Introduction.

In back-propagation learning (see [6]) the *search direction* $\mathbf{d}_n$ is given by the negative gradient of the energy, while the step along this direction is proportional to $\mathbf{d}_n$ with a fixed constant ϵ chosen by the user (*learning rate*), as follows:

$$\mathbf{d}_n = -\nabla E(\mathbf{w}_n) \tag{1}$$

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \epsilon\ \mathbf{d}_n \tag{2}$$

Now, it is well known from the optimization literature that pure gradient descent methods tend to be very inefficient [3]. A case in which this happens is when "the search space contains long ravines that are characterized by sharp curvature across the ravine and a gently sloping floor" [6]. The situation can be ameliorated in part modifying the search direction with the introduction of a *momentum* term α, leading to the following rule:

$$\mathbf{d}_n = -\nabla E(\mathbf{w}_n) + \left(\frac{\alpha}{\epsilon}\right) \Delta\, \mathbf{w}_{n-1} \tag{3}$$

A recent overview of other heuristics employed to accelerate back-propagation has been presented in [5]. Unfortunately up to now there are ***no good general prescriptions*** for selecting the parameters defining the optimization strategy (like ϵ or α). It is usually left to the user to find a good or optimal combination of these parameters that leads to avoidance of local minima and fast convergence times. This process of ***meta-optimization*** (optimization of the optimization method) leads in general to a sizeable waste of computational resources.

The focus of this work has been on transferring some *meta-optimization* techniques usually left to the user to the learning algorithm itself. Since this involves measuring optimization performance and correcting some parameters while the optimization algorithm is running, some *global* information is required. *Parallelism* of computation is nonetheless maintained, resulting in efficiency close to 100% when the algorithm runs on a parallel computer.

In all cases the "standard" back-propagation algorithm is used to find the values of the energy and the negative gradient for a given configuration. The differences are in the definition of the *search direction* and/or in the selection of a *step size* along the selected direction.

In the first method proposed the search direction remains equal to the negative gradient but the step size is adapted during the computation. In the second one both the search direction and the step size are changed in a suboptimal but apparently very efficient way.

In both cases the network is updated only after the entire set of patterns to be learned has been presented to it.

2 The "bold driver" method (BD).

This strategy has been suggested independently in [9] and is here summarized for convenience before using it in the test problems. The proposed heuristic is to start with a given learning rate and to monitor the value of the energy function $E(\mathbf{w}_n)$ after each learning cycle. If E decreases, the learning rate is increased by a factor ρ. Vice versa if E increases, this is taken as an indication that the step made was too long, the learning rate is decreased by a factor σ, the last change is cancelled and a new trial is done. The process of reduction is repeated until a step that decreases the energy value is found.

Heuristically, ρ has to be close to unity (say $\rho \approx 1.1$) in order to avoid frequent "accidents", because the computation done in the last back-propagation step is wasted in these cases. Regarding the parameter σ a choice of $\sigma \approx 0.5$ can be justified with the reason that if the local "ravine" in the search space is symmetric on both sides this will bring the configuration of the weights close to the bottom of the valley.

The performance of this apparently "quick and dirty" method is close to and usually better than that obtainable by optimizing a learning rate that is to remain fixed during the procedure.

3 Conjugate gradient with inexact linear searches (CG).

Let's define the following vectors: $\mathbf{g}_n = \nabla E(\mathbf{w}_n)$, $\mathbf{p}_n = \mathbf{w}_n - \mathbf{w}_{n-1}$ and $\mathbf{y}_n = \mathbf{g}_n - \mathbf{g}_{n-1}$.

Shanno ([7]) reviews several conjugate gradient methods for function minimization and suggests one method that "substantially outperforms known conjugate gradient methods on a wide class of problems". In the suggested strategy the search direction for the n'th iteration is defined as[1]

$$\mathbf{d}_n = -\mathbf{g}_n + A_n \mathbf{p}_n + B_n \mathbf{y}_n \tag{4}$$

where the coefficients A_n and B_n are combinations of scalar products of the vectors defined at the beginning of this section, as follows:

$$A_n = -\left(1 + \frac{\mathbf{y}_n \cdot \mathbf{y}_n}{\mathbf{p}_n \cdot \mathbf{y}_n}\right) \frac{\mathbf{p}_n \cdot \mathbf{g}_n}{\mathbf{p}_n \cdot \mathbf{y}_n} + \frac{\mathbf{y}_n \cdot \mathbf{g}_n}{\mathbf{p}_n \cdot \mathbf{y}_n} \tag{5}$$

$$B_n = \frac{\mathbf{p}_n \cdot \mathbf{g}_n}{\mathbf{p}_n \cdot \mathbf{y}_n} \tag{6}$$

Every N steps (N being the number of weights in the network) the search is restarted in the direction of the negative gradient .

[1] Correction of the search direction based on previous steps is in part reminiscent of the use of a *momentum* term introduced in [6], with the added feature that a definite prescription is given for the choice of the various factors.

Successive approximations to the minimizer $\mathbf{w}^*$ of $E(\mathbf{w})$ are generated using one-dimensional minimization along the search direction:

$$\epsilon_n = \min_{\epsilon} E(\mathbf{w}_{n-1} + \epsilon\ \mathbf{d}_n) \tag{7}$$

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \epsilon_n\ \mathbf{d}_n \tag{8}$$

The one-dimensional minimization used in this work is based on quadratic interpolation and tuned to back-propagation where in a single step both the energy value and the negative gradient can be efficiently obtained. A small number of energy evaluations is sufficient. Details on this step are contained in [1].

Two example problems and the obtained results are described in the two following sections. A similar optimization approach, using Polak-Ribiere optimization, is presented in [4]. They also obtain a sizeable speed-up with respect to standard back-propagation, although they do not optimize its parameters for the comparison.

4 Test: the parity function.

Recently Tesauro and Janssens [8] measured optimal averaging training times and optimal parameters settings for standard back-propagation with momentum term. In order to benchmark the two proposed methods, the same network is used (n input units, $2n$ hidden units, one output).

The results of 100 simulations for each problem show first that back-propagation with adaptive learning rate produces results that are close to those obtained by optimizing parameters in back-propagation with fixed learning rate, second that the inexact conjugate gradient method brings a sizable speedup on both previous methods. Results are in table 1. Since the number of local minima is small in this case, only data regarding correct convergence are shown.

patterns	BP	BD	CG	speedup (BD/CG)
	av. cycles (st.dev.)	av. cycles (st.dev.)	av. cycles (st.dev.)	
2	24 (N/A)	46 (11)	16 (8)	2.8
3	33 (N/A)	57 (17)	22 (10)	2.6
4	75 (N/A)	137 (57)	68 (58)	2.0
5	130 (N/A)	213 (115)	93 (69)	2.3
6	310 (N/A)	616 (835)	199 (127)	3.0
7	800 (N/A)	875 (359)	371 (300)	2.3
8	2000 (N/A)	4310 (3088)	700 (368)	6.1

Table 1: Results for parity problem.

5 Test: the dichotomy problem.

This problem consists in classifying a set of randomly generated patterns in two classes. It has been demonstrated in [2] that an arbitrary dichotomy for any set of N points in general position in d dimensions *can* be implemented with a network with one hidden layer containing $\lceil N/d \rceil$ neurons. In this test the pattern coordinates are random values belonging to the [0-1] interval.

A dichotomy problem is defined by the number of patterns generated. The dimension of the space and the number of inputs is two, the number of middle-layer units is $\lceil N/2 \rceil$ by the above criterion and one output unit is responsible for the classification.

Simulation runs have been made starting from small random weights, with maximum size r equal to 0.1. Correct performance is defined as coming within a margin of 0.1 of the correct answer.

The capability of the network does not avoid the problem of local minima. In fact the results show that their number is increasing as a function of the dimension of the search space (i.e. the number of weights in the network). Average results for different test runs (the random number seed is changed) are given in table 2. Cases for correct solutions or local minima are shown separately.

patterns	BD	CG	speedup (BD/CG)
6	cases: av. cycles (st.dev.)	cases: av. cycles (st.dev.)	
correct	124: 1040 (1458)	115: 44 (56)	23.6
loc.min.	4: 9032 (10403)	13: 49 (74)	
10			
correct	104: 5044 (6870)	90: 204 (368)	24.7
loc.min.	24: 3923 (4914)	38: 404 (1005)	
16			
correct	106: 13245 (10572)	94: 295 (513)	44.8
loc.min.	22: 14116 (11960)	34: 755 (1605)	
20			
correct	111: 23293 (16792)	87: 380 (433)	61.3
loc.min.	17: 41000 (28583)	41: 1632 (3021)	
30			
correct	44: 46265 (20761)	36: 710 (418)	65.1
loc.min.	20: 59843 (16555)	28: 1800 (1300)	
50			
correct	4: 157296 (36837)	13: 1347 (600)	116.7
loc.min.	4: 211292 (59424)	51: 4307 (2159)	
100			
correct	0:	0:	N/A
loc.min.	8: 1435950 (560974)	64: 12645 (4161)	

Table 2: Results for dichotomy problem.

Acknowledgments.

Work supported in part by DOE grant DE-FG-03-85ER25009, the National Science Foundation with grant IST-8700064 and by IBM.

References

[1] R. Battiti, "Optimization Methods for Back-propagation: Automatic Parameter Tuning and Faster Convergence", *Concurrent Computation Tech. Rep.* **714-B** (Caltech,Pasadena,CA 91125,1989).

[2] E. B. Baum, "On the Capabilities of Multilayer Perceptrons", *Journal of Complexity* 4 (1988) 193–215.

[3] P. E. Gill, W. Murray and M. H. Wright, *Practical Optimization* (Academic Press,1981).

[4] A. H. Kramer, A. Sangiovanni-Vicentelli, "Efficient Parallel Learning Algorithms For Neural Networks", *Advances in Neural Information Processing Systems* Vol. **1** 75–89 (Morgan Kaufmann, CA, 1988).

[5] R. A. Jacobs, "Increased Rates of Convergence Through Learning Rate Adaptation", *Neural Networks* **1** (1988) 295–307.

[6] D. E. Rumelhart and J. L. McClelland (eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations,* (MIT Press,1986).

[7] D. F. Shanno, "Conjugate gradient methods with inexact searches", *Mathematics of Operations Research* **3 - 3** (1978) 244–256.

[8] G. Tesauro and B. Janssens, "Scaling relationship in back-propagation learning", *Complex Systems* **2** (1988) 39–44.

[9] T.P. Vogl, J.K. Mangis, A.K. Rigler, W. T. Zink and D. L. Alkon, "Accelerating the Convergence of the Back-Propagation Method", *Biological Cybernetics* **59** (1988) 257-263.

The Tempo-Algorithm: Learning in a Neural Network with Variable Time-Delays

Ulrich Bodenhausen, Department of Biophysics,
Philipps-University Marburg, Renthof 7, 3550 Marburg, FRG

Abstract

The artificial neural network proposed in this paper is a generalization of the Recirculation network proposed by Hinton and McClelland [6]. The network contains groups of non-linear units arranged in a closed loop. The connections between the units of the generalized network can be adjusted in weight and time-delay by two separate learning procedures. The purpose of this network is content-addressable storage of pattern sequences. During learning, the "hidden" units learn to detect specific phase differences between the activations of the "visible" units and vice versa. Simulations of an asymmetric network show that the hidden units can learn temporal relationships between pattern sequences.

1. Introduction

Connectionist networks have become an important approach to artificial pattern recognition and other applications [1] - [4]. Supervised gradient-descent learning procedures such as Back-Propagation have been shown to construct interesting internal representations in "hidden" units that are not part of the input or output of a connectionist network [1], [5]. A criticism of Back-Propagation is that it is neurally implausible because it requires the units to use different input-output functions for the forward and backward passes. The Recirculation learning procedure was designed to overcome this criticism [6]. This procedure adjusts the weights in a network module consisting of two layers of units arranged in a closed loop. The network can work as a content-addressable memory or an encoder network. The units of the first layer are called "visible" units. The input vector is the state of the visible units at $t = 0$. In the standard Recirculation network the connections have a fixed time-delay of one; thus, the signal reaches the hidden layer at $t = 1$ and is back to the visible layer at $t = 2$. The output of the content-addressable memory is the state of the visible units at this time. The basic idea of the generalization of the Recirculation learning procedure described in this paper is that the time-delays of the connections can be adjusted by a separate delay adjusting procedure.

Other time-delay neural networks (TDNN) have been proposed by Tank and Hopfield [7] and by Waibel et al. [8]. Waibel et al. use a generalization of the Back-Propagation network for phoneme recognition.

The basic advantage of TDNN is that they can learn the temporal relationships between patterns (for example: N patterns). Tank and Hopfield and Waibel et al. realize this ability by N connections with the time-delays 0, .. , N-1 between two units. The time-delays of these networks are fixed. The basic idea of the network proposed in this work is that a network with less than N connections between two units can learn the relationships between N patterns, if the time-delays are adjusted by a delay adjusting procedure.

Each hidden unit receives weighted information about the activities of the visible units at some specific times in the past.That means that the hidden units learn to represent features that are distributed in time *and* space.

2. The Tempo-Learning Procedure

The Tempo-learning procedure adjusts weights and time-delays in networks of the type illustrated by Fig 1. Each visible unit has one directed connection to every hidden unit and each hidden unit has one directed connection to every visible unit. With fixed time-delays of one and no adjustment of the time-delays, the network reduces to the standard Recirculation network.

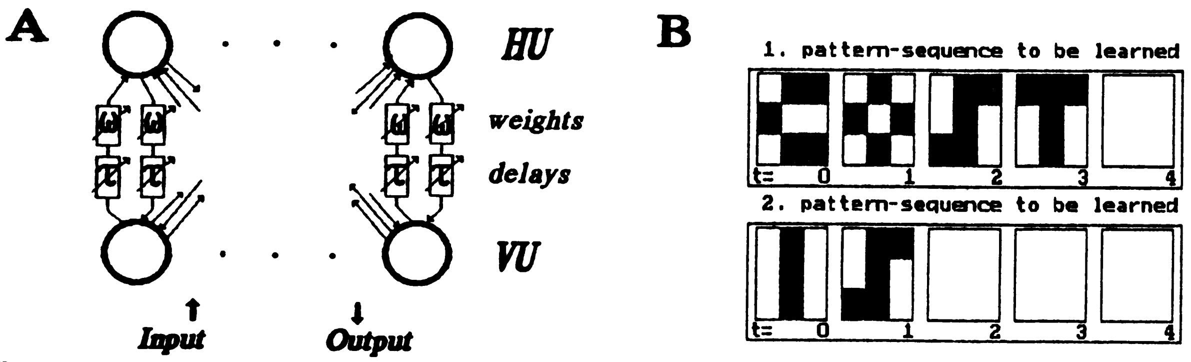

A
HU
weights
delays
VU
Input
Output
B
1. pattern-sequence to be learned
t= 0 1 2 3 4
2. pattern-sequence to be learned
t= 0 1 2 3 4

Fig. 1A: The Tempo-Network.

Fig. 1B: This figure illustrates the type of pattern-sequences the Tempo-network was tested with. The network (9 VU, 25 HU) had to learn the word "COST" (consisting of the patterns "C" at t = 0, "O" at t = 1, "S" at t = 2 and "T" at t = 3) and the word "IS".

The purpose of the generalized network is encoding and content-addressable storage of pattern-sequences. Each sequence consists of a definite number of frames. For example, the picture of a letter (see Fig. 1B) or the spectral coefficients of a time-window could be one frame. The number of units in the visible layer is equal to the number of pixels or the number of coefficients of one frame. The output of the network starts one timestep after the input of the last frame.

The activation functions of visible and hidden units are smooth monotonic functions with bounded deriatives. The logistic function is a very popular possibility.

The conditions under which a Recirculation network approximates gradient descent are [6]:

1. The visible units are linear.
2. The weights are symmetrical ($w_{ij} = w_{ji}$ for all i, j).
3. The visible units have high regression.

The Tempo-learning procedure requires the same conditions as the Recirculation learning procedure. "Regression" in a Tempo-Network means that

$$Y_i(t) = \lambda \cdot Y_i(t - n) + (1 - \lambda) \cdot Y_i(t)$$

with λ: regression, $0 \leq \lambda < 1$
$Y_i(t)$: state of the ith unit at time t
n: number of patterns of the pattern-sequence.

The learning rules for weights and time-delays are the derivations of an extended error-measure. The behaviour of the network can be described as follows: The hidden units learn to detect "correlated" activities of the visible units. "Correlated" does not mean that the hidden units detect phase coincidence. Rather, they detect specific phase differences between the activations of the visible units. On the backward pass, the visible units learn to detect specific phase differences between the activations of the hidden units. The network consists of two layers of "adaptive correlators" in the sense described above. Other models based on correlated activity are [3] and [9].

3. Simulations with the Tempo-Network

All results shown in this paper were obtained with an asymmetric network. It was chosen because a symmetric network is not biologically plausible. Although the visible units have to be linear to derive the learning rules, a network with uniform units was chosen. So the visible units use the same non-linearity as the hidden units. Although the network has to be time-continous for the derivation of the learning rules of the time-delays, a time-discrete network has to be simulated on a conventional computer.

Each weight- and time-delay update was performed after each training sequence. All weights in the network are started at small random values uniformly distributed in the range -0.5 to +0.5. All time-delays are initialized randomly between 0 and 5. During learning, the learning rates E1 (weights) and E2 (time-delays) are constant. After learning, both are set to zero.

I applied the Tempo-learning procedure to a network with 9 visible units and a variable number of hidden units. Two-dimensional patterns consisting of 9 pixels were used to show the performance of the network (Fig 1B). Each pixel is assigned to one visible unit. All pattern-sequences consist of 5 frames. In each case two pattern-sequences were learned. The examples (Fig. 3A) show that the Tempo-network is able to learn pattern-sequences and to substitute missing frames of incomplete test-sequences. Fig. 2 shows the distribution of the time-delays.

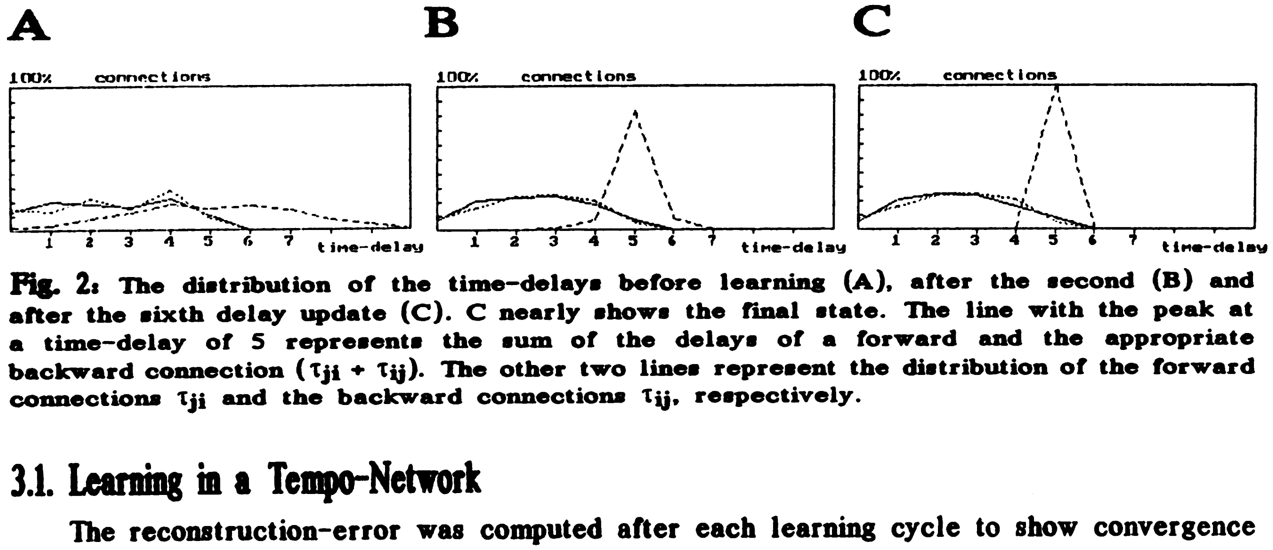
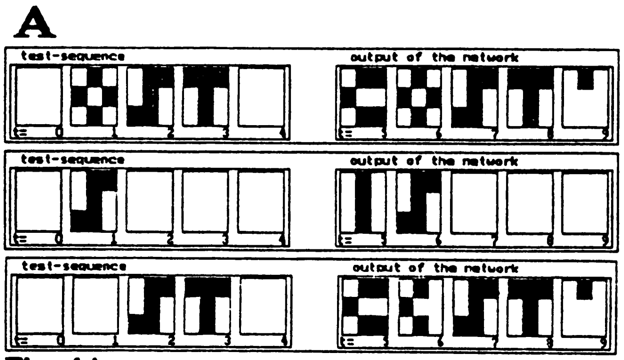


Fig. 2: The distribution of the time-delays before learning (A), after the second (B) and after the sixth delay update (C). C nearly shows the final state. The line with the peak at a time-delay of 5 represents the sum of the delays of a forward and the appropriate backward connection ($\tau_{ji} + \tau_{ij}$). The other two lines represent the distribution of the forward connections τ_{ji} and the backward connections τ_{ij}, respectively.

3.1. Learning in a Tempo-Network

The reconstruction-error was computed after each learning cycle to show convergence of the Tempo-network. The weight and time-delay updates were computed after each training-case. My investigations concentrated on varying the number of hidden units, the learning rate of the weights (E1) and the learning rate of the time-delays (E2). In each run the pattern-sequences "COST" and "IS" had to be learned. The main results are:

1. More than 11 hidden units are needed for convergence.
2. The learning rate of the weights can't be chosen independently from the learning rate of the delays and vice versa. This behaviour can be explained by the interaction of the weight and the delay adjusting procedures. If the weights are changed "slowly", the delays have to be changed "slowly", too. Otherwise weights or delays are adjusted "egoistically" and the other variable can't be adjusted.
3. With the right choice of the parameters E1, E2 and λ, convergence can be achieved after less than 50 learning cycles (Fig. 3B).



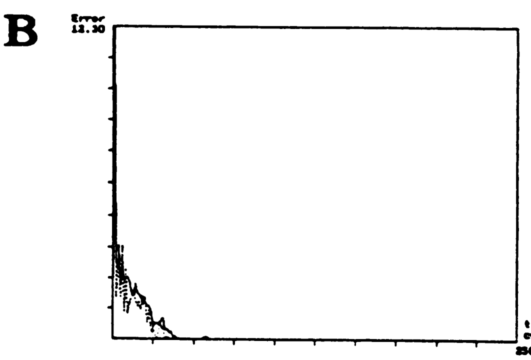


Fig. 4A: The performance of the network. The previously learned sequences are illustrated by Fig. 1B.

Fig. 4B: The reconstruction-error as a function of the number of training cycles (E1 = 1.5, E2 = 0.7, λ = 0.4, 25 hidden units).

4. Conclusion

The network proposed in this work is a new approach to processing of pattern-sequences. Networks with time-delays have been shown to learn temporal relationships between patterns [7], [8]. The basic idea of this paper is that the number of connections in time-delay neural networks can be reduced by the implementation of learning rules for time-delays. The present realization of this idea (the Tempo-network) is restricted to one directed connection between each visible unit and each hidden unit and vice versa. With a sufficient number of hidden units it can learn the temporal relationships between patterns and work as a content-addressable memory of pattern-sequences.

Although the Tempo-network can work as a content-addressable memory with an interesting internal representation of the stored information, it can't work as an encoder network because the number of hidden units has to exceed the number of visible units. The number of hidden units needed for convergence can probably be reduced by using another type of unit (for example non-linear leaky integrators) and/or increasing the number of connections between the units (two or three instead of one). The type of delay-adjusting procedure may also be a useful generalization for other networks.

References:

[1] Rumelhart, D. E. , McClelland, J. L. 1986, "Parallel distributed processing", Vol. 1 & 2, MIT-Press (Cambridge 1986)

[2] Hinton, G. E. 1987, " Connectionist learning procedures " , Artificial Intelligence

[3] Eckhorn, R., Reitboeck, H.J., Arndt, M., Dicke, P. 1989, "Feature Linking via Stimulus-Evoked Oscillations", Proceedings of the IJCNN Washington D.C., 1989, Vol. 1, pp. 723 - 730

[4] Sejnowski, T. J. , Rosenberg, C. R. 1987, "Parallel networks that learn to pronounce english text ", Complex Systems, Vol. 1, pp. 145 - 168

[5] Rumelhart, D. E. , Hinton, G. E. , Williams, R. J. 1986, "Learning representations by back-propagation", Nature, Vol. 323 (9), pp. 533-536

[6] Hinton, GE., McClelland, J.L. 1987, " Learning Representations By Recirculation ", Manuscript in preparation, Carnegie Mellon University, Pittsburgh, PA

[7] Tank, D.W., Hopfield, J.J. 1987, " Neural computation by concentrating information in time ", Proc. Academy Sci, Apr. 1987, pp. 1896 - 1900

[8] Waibel, A., Hanazawa, T., Hinton, G. E., Shikano, K., Lang, K.J. 1989 , " Phoneme Recognition Using Time-Delay Neural Networks ", IEEE Transactions On Acoustics, Speech, And Signal Processing, Vol. 37, No. 3

[9] Malsburg, C.v.d., Bienenstock, E. 1987, " A Neural Network for the Retrieval of Superimposed Connection Patterns ", Europhysics Letters, 3 (11), pp. 1243 - 1249

Stepsize Variation Methods for Accelerating the Back-Propagation Algorithm

J.R.Chen and P.Mars
School of Engineering and Applied Science
University of Durham, South Road, Durham, DH1 3LE U.K

Abstract:

In this paper we discuss results on improving the convergence speed of the back-propagation algorithm, and introduce an adaptive stepsize technique and a differential stepsize method to accelerate the convergence speed of the back-propagation algorithm. Simulation results are presented which illustrate the improved convergence.

1.Introduction

The recent revival of research activities in neural networks was significantly influenced by the publication of [1]. With the learning algorithm called the back error-propagation, it was shown that the Multi Layer Perceptron (MLP) can perform interesting computations[1]. Unlike the perceptron analysed by Minsky[2] which can only solve linear separable problems, the MLP , theoretically , can divide the input space into arbitary shape, provided that there are enough hidden units. Thus MLP methods have been applied to several complex pattern classification-like problems, such as that reported in [3]. However the main drawback in applying MLP networks to many real problems is the slow convergence speed of the back-propagation algorithm.

While the back-propagation algorithm is a kind of gradient descent algorithm, error surfaces for learning problems frequently possess some geometric properties that makes the algorithm slow to converge. The stepsize of the algorithm is sensitive to the local shape and curvature of the error surfaces. For example, a small stepsize will make the algorithm take a very long time to cross a long flat slope. On the other hand, a large stepsize will cause the iteration process to bounce between the two opposite sides of a valley rather than following the contour of its bottom. Even if a satisfactory stepsize for one stage of the learning process is found, this does not ensure it will be appropriate for any other stage of the same learning process. On the other hand, the premature saturation of the network units also causes problems for the convergence of the algorithm. Thus in the following we introduce an adaptive stepsize back-propagation algorithm and a simple method for circumventing the premature saturation.

2.Previous Research

There has been some research on improving the convergence speed of the back-propagation algorithm, such as that mentioned in [4][5][6]. In [4] the authors suggested Conjugate gradients, Quasi-Newton algorithm and other more sophisticated algorithms. They are also called second order methods. According to our knowledge, the convergence speed reported in [1] on the XOR problem is the fastest among the existing algorithms. However all these algorithms are much more computationally expensive, especially when the scale of the problem is large, so that in many cases it is impractical to use them. In order to reduce the computation cost of the second order method, a kind of approximation technique has been introduced into the Newton's algorithm[5]. The authors used a diagonal matrix to approximate the Hessian matrix. This makes it possible to derive a back propagation algorithm for the second order derivatives as that for the first order derivatives. But the applicability of this new algorithm depends on how well the diagonal Hessian approximation models the true Hessian[5]. Only when the effects of weights on the output are uncoupled or nearly uncoupled, can the diagonal Hessian represent a good approximation. We have implemented this Newton-like method in our back-propagation simulation program. At this stage we have not found it to exhibit any advantage over the ordinary back-propagation algorithm. This may be due to the use of sub-optimal learning parameters. Just as was mentioned in [5], we found the learning parameters are more critical in obtaining reasonable behaviour with this Newton-like algorithm than with the back-propagation algorithm. Another attempt to use a second order method to improve the convergence property of the back-propagation algorithm was introduced in [6], which is called Quickprop. It uses the difference between two successive $\frac{\partial E}{\partial w}$ as a measure of the change of curvature and uses this information to change the stepsize of the algorithm. E is the output error function, and w represent weights. Using this method a significant improvement on convergence speed has been reported in [6].

In [7] another kind of adaptive stepsize algorithm was introduced. According to this algorithm, if an update of weights results in reduced total error, the stepsize is increased by a factor $\phi > 1$ for the next iteration. If a step produces a network with a total error more than a few percent above the previous value, all changes to the weights are rejected, the stepsize is reduced by a factor $\beta < 1$, the momentum term is set to zero, and the step is repeated. When a successful step is then taken, the momentum term is reset.

As is well known in adaptive signal processing theory, the direction of the negative gradient vector may not point directly towards the minimum of the error surface. In adaptive filter theory, this kind of bias can be measured by the ratio of the maximum eigenvalue and the minimum eigenvalue of the auto-correlation matrix[8]. Recently an adaptive stepsize algorithm which gives every weight a stepsize which can adapt separately has been proposed[9]. This is only a rough approximation, as it will be noted that these stepsizes adapt on the direction of each weight rather than on the eigenvector direction as required[8][9].

In the back-propagation algorithm, the update of weights can take place after presenting all the training samples to the network or after every presentation of a training sample, they are called batch mode back-propagation and online back-propagation respectively. Generally speaking, online back-propagation algorithms converge faster than the batch mode back-propagation[5][6], and batch mode back-propagation is more likely to fail to converge on a large training sample set[10]. The algorithms described above are all batch mode back-propagation, because for the second order method it can only use batch mode. In the following we introduce an adaptive stepsize online back-propagation algorithm. It is considered to represent an advance on existing algorithms.

3.Adaptive Stepsize Back-Propagation

In designing an appropriate algorithm the following factors should be considered. First the momentum term cannot be set to zero, as the update occurs for every presentation of a new training sample. If the momentum term is set to zero, there exists a risk of losing past experience. Generally speaking. a large training sample set requires a large η value (η is the stepsize for the momentum). This fact has been confirmed by computer simulation[11]. Thus the adaption is restricted to the gradient term. We used the following form of adaptive stepsize algorithm:

$$\alpha(t) = \alpha(t-1)(1 - f(t)\sqrt{E(t)}\ ) \tag{1.a}$$

$$f(t) = u_1 f(t-1) + u_2 \Delta E(t) \tag{1.b}$$

$$\Delta E(t) = E(t) - E(t-1) \tag{1.c}$$

$\alpha(t)$ is the stepsize for the gradient term in the update formula in the back-propagation algorithm. It is the stepsize at moment t. E(t) is the summation of squared discrepencies between the desired output and the actual output at time t. It can be calculated as following:

$$E = \frac{1}{2}\sum_{k=1}^{N}\sum_{i=1}^{p}(d_i^k - o_i^k)^2 \tag{2}$$

$\Delta E(t)$ is the decrement of the E(t). f(t) is a filtered version of $\Delta E(t)$. Actually (1.b) is a first order low-pass recursive filter, which can smooth the significant changes in $\Delta E(t)$, making the algorithm more stable. u_1 and u_2 are the parameters used to control the adaptation. For small u_1 and big u_2, the adaptation is fast, but it is also more likely to be trapped in oscillation. For big u_1 and small u_2, the adaptation is slow, but it is more stable. Thus the parameter selection involves a trade off. In our simulation, we used $u_1 = 0.9$ and $u_2 = 0.3$. The term $(1 - f(t)\sqrt{E(t)}\ )$ also controls the adaptation of the stepsize. If f(t) is positive, that means the tendency of E(t) in the near past is to increase, so $1 - f(t)\sqrt{E(t)} < 1$, the stepsize will be decreased. A similar analysis shows that if the tendency of E(t) is to decrease, the stepsize will be increased. When the E(t) is very small, that is the network has almost learned, the adaption will be very weak, which stablizes the algorithm. The square root is used as compensation, it can amplify the small E(t) to avoid the premature termination of adaptation.

We now present some simulation results to show the advantage of the adaptive step size algorithm. In the diagrams shown, the E defined in (2) are plotted as a function of iteration times for different learning problems. They are called learning curves, and can be used to evaluate the convergence property of the learning algorithm. Their maximum are normalized to 1. In Fig-1 we show comparative simulation results of the non-adaptive back-propagation algorithm and the adaptive algorithm for the 4-4-1 parity problem. It is clear the adaptive stepsize has improved the convergence speed, just as we expected. In our simulation we find that the improvement on the complex problem are more impressive than that on simple problem. The reason may be that since adaptation is a dynamic process, it needs a finite time to be effective. For simple problems, the learning process is very short, the adaptation process has not sufficient time to play its role. Thus there is only a small effect of adaption on simple learning problems.

4.Differential Stepsize Back-Propagation

Although the adaptive stepsize back-propagation algorithm has improved the learning speed to some degree, it cannot cope with the premature saturation of the network units. We have noted in our simulations that MLP neural nets are often trapped in a very flat valley or so called local minima, in which area the convergence speed is very slow which corresponds to the flat line intervals on the learning curves of Fig-1. This cannot be solved by an adaptive stepsize technique, because the reason for this phenomenon is that the absolute value of weights are growing so fast as to make the units, especially hidden units, prematurely saturated. There is a term like s(1-s) in the update formula for the back-propagation algorithm, in which s is the output state of the unit. It is quite clear that if s is close to 1 or 0, whichever output is desirable, almost no update will be passed backward through that unit. This kind of phenomenon is also known as the flat spot[6]. In [6] the author suggested to change the sigmoid-prime function s(1-s) to s(1-s)+0.1, so it can avoid the flat spot. But according to our simulations, this change often causes the weights to grow so fast as to lead to floating point overflow on the digital computer. Although some weight-decay term may be used to counteract this[6], it makes the algorithm more complex. We have used a very simple method to cope with the flat spot.

A straight forward idea to circumvent the flat spot is to remove the term s(1-s) from the update formula for the output layer, and set the stepsize for the update of weights between the hidden layer and the input layer smaller than that for the weights between the upper layers. We denote the stepsize for the update of weights between the output layer and the hidden layer as α_2, and the stepsize for the update of weights between the hidden layer and the input layer as α_1, then $\alpha_2 > \alpha_1$. We call this the differential stepsize back-propagation algorithm(DSBP). In our simulation, we used $\alpha_1 = 0.1\alpha_2$. The simulation results are shown in Fig-2, and it is very clear the convergence speed is improved considerably.

In [6] the Quickprop algorithm was claimed to be the fastest learning algorithm among the existing algorithms. In order to compare our DSBP with the Quickprop, we have run 30 simulation trials on the 10-5-10 encoder problem. The termination condition for the simulation is that the discrepancy between the desired output and the actual output for every output unit and every training sample is less than 0.1. The average training time for this problem by DSBP is 23.5, with a standard derivation of 3.27. This is only marginally slower than the Quickprop algorithm, for which the average training time is 22.1. However although the Quickprop plus a hyperbolic arctan error function algorithm can reach the same solution with an average training time of 14.01, it is much more complex than DSBP, and a weight-decay term is needed. The results for the simple DSBP algorithm represent a considerable improvement on the standard back-propagation algorithm, which gave an average training time of 129 iterations.

5.Conclusion

From the above discussion, it is clear that the adaptive stepsize technique can improve the convergence speed of the back-propagation algorithm. It is obvious that the degree of improvement for a complex learning problem is greater than that for simple problems. We consider that the potential of the adaptive stepsize technique lies in the area of real large scale application problems, such as Net-Talk[3], in which the training sample set is very big, and the training process may last for a few days. From the simulation results shown above, we can also conclude that the DSBP method we used to circumvent the premature saturation or flat spot is effective. It is also surprising that such a small change to the algorithm can produce such a significant improvement, and confirms the importance of concentrating on a theoretical understanding of the dynamics of the back-propagation algorithm.

References

[1] David E.Rumelhart, James L.McClelland, *Parallel Distributed Processing* Vol-1 The MIT Press 1987

[2] Marvin L.Minsky, Seymour A.Papert, *Perceptrons* Expanded Edition The MIT Press 1988

[3] T.J.Sejnowski, C.R.Rosenberg, *Parallel Networks that Learn to Pronounce English Text* Complex System 1 pp145–168 1987

[4] A.R.Webb, David Lowe, M.D.Bedworth, *A Comparison of Nonlinear Optimisation Strategies for Feed-Forward Adaptive Layered Network* Royal Signals and Radar Establishment Memorandum 4157 1988

[5] Sue Becker, Yann le Cun, *Improving the Convergence of Back-Propagation Learning with Second Order Methods* Proc 1988 Summer School on Connectionist Model, Carnege-mellon Univ, pp29–37

[6] Scott E.Fahlman, *Fast-Learning Variations on Back-Propagation : An Empirical Study* Proc 1988 Summer School on Connectionist Model, Carnege-Mellon Univ, pp38–51

[7] T.P.Vogl, J.K.Mangis, A.K.Rigler, W.T.Zink and D.L.Alkon, *Accelerating the Convergence of the Back-Propagation Method* Biological Cybernetics vol-59 pp257–263 1988

[8] S.Haykin, *Adaptive Filter Theory* Englewood Cliffs, NJ, Prentice-Hall 1986

[9] Robert A.Jacobs, *Increased Rates of Convergence Through Learning Rate Adaptation* Neural Network vol-1 pp195–307 1988

[10] P.J.Lloyd, *Guidlines for Implementing Problems on a Multi-layer Perceptron Network* RIPRREP/1000/27/88

[11] G.Tesauro and B.Janssens, *Scaling Relationships in Back Propagation Learning* Complex System vol-2 No.1 pp38–44 1988

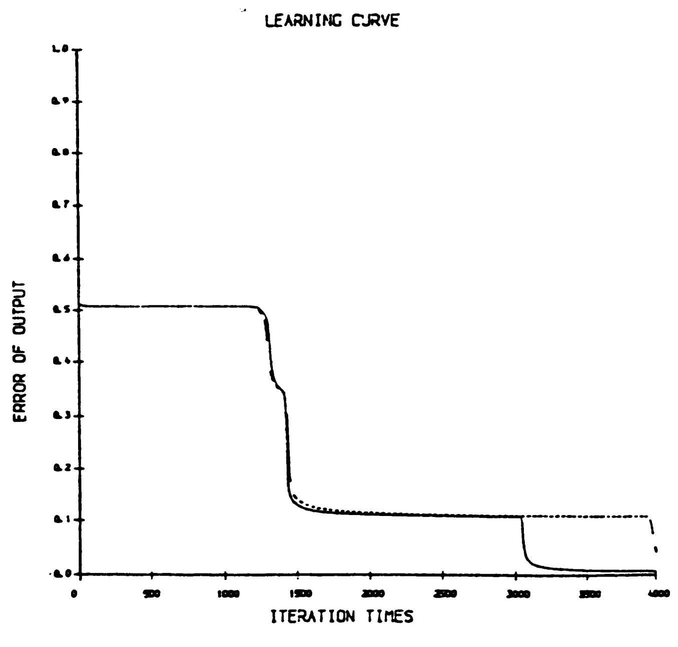


Fig-1

Learning curves for the 4-4-1 parity problem. Broken line stands for the learning curve of non-adaptive algorithm. The initial random value of weights are within the range(-0.5, 0.5), w-seed=5697, th-seed=8461, $\alpha = 0.4$, $\eta = 0.9$.

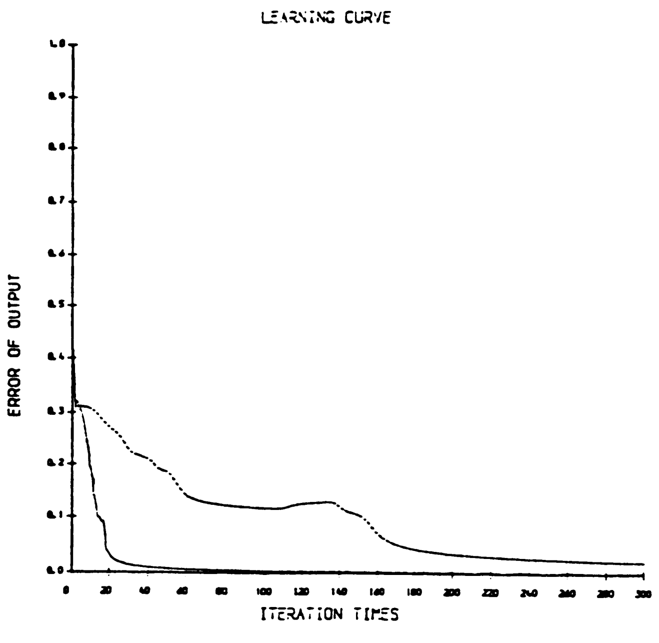


Fig-2

Learning curves of the 10-5-10 encoder problem. Solid line stands for the learning curve of the differential step-size back propagation algorithm(DSBP). The initial random value of weights are within the range(-1, 1), w-seed=4581, th-swwd=818953, $\alpha = 0.6$, $\eta = 0.9$. But for the DSBP $\alpha_1 = 0.06$, $\alpha_2 = 0.6$.

An Accelerated Learning Method with Backpropagation

Sung-Bae Cho and Jin H. Kim
Computer Science Department
Korea Advanced Institute of Science and Technology
P.O.Box 150, Cheongryang, Seoul 130-650, Korea
sbcho@csd.kaist.ac.kr (Internet)

ABSTRACT : *Backpropagation is the most widely used learning technique of neural network because of its simplicity and robustness. The slowness of its learning, however, is the major obstacle for its application to real-world problems.*

In this paper, an accelerated learning method which is based on the iteration for solving nonlinear optimization problems is proposed. This technique not only accelerates the rate of convergence, but also induces convergence in some cases if the iteration diverges. Experimental results are evaluated and its superiority relative to other methods is discussed.

I. Introduction

The learning of neural networks is the systematic adjustment of connection weights so that the output of the network approximates the desired output. This process can be formulated as a nonlinear optimization since the response computed by neural network is typically a nonlinear function of the connection weights [7,8]. In recent years, backpropagation algorithm has appeared as one of the most efficient learning procedures for multi-layer neural networks. The main reason of the success is on its simplicity. Backpropagation learning algorithm which is basically a gradient descent search method, however, is too slow to apply for the real-world problems.

Researchers have tried several different approaches to speed up the convergence of backpropagation learning. One approach is to use more elaborate search methods. Most of these are variations of Newton's method, and require the computation, or approximation of second partial derivatives [4,5,6]. Others have tried a systematic, empirical study of learning speed in the backpropagation algorithm from finding the heuristics for achieving faster rate of convergence [2,3]. Thus, many algorithms that has been proposed for rapid learning use the approximated high-order derivative of the error function, which provides the information about the shape of the weight space, and the rate of convergence is dramatically increased. These methods, however, require much computation, and are inclined not to scale up very well as the problem size increases. Therefore, it is necessary to develop faster learning methods, which require a moderate amount of computation and have a good scale-up property.

In this paper, the backpropagation method is reformulated in the context of nonlinear optimization, and a new accelerated learning technique is proposed. In addition to speed up, it is capable of producing solutions even in those cases where the iteration of backpropagation may be divergent. The learning process of this method consists of two stages; acceleration stage and attention stage. In acceleration stage, early steps of the learning, search process moves the network quickly across the solution space, and in late attention stage, focuses the search direction slowly and accurately toward a minimum.

II. Backpropagation Revisited

Neural network can be considered as a mapping between a set of input and a set of output. Mathematically speaking, a neural network represents a function F that maps I into O; F:I -> O, or Y = F(X) where Y in O and X in I. In the backpropagation learning, the mapping relationship is described by $Y = F(X; W^{MI}, W^{OM})$ to indicate that F depends on the values of the weights W^{MI} for connections between input and hidden layers, and W^{OM} between hidden and output layers [7].

The function to be solved by backpropagation is similar to that by nonlinear optimization, and backpropagation, in this sense, is simply considered as an iterative scheme that are performed by the network itself in order to solve the nonlinear optimization. In general, the iterative methods of optimization find a desirable direction of search, and then the objective function is minimized along the direction of search [9]. The goal of it is to find the connection weights (analogous to the unknown parameters) to minimize the total error function which can be made explicit in terms of the difference between the actual and desired outputs of the network to the set of input terms. The following table compares the backpropagation approach with the general nonlinear optimization.

	Nonlinear optimization	Backpropagation approach
Objective function		$E(W) = \frac{1}{n}\sum_{j=1}^{n}\sum_{t=1}^{o}\left[f\left\{\sum_{k=1}^{m} W_{tk}^{OM} f\left(\sum_{i=1}^{I} W_{ki}^{M} X_{ji}\right)\right\} - Y_{jt}\right]^2$
Function to be solved	$f(x) = 0$	$\frac{\partial}{\partial W^{MI}}E(W) = \frac{\partial}{\partial W^{OM}}E(W) = 0$
Iterative formula	$x_{n+1} = x_n + \Delta x_n$	$W^{MI}(n+1) = W^{M}(n) + \eta \frac{\partial}{\partial W^{MI}(n)}E(w)$ $W^{OM}(n+1) = W^{OM}(n) + \eta \frac{\partial}{\partial W^{OM}(n)}E(W)$

III. Accelerated Learning with Aitken's Δ^2 Process

Backpropagation is thought of as an iterative gradient method for solving nonlinear optimization problems. In optimization problems, the more information about the objective function yields rapid convergence. The gradient methods, we claim, utilize not sufficient information in approximating its minimum. The trend of successive gradient vectors may provide information about the error surface, which is useful for speed up convergence of the searching process. In this context, we propose an accelerated learning method which is based on a second order gradient method known as Aitken's Δ^2 process [10].

Let $\{w_n\}^\infty$ be a linearly convergent sequence of values converging to some point p; that is, for $e_n = w_n - p$,

$$\lim_{n\to\infty}\frac{|e_{n+1}|}{|e_n|} = c < 1. \tag{1}$$

To investigate the construction of a sequence $\{\underline{w}_n\}^\infty$, which converges more rapidly to p, suppose that the iteration $w_n = g(w_{n-1})$, n=1,2,3,...., satisfies

$$w_{n+1} - p = c(w_n - p), \tag{2}$$

$$w_{n+2} - p = c(w_{n+1} - p); \tag{3}$$

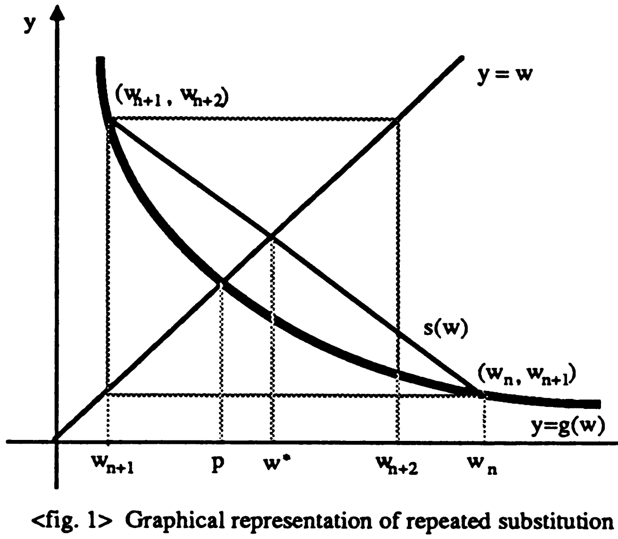
and solving equations (2) and (3) for p while eliminating c leads to

$$p = \frac{w_n w_{n+2} - w_{n+1}^2}{w_{n+2} - 2w_{n+1} + w_n}$$

$$= w_n - \frac{(w_{n+1} - w_n)^2}{w_{n+2} - 2w_{n+1} + w_n}$$

$$= w^*$$

$$= \Delta^2(w_n)$$



<fig. 1> Graphical representation of repeated substitution

In general, the original assumption (2) will not be true; nevertheless, it is expected that the sequence $\{\underline{w}_n\}^\infty$, defined by

$$\underline{w_n} = w_n - \frac{(w_{n+1} - w_n)^2}{w_{n+2} - 2w_{n+1} + w_n} \tag{4}$$

converges more rapidly to p than the original sequence $\{w_n\}^\infty$. The point w^* is a better approximation of p than w_n or w_{n+1} is. Graphically, the solution of $w = g(w)$ amounts to the problem of finding the point of intersection of the curve $y = w$ and $y = g(w)$, and w^* is the solution of the linear interpolant to $g(w)$ at w_n, w_{n+1}. If $g(w)$ is approximately a straight line between w_n and p, then the secant $s(w)$ is a very good approximation of $g(w)$ in that interval; hence the fixed point $w*$ of the secant is a very good approximation of the solution p. In this way, the better approximation is found in each iteration.

This process, which is called the Aitken's Δ^2 process, accelerates the convergence of any sequence that is linearly convergent, and gives quadratic convergence without evaluating a derivative. Moreover, Aitken's Δ^2 process not only accelerates convergence but also converts divergence into convergence in some cases. It is easy to verify that if

$$w_n = 1 + p + ... + p^n,$$

then

$$\Delta^2(w_n) = (1 - p)^{-1}.$$

Therefore, the Aitken's Δ^2 process gives immediately the limit of this particular sequence when it converges and assigns a meaningful value even when it does not.

Since the weight updating formula of the backpropagation algorithm is considered as $w_n = g(w_{n-1})$ for n=1,2,3,..., the backpropagation using Aitken's Δ^2 process achieves more rapid rate of convergence in theory. Experiments, however, show the infinite iterations in this method. That is, after some fast convergence, this method doesn't converge any more. The reason of oscillation is the overshooting of the minimum point, which is due to the nonlinear convergences of backpropagation algorithm in some interval.

Gradient descent used for backpropagation algorithm is known that its convergence is linear [9]. In practice, however, the latter half of the convergence process may be nonlinear according to the problems, and the necessary condition for Aitken's method may be not satisfied. So does not operate the straightforward application of the Aitken's method to the backpropagation. In most problems, however, the early stages of the learning satisfy the condition (1), though their convergence is even nonlinear, because the backpropagation method initializes its weights very small.

Therefore, the learning is composed of two parts; acceleration part and attention part. In the beginning of the learning process which satisfies (1), the Aitken's Δ^2 process with backpropagation accelerates the rate of convergence, and at the moment of oscillation due to its violation of (1), the standard backpropagation focuses the search direction slowly and accurately toward a minimum.

The last point to be considered is how to find the transition point between acceleration process and attention process. If it requires too much effort, the advantage of acceleration method is overrided. Fortunately, the oscillation point may be easily found by local computation, which uses only the ratio of previous and current total-error.

IV. Experiments and Results

The XOR problem [1] which is the most popular benchmark is used for our experiments. With the 25 trials randomly initialized and terminated when the total error is less than 0.04, the average trial over all the runs was 334 epochs with standard deviation 148 when the backpropagation with momentum is used. The proposed method, however, requires 159 average epochs with standard deviation 48.

Therefore, the new method not only requires less number of iterations than others, but also is more stable in the sense that the difference between the worst and the best case is small; besides, three non-convergent cases out of four in the standard backpropagation converge to the solution in the proposed method. We also compared the proposed method with the backpropagation with 0.1 added sigmoid prime [3]. It turns out that this method is superior to standard backpropagation in many cases, but somewhat unstable because it has a large difference between the worst and the best case. <fig. 2> through <fig. 4> shows the shape of convergence in the case of the usual, the best and the worst, respectively.

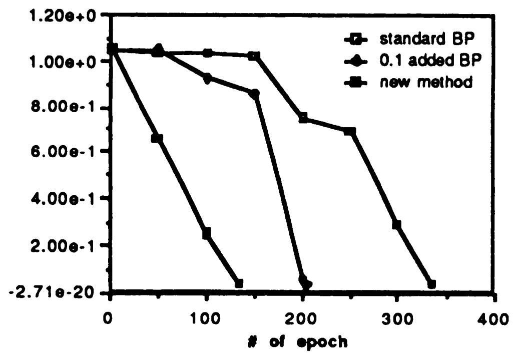
V. Concluding Remarks

In this paper, we propose the new speedup method for the learning, which is based on Aitken's Δ^2 process. Experiment with the XOR problem confirms the improvement of the proposed method.

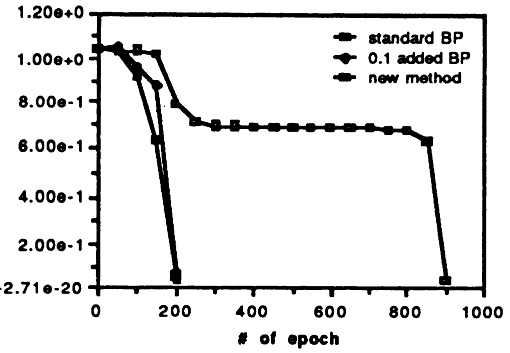
The simplicity of form and quadratic convergence of the Newton-Raphson method have made it very popular in the speedup technique of learning, but the need to evaluate a derivative is sometimes a serious drawback. The proposed method, on the other hand, which avoids the computation of a second derivative but retains quadratic convergence is therefore very useful in many cases. In addition, even if the backpropagation does not converge, it converges in some cases, and generates a search direction more consistently in the direction of the solution.

	Max	Min	Average	S.D.	Cases of oscillation
BP	899	231	333.57	148.45	4
BP (Sigprime+0.1)	545	191	271.88	109.34	0
New method	257	61	159.29	48.49	1

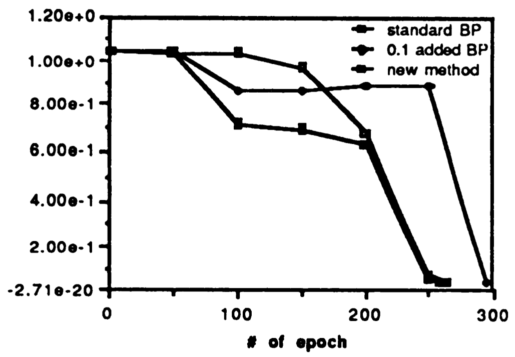
<table 1> the number of epochs with 25 trials



<fig. 2> # of epoch vs. total error (usual case)



<fig. 3> # of epoch vs. total error (best case)



<fig. 4> # of epoch vs. total error (worst case)

References

[1] Rumelhart, D. E. and McClelland, J. L., *Parallel Distributed Processing : Explorations in the Microstructure of Cognition*, MIT Press, 1986.

[2] Jacobs, R. A., "Increased Rates of Convergence Through Learning Rate Adaptaion," *Neural Networks*, Vol. 1, pp. 295 - 307, 1988.

[3] Fahlman, S. E., "An Empirical Study of Learning Speed in Back-Propagation Networks," *Technical Report CMU-CS-88-162*, June 1988.

[4] Parker, D. B., "Optimal Algorithms for Adaptive Networks : Second Order Backpropagation, Second Order Direct Propagation, and Second Order Hebbian Learning," *Proceedings of the IEEE International Conference on Neural Networks*, Vol. II, pp. 593 - 600, 1987.

[5] Parker, D. B., "A comparison of algorithms for neuron-like cell," *Neural Networks for Computing*, pp. 327 - 332, 1986.

[6] Becker, S. and le Cun, Y., "Improving the Convergence of Back-Propagation Learning with Second Order Methods," *Proceedings of the 1988 Connectionist Models Summer School*, pp. 29 - 37, 1989.

[7] Angus, J. E., "On the connection between neural network learning and multivariate nonlinear least squares estimation," *Neural Networks*, Vol. 1, No. 1, pp. 42 - 47, January 1989.

[8] Watrous, R. L., "Learning Algorithms for Connectionist Networks : Applied Gradient Methods of Nonlinear Optimization," *Proceedings of the IEEE International Conference on Neural Networks*, Vol. II, pp. 619 - 627, 1987.

[9] Luenberger, D. G., *Linear and Nonlinear Programming*, Addison Wesley, 1984.

[10] Conte, S. D. and de Boor, C., *Elementary Numerical Analysis*, 3rd Ed., McGraw-Hill, 1980.

ALGEBRAIC ANALYSIS OF NEURAL NETWORKS APPLICATIONS INDEPENDENT OF GLOBAL NETWORK ARCHITECTURE

William H. Clingman
W.H. Clingman & Co.
700 N. Pearl St., Suite 300
Dallas, Texas 75201

Donald K. Friesen
Department of Computer Science
Texas A & M University
College Station, Texas 77843

ABSTRACT

This paper is an extension of earlier work of the authors on the algebraic analysis of learning paradigms in neural networks. The technique is to map the stepwise convergence of the neural network toward a goal into a learning automaton with certain convergence characteristics. Such automata have been studied by the authors and their algebraic structure analyzed. From this structure, a lower bound can be assigned to the number of steps in the goal—seeking process. The authors previously reported applications to a three layer neural network with a back—propagation learning paradigm. In this paper, these results have been extended to any general purpose network whose training paradigm has certain local symmetry properties.

INTRODUCTION

The authors have been developing a theory of learning automata that is applicable to studying goal—seeking systems of many types. The stepwise convergence of such systems to a goal state can be mapped into training sequences of an appropriate learning automaton. The state transformations of the automaton that enable the learning process form a semigroup that is characteristic of the automaton. The authors have derived elsewhere (1,2) relations between the characteristic semigroup structure and convergent properties of the automaton training sequences. For certain special cases enough information can be derived about the semigroup to establish lower bounds on learning times. If a real—world system, such as an artificial neural network, can be mapped into one of these learning automata, then similar lower bounds would apply to the number of steps required for the system to reach its goal.

Previously the authors applied this theory to a three layer, feed—forward, neural network that is trained by back—propagation (2). In the present paper, the applications are extended to general purpose networks, where the global architecture and training paradigm are not specifically defined. The theory is used to relate the local structure of the network to global, worst case learning times.

CONCEPTS AND DEFINITIONS

A <u>learning automaton</u> is a 5—tuple D=(I,S,h,F,g) where I is a finite set of inputs, S is a set of states, h:IxS —-> S is the state transition function, F is a family of subsets of S, and g: S —--> F associates with each state, x, a subset of acceptable states which must include x. Two other functions are associated with the automaton. These are f: LxS —-> 0 and d_z (t)= {f(t,x): x ∈ g(z)} where L is a set of tests and 0 is a set of test results. L is a subset of I. The function, f, gives the result of a test and d_z gives the subset of acceptable test results if z is the goal state.

A <u>training sequence</u>, T(x,y), for D is a sequence of states $(S_1,S_2, ''' S_n)$ where $S_1 = x$, S_n is in g(y), and $S_{j+1} = h(i_j, S_j)$ for some i_j in I. The distance, R(x,y), from x to y is the minimum number of states for which there exists a training sequence of that length from x to y. The <u>learning time</u> of D is the maximum value of R(x,y) for x and y in S.

A <u>training algorithm</u> for D is a function q_z from S onto T, a subset of inputs, such that the sequence of states $(x=S_1,S_2,\dots,S_n)$, where $S_{j+1} = h(q_z(S_j), S_j)$, $j=1,\dots,n-1$. is a training sequence T(x,z) for all x in S. A <u>training set</u> for D is a subset, T, of inputs such that any function from S onto T is a training algorithm for any goal state in S. A <u>limited training set</u> for D is a subset, T, of inputs such that any function from S onto T is a training algorithm for a specific goal state, z. The 5–tuple $D_z(I') = (A_{I'}, S, \phi_z, F, g)$ is a <u>specialization</u> of D if I' is a subset of I, $A_{I'}$ is the set of all finite strings over I', and ϕ_z: $A_{I'} \times S$ ––> S, where $\phi_z(t^*, S_o) = S_n$ and $t^* = (t_1\, t_2 \dots t_n) \in A_{I'}$, and $S_j = h(t_j, S_{j-1})$ if $f(t_j, S_{j-1}) \notin d_z(t_j)$ and $S_j = S_{j-1}$ if $f(t_j, S_{j-1}) \in d_z(t_j)$. Let $A^*_{I'}$ be the subset of finite strings over I' that contain each member of I' at least once. The learning automaton D is <u>general purpose</u> if for any z and I', any subset of $A^*_{I'}$ is a limited training set for $D_z(I')$ with goal z.

The following theorem has been established previously by the authors and is the basis for applications in this paper.

MAIN THEOREM

If D=(I, S, h, F,g) is a general purpose learning automaton and $f(t_j, h(t_j,S_1)] = f[t_j, h(t_j,S_2)]$ iff $f[t_j,S_1] = f[t_j,S_2]$ where N_j is the number of values of $f[t_j, s]$ for $s \in S$ and M equals the number of inputs in I, then the characteristic semigroup of D is a finite group, G, with the following structure:

1. If Gx is the subgroup that leaves an initial state x invariant then $G = [Q_1 \cdot Q_2 \cdot \cdots Q_M] \cdot G_x$, where Q_j is homomorphic to the cyclic group C_{N_j} and $Q_j \simeq C_{N_j}$ if G_x is trivial.
2. If G is commutative then $G \simeq [C_{N_1} \times C_{N_2} \times \cdots \times C_{N_M}]$.

APPLICATION TO GENERAL NETWORKS

The first step will be to make a discrete approximation to the neural network training paradigm. Within this approximation network training sequences map into training sequences of an associated finite learning automaton. The second step will be to relate the learning time of the latter to its local structure.

Let S'' = the complete set of values for all weights in the network. Let I' = a finite set of training inputs. Let Z = the set of all integers. Assume that in a discrete approximation to the training paradigm there exists an $h' : Z \times I' \times S''$ –––> S'' which is a one–to–one mapping of S'' onto S'' for each member of (Z x I') and such that if t is a training input and if the network is in state r, the next state is given by $h'[t^m, r]$ where m is an integer. In this notation $h'[t^j, r] = h'[t, h'[t^{j-1}, r]]$ and $h'[t^{-1}, r] = h^{-1}[t, r]$. One can think of h' as providing a set of infinitesimal or incremental transformations out of which all others can be generated to a first approximation.

Now consider an initial state, X_o, for the network. Let S' = the set of all states generated from X_o by successive application of h'. Let S be a finite subset of S'. We will now define a finite learning automaton, D={I,S,h,F,g}, into which can be mapped network training sequences that remain in S. In applying the theory the subset S is chosen so as to include those network training sequences that one wishes to study.

If $I' = (t_1, t_2, \dots, t_M)$ then let $I = (t_1, t_2, \dots t_M, t_{M+1} \dots t_{2M})$ where t_{j+M} is defined as follows: $h'(t_{j+M}, x) = h'(t_j^{-1}, x)$ for all x in S'.

Now consider the set $h'(t_j,S) = V(t_j)$ and $t_j \in I$. Since for all t_j in I, $h'(t_j,x)$ is an invertible transformation of S', $|V(t_j)| = |S|$. Also $|V(t_j) - S| = |S - V(t_j)|$. Let B_j be a one–to–one map from $V(t_j)$ onto S that leaves $(V(t_j) \cap S)$ invariant and if $x \in (V(t_j) - S)$ then $B_j[h'(t_j,x)] = x$. Let h: I X S ––> S be defined as $h(t_j, x) = B_j[h'(t_j, x)]$ for x in S and t_j in I. The transformations defined by h are invertible transformations of S that have the property that if $y = h(t_j, x)$ and x and y are in S then $y = h(t_j, x)$.

Then D is a finite learning automaton. The characteristic semigroup generated by the elements of I is a finite group determined by h. Let Q be the maximum order for the elements in I.

It is assumed that between any y_o and y_m in S the network has a training sequence generated by h and within S. Let $L(y_o,y_m)$ equal the minimum length of all such sequences. Then <u>Network Training Time over S</u> is the maximum value of L(x,y), as x and y range over S, divided by Q. The factor Q is introduced because in using h' to train the network instead of h many steps, up to Q, are combined into one. There is then a corresponding sequence in D of no greater length. Let LT (D) be the learning time of D . Then LT (D) $\leq$ Q * [Network training time over S]

It is desirable for the network to be general purpose. The only constraint is that the training sequences are to lie within S. (1) Starting with any initial state , x, in S (2) any subset T of training inputs is selected and (3) any desired outputs for these are selected. Let g'_T (z) be a set of states in S and including z which has these desired outputs and let d_z (t_i) be a set of desired outputs for t_i in T. It is further assumed that (4) the order of presentation of the training inputs does not matter as long as each is presented sufficiently often and t_i is not presented if $f(t_i,s) \in d_z$ (t_i) or if h $(t_i,x) \notin$ S. A network is a <u>general purpose network over S</u> if under these four conditions a goal state within $g'_T(z)$ is reached in a finite number of steps.

The network is general purpose over S iff D is a general purpose learning automaton. This results from a one–to–one correspondence of training sequences between the network and D. Conditions (1), (2) and (4) for the network are equivalent to the requirement that any subset of A $_T$ is a limited training set for D_z (T), a specialization of D. By conditions (2) and (3) z and T may be chosen arbitrarily.

Next we wish to relate the learning time of D to its local structure. Consider a single, fixed unit in the network which has inputs, an output, and a subset of the adjustable weights. Let S_u be the set of all values for this subset. Let $\hat{S}$ be the set of all values for the remaining weights. There is then a mapping, ρ: S —–> S_u where $\rho(r)$ has the same unit weights as r and a similar mapping $\hat{\rho}$:S –––> $\hat{S}$. Each element of (I x $\hat{S}$) defines a transformation of S_u. Let I_u = (I x $\hat{S}$). Then h_u:I_u X S_u –––> S_u is defined as follows: $\rho[h(t,x)] = h_u\ [(t,\hat{\rho}(x)), \rho(x)]$. Let g_u (z) = $\rho[g(\rho^{-1}(z))]$. If g (y) = {y}, then $g_u(z)$ = {z}. In order to classify states in D_u according to output two new functions C: S X I —–> 2^S and C_u: S_u X I_u —–> 2^{S_u} are introduced. C (x,t) = {y $\in$ S :f(t,y) = f(t,x)}. Then C_u is defined by $\rho[C\ (x,t)] = C_u\ [\ \rho(x),\ (t,\hat{\rho}(x))]$.

Every training sequence in D maps into a training sequence in D_u. Also for any two states, x and y, in S_u there will be states x' and y' in S such that $\rho(x') = x$ and $\rho(y') = y$. It is assumed that the learning time of D is not infinite. Then there will be a training sequence, T(x',y') , in D of finite length. This will map into a training sequence, T(x,y) in D_u, of no longer length. Since x and y were arbitrary it follows that the learning time of D_u is less than or equal to that of D. LT $(D_u) \leq$ LT (D)

Let $\bar{I}_u$ equal the set of equivalence classes of members of I_u that effect the same transformation of S_u. Now consider a special case meeting the two conditions: (1) the projection of each member of $\bar{I}_u$ into $\hat{S}$ equals $\hat{S}$ and (2) $C_u[x',t'_1] = C_u[x',t'_2]$ if t'_1 and t'_2 belong to the same equivalence class of I_u. Condition (1) means that for any state of the global network and any of the possible transformations of S_u there is a global input in I that will effect that transformation. Condition (2) means that the output of D_u depends only on its state and the equivalence class of the input. Let $\bar{D}_u$ = {$\bar{I}_u$, S_u, h_u, F_u, g_u}. LT $(\bar{D}_u)$ = LT(D_u). We will call a $\bar{D}_u$ meeting conditions (1) and (2) a <u>local structure</u>. A local structure is characterized by having a local input that determines in conjunction with its state the output and incremental state transition.

An example of a local structure is an input unit that is an artificial neuron whose single output is given by $\psi(\vec{t} \cdot \vec{w})$ where $\vec{t}$ is an input vector in $\bar{I}_u$ and $\vec{w}$ is a weight vector in S_u. The incremental updating of $\vec{w}$, effected by h, would also depend only on $\vec{t}$ and $\vec{w}$. An example of the latter would be a generalized delta rule where $\Delta\vec{w} = m\,\eta\,\vec{t}$, η is a small constant, and m is an integer that depends on the global network state and input. In this case the incremental transformation would be given by $d\vec{w} = \eta\,\vec{t}$.

If D is general purpose and $\bar{D}_u$ is a local structure, then $\bar{D}_u$ must be general purpose. This can be seen as follows. Choose an arbitrary goal z' and arbitrary subset T' of $\bar{I}_u$. Then there is a z in S such that $\rho(z) = z'$. Each member of T' is a subset of (I x $\hat{S}$). Let T $\sqsubseteq$ I be the union of the projection of each of these subsets onto I. Let x' be an arbitrary initial state in $\bar{D}_u$. Let (r'_o, r'_1,r'_2,...) be a sequence of states in S_u generated by the input sequence in T', (t'_1, t'_2, ...), and the relation $r'_j = h_u\ (t'_j, r'_{j-1})$ where r'_o = x'. Then there will be a corresponding sequence of states (r_o, r_1, r_2, ...) in S and a sequence in T, (t_1, t_2, ...) such that $\rho(r_j) = r'_j$, $r_j = h\ (t_j,r_{j-1})$, and

(t_j, r_{j-1}) is in the same equivalence class of $\bar{I}_u$ as t'_j. The existence of $(t_1, t_2, \ldots)$ is insured by condition (1) in the definition of local structure for $\bar{D}_u$.

Choose the sequence $(t'_1, t'_2, \ldots)$ so that the elements are from T' in any desired order, the only constraint being that $C_u[r'_{j-1}, t'_j] \neq C_u [z', t'_j]$. The sequence $(t_1, t_2, \ldots)$ will then have the property that all of its members are from a subset of T and $C [r_{j-1}, t_j] \neq \underline{C} [z, t_j]$. This follows from the definition of C_u and condition (2) in the definition of local structure for $\bar{D}_u$.

Since D is general purpose there will be an m such that $r_m = z$. Thus $r'_m = z'$ and $\bar{D}_u$ is also general purpose. It may be possible to pick a $\bar{D}_u$ that is not a local structure and show directly that it is general purpose. In any event, it is assumed below that $\bar{D}_u$ is general purpose.

The final step is to impose the further orthogonality condition: $C_u [h_u (t_j, S_1), t_i] = C_u [h_u(t_j, S_2), t_i]$ iff $C_u [S_1, t_i] = C_u [S_2, t_i]$. Then the main theorem can be invoked to deduce the structure for the characteristic group of $\bar{D}_u$. $\bar{I}_u$ is a finite set. Let $M = |\bar{I}_u|$. Let N_j equal the number of different values for $C_u (x, t_j)$ as x ranges over S_u.

As discussed previously by the authors (2), the group structure of $\bar{D}_u$ in general will contain a large cyclic group of order $(N_1 N_2 \cdots N_M)$. Let N be the average size of N_j. Then if the training inputs are repetitively presented to the network in a fixed order, the network will have a worst case training time of $[P N^M / Q]$ where $P = |I|$.

A special case can arise when $h_u (t_1, h_u (t_2, x)) = h_u(t_2, h_u (t_1, x))$ and $N_1 = N_2 = \ldots N_M = N$. Then if the training inputs are repetitively presented to the network in a random order, the network will have a worst case training time of $[P M N/Q]$ (2).

A second special case is when $h_u(t_1, h_u (t_2, x)) \neq h_u (t_2, h_u (t_1, x))$ and $M = 2$: Then the network will have a worst case training time of $[P (N_1 + N_2) / Q]$ independent of the order of presentation of training (2).

SUMMARY

These results on the training time for the global network depend only on the presence of the orthogonality condition in some subset of the network. This is provided that the finite approximation to the network with incremental training is general purpose.

One local structure in which the orthogonality condition is easily satisfied is a single neuron whose output depends on a linear sum of weights. That is, $(\vec{t} \cdot \vec{w})$ determines the output, where $\vec{w}$ is the weight vector and $\vec{t}$ is the local input. Let θ_{ji} be the incremental adjustment in w_i when the training input is t_j. If the θ_{ji} are constants, it is easy to verify that the orthogonality condition holds and that the transition function is commutative. This is the case, for example, when a delta rule is used to modify the weights. Then the first special case discussed above will apply or the worst case training time will be exponential in M.

Now assume that θ_{ji} is a function of $(\vec{t}_j \cdot \vec{w})$ but that $\vec{t}_k \cdot \vec{\theta}_j$ is a constant if $k \neq j$. In this case, the orthogonality condition still holds but the transition function will be in general non–commutative. Then the second special case will apply.

REFERENCES

1. Clingman, W. H. and Friesen, D. K., "Algebraic Analysis of Goal–Seeking Automata," Texas A&M University Computer Science Department Technical Report, TAMU–89–001, College Station.

2. Clingman, W. H. and Friesen, D. K., "Algebraic Analysis of Goal–Seeking Systems", Texas A&M University Computer Science Department Technical Report, TAMU–89–002, College Station.

Extrapolatory Methods for Speeding Up the BP Algorithm

Hasanat M. Dewan
Department of Computer Science
Rutgers University
New Brunswick, NJ 08903
dewan@paul.rutgers.edu

Eduardo D. Sontag
SYCON-Center for Systems and Control
Rutgers University
New Brunswick, NJ 08903
sontag@fermat.rutgers.edu

August 8, 1989

Abstract

We describe a speedup technique that uses extrapolatory methods to predict the weights in a Neural Network using Back Propagation (BP) learning. The method is based on empirical observations of the way the weights change as a function of time. We use numerical function fitting techniques to determine the parameters of an extrapolation function and then use this function to project weights into the future. Significant computational savings result by using the extrapolated weights to *jump* over many iterations of the standard algorithm, achieving comparable performance with fewer iterations.

1 Introduction

In this note we describe some extrapolation techniques that appear to speed up convergence in back-propagation (BP). Numerical analysis techniques are often used in order to make BP more efficient than straightforward gradient descent, and recently it has been proposed that stiff ODE solvers be used instead of discrete approximations [1]. Our remark here is that *in addition* to these techniques one may be able to exploit the particular form of the differential equation being solved (or its discretization). More precisely, if one uses a sigmoidal response $\frac{1}{1+e^{-x}}$ for neurons, then a rough and nonrigorous analysis suggests that weights tend to grow logarithmically after many iterations, while they tend to behave as $1/t$ for intermediate values of the number of iterations, t. The logarithmic asymptotic behavior is suggested by an approximation of the differential equations [2], while the form $1/t$ is apparent from empirical observations of the way the weights change as a function of time. We use these observations as a basis of a speedup technique that uses extrapolatory methods to predict the weights in a network at a future time, given the weights up to the present. By extrapolating the weights, it is possible to economize on the iterations required by BP before an acceptable set of weights result. We use the general form

$$w(t) = a + b/t + c\log t$$

and variants where either b or c are forced to be zero. The parameters are fit via least squares techniques, and this function is then used to predict future weights. We then feed the projected weights back into the BP simulator and continue iterating. The phases of extrapolation and iteration are alternated until a satisfactory set of weights are obtained.

For simplicity, we base our experiments on a standard BP simulator, but the same technique could be used with any variants such as those using stiff ODE solvers. Although this work is empirical in nature, the simulation results are very encouraging, frequently affording considerable savings in computation time.

2 Weights as a Temporal Function

If the growth of the weights follow a logarithmic trend, given by the equation $w(t) = a + b\log t$ where a and b are constants and t represents time or the number of iterations, then for large t the expression $t(w(t+1) - w(t))$ would have to approach a constant since

$$t(w(t+1) - w(t)) = t\frac{w(t+1) - w(t)}{(t+1) - t} \approx tw'(t) = t\frac{b}{t} = b$$

On the other hand, if the hyperbolic function $w(t) = a + \frac{b}{t}$ approximates the weights, then the expression $t^2(w(t+1) - w(t))$ should approach some constant for large t, since

$$t^2(w(t+1) - w(t)) = t^2\frac{w(t+1) - w(t)}{(t+1) - t} \approx t^2w'(t) = t^2\frac{-b}{t^2} = -b$$

To verify these possibilities, we set up a 2-2-1 (2 input, 2 hidden, 1 output unit) network to learn the XOR problem. The BP algorithm was allowed to run for some time after the network classified the four inputs for XOR correctly. Any output unit is considered to have classified correctly if the desired output is 1 and the activation is greater than 0.5, or if the desired output is 0 and the activation is less than 0.5. Some typical graphs for the products mentioned above are shown in fig. 1 as a function of t.

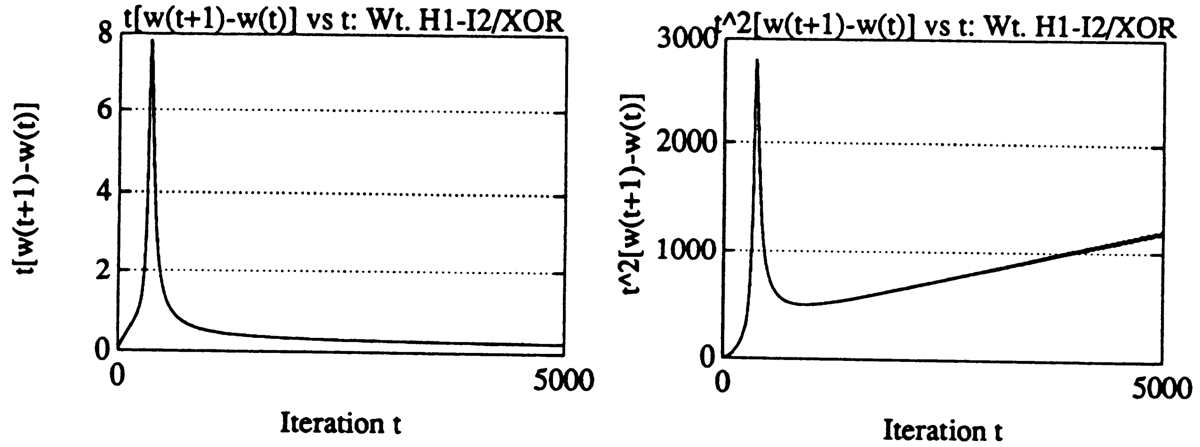
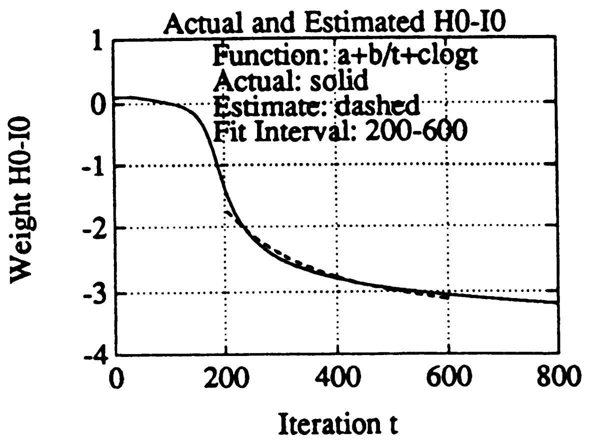


Figure 1: Growth of Weights may be a Log or Combined Hyperbolic-Log Function

It appears from the graphs in fig. 1 that the product *t(w(t+1)-w(t))* approaches some constant value as t becomes large, hence the growth of the weights may indeed be logarithmic. However, the product $t^2(w(t+1) - w(t))$ is asymptotically a straight line. Thus, for some constants B and C, $t^2w'(t) \approx B + Ct$. Dividing by t^2 and integrating both sides, we get $w(t) = A + B/t + C\log t$. Thus the weights seem to follow a combined hyperbolic-logarithmic evolution. Near zero, this is mostly hyperbolic, while for large t it is logarithmic.

In fig. 2 we show typical weight curves from the XOR example, superposed with the hyperbolic-logarithmic functions that approximate them. The actual data is shown in solid lines, while the functions are shown in dashed lines. It is easy to see that the functions approximate the actual weights quite closely.



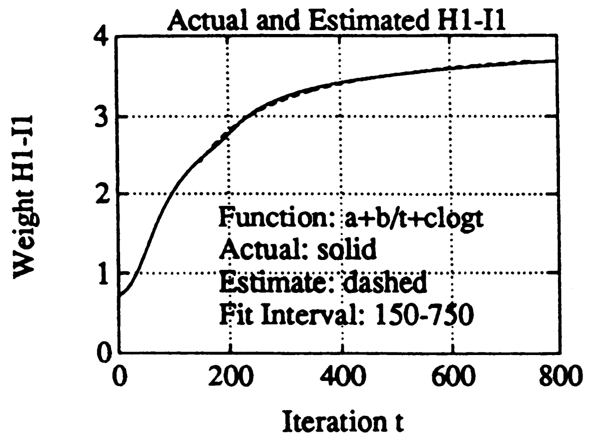


Figure 2: Some Actual and Estimated Weights from Hidden Layer to Input Layer

3 Experiments with Various Networks

3.1 Extrapolation Procedure

Briefly, the extrapolation procedure consists of first obtaining a value t_c of t for which a given network learns a certain problem. This is the 100% learning point, indicated by the fact that all output units match their desired values according to the following criterion: An output unit is considered to have classified correctly if the desired output is 1 and the activation is greater than 0.5, or if the desired output is 0 and the activation is less than 0.5. For our experiments, we obtained t_c by averaging over several runs of training the network in question. However, this *guessing* operation can be somewhat automated by noting that as a rough approximation, t_c can be considered to be directly proportional to the sum of number of input and output units, while it is inversely proportional to the number of hidden units, and then developing some heuristics based on these observations. It should be mentioned that such heuristics can only provide approximate values of t_c, and will not perform well for every problem.

After obtaining t_c, we set the extrapolation starting point to $t_s = 0.5t_c$. We then fit the hyperbolic function $w(t) = a + b/t$ to typically 20 iterations of actual weight data starting at t_s. Once the constants are determined, we use the hyperbolic function to extrapolate the weights to $t_e = 2.0t_c$. The weights thus obtained are then fed back into the BP simulator, and it is allowed to run until it maps 100% correctly. We keep track of the total number of actual simulator iterations. This is denoted by t_a. It is frequently the case that $t_a < t_c$, indicating computational savings in training the network. The ratio $(t_c - t_a)/t_c$ is a measure of the improvement obtained.

At this point, the network has learned the training data. However, the normalized error per output unit may still be quite high. To reduce this error, we perform the following steps repeatedly: the combined hyperbolic-logarithmic function $w(t) = a + b/t + c\log t$ is fit to approximately 20 points of weight data and the the weights are extrapolated for an aditional interval in the range $2.0t_c$ to $3.0t_c$. The weights are then fed back and the simulator restarted for $0.25t_c$ iterations, and the process is alternated until the error per output unit (a measure of convergence) reaches the desired value.

3.2 Test Cases

Our first test case is a 2-2-1 network, learning the Excusive OR function. The next test case is a 3-3-3 network which maps its binary inputs to their two's complement. The last case is a 3-2-8 network that learns the 3-to-8 decoding function for binary inputs.

4 Summary of Results

The results obtained by following the extrapolation procedure outlined above as applied to the three test cases is shown in the two tables below. The first table summarizes for three networks, the percent improvement in terms of actual iterations of BP that was obtained in mapping inputs to outputs 100% correctly by using extrapolation. For the same three networks, the second table shows the normalized error per output unit at the iteration when all outputs were correct (i.e at t_c), the normalized error at $4.0t_c$ obtained by extrapolating from t_c for an iterval of $3.0t_c$, and a percent reduction in the normalized error per output unit.

Network	t_c	t_s	t_a	% Improvement
2-2-1 XOR	300	150	162	46
3-3-3 Two's Compl	813	407	647	20
3-2-8 3 to 8 Decoder	1468	734	1221	17

Network	*Normalized Error per Output Unit at t_c*	*Normalized Error per Output Unit at $4.0t_c$*	*% Reduction in Norm. Error*
2-2-1 XOR	0.1499	0.0321	78
3-3-3 Two's Compl	0.0293	0.0127	56
3-2-8 3 to 8 Decoder	0.0319	0.0121	61

5 Conclusion

We have shown that extrapolatory techniques may substantially increase the speed of learning *and* the speed of convergence in networks using the BP algorithm. This provides motivation for constructing parametrized BP simulators with integrated ability for extrapolating weights using specified functions and heuristics. It is observed from our experiments that the particular extrapolation function can affect the acceleration of learning to a considerable degree. Discovering the extrapolation functions that work best requires further work.

References

[1] A. J. Owens and D.L. Filkin, *Efficient Training of the Back Propagation Network by Solving a System of Stiff Oridinary Differential Equations; in Proceedings IJCNN, 1989, pp. II-381:II-386.*

[2] Eduardo D. Sontag, *Some Remarks on the Backpropagation Algorithm for Neural Net Learning, SYCON Report 88-02, Rutgers Center for Systems and Control, Dept. of Mathematics, Rutgers University, July 1988.*

[3] W. M. Kolb, *Curve Fitting for Programmable Calculators, Syntec Inc., Bowie, MD 20716.*

[4] J. L. McClelland and D. E. Rumelhart, *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises, MIT Press, 1988.*

Accelerated Back Propagation using Unlearning based on Hebb Rule

MASAFUMI HAGIWARA

Dept. of Elec. Eng.
Facul. of Sci. and Tech.
Keio University
3-14-1 Hiyoshi, Kohoku-ku, Yokohama
223 Japan

Abstruct Accelerated Back Propagation using Unlearning based on Hebb rule (Hu-BP) is proposed. The concept of "unlearning" is founded on a biological assumption and is used for learning of the Boltzmann machines. However, the unlearning has not been considered associated with the back propagation. The proposed Hu-BP is very simple, effective, and that it can be used combined with other acceleration methods. According to the simulation, the convergence time using the proposed Hu-BP is reduced at least by two figures compared to the networks using a simple back propagation.

1.Introduction

Back propagation learning algorithm is the most popular learning paradigm, and has been used to perform a variety of input-output mapping tasks for recognition, generalization, and classification [1-4]. However, its high performance results are attained at the expense of training time, which is a major obstacle for training [5]. So, it is very important to shorten the training time of the back propagation.

There are several ways to accelerate convergence:
(1) Model a steeper error surface without flat spots[5].
(2) Carefully choose the parameters such as the step size, the momentum, and so on [5].
(3) Use a concept of "Artificial selection of hidden units"[6].
(4) Define an appropriate learning strategy[5].
However, methods (1)-(3) are seemed to be rather difficult to optimize their parameters. Method (4) can be used combined with the other methods including the proposed method because it is about learning strategy.

In this paper, I propose an accelerated Back Propagation algorithm using Unlearning based on Hebb Rule (Hu-BP). The concept of "unlearning" is founded on a biological assumption and is used for learning of the Boltzmann machines [1]. However, the unlearning has not been considered associated with the back propagation and its easy application to the back propagation tends to cause divergence of the networks. The proposed method is very simple, effective, and that it can be used combined with other methods. According to the simulation, the convergence time using the proposed method is reduced at least by two figures compared to the networks using a simple back propagation.

Following this introduction, the proposed algorithm Hu-BP is explained in Sec.2. In Sec.3, computer simulation results are shown.

2. Back-propagation algorithm using unlearning based on Hebb rule (Hu-BP)

According to Ref.[1], Crick and Mitchison have suggested that a form of reverse learning might occur during REM sleep in mammals. It was based on the assumption that parasitic modes develop in large networks that hinder the distributed storage and retrieval of information. In learning algorithm of the Boltzmann machines, positive Hebbian learning occurs in phase(+), and negative one occurs in phase(-).

In the proposed Hu-BP, positive learning corresponds to the error back propagation, and negative one corresponds to the Hebbian learning.

Now, I derive the Hu-BP algorithm using energy function.

The cost function of the back propagation is defined as

$$E_1 = \frac{1}{2} \sum_p \sum_j (t_{pj} - o_{pj})^2 \tag{1}$$

where, *p* is an index over cases, and *j* is an index over output units, *o* is the actual state of an output unit and *t* is its desired state.

In the proposed Hu-BP algorithm, an energy function

$$E_2 = - \sum_{i<j} w_{ji} o_j o_i + \sum_j \theta_j o_j \tag{2}$$

is included in the cost function, where w_{ji} is the weight from the *i*-th to the *j*-th unit, o_j is the state of the *j*-th unit, and θ_j is a threshold. Here, the cost function of the proposed algorithm is defined as

$$E = \eta\, E_1 - \beta(m)\, E_2 \tag{3}$$

where, m is the number of learning sets. The rule for changing weights can be calculated as in Ref.[1].

$$\Delta w_{ji} \propto - \frac{\partial E}{\partial w_{ji}} \tag{4}$$

Thus we can get

$$\begin{aligned} \Delta w_{ji} &= \eta\, \delta_j o_i - \beta(m)\, o_j o_i \\ &= \eta\, [\, \delta_j - \gamma(m)\, o_j]\, o_i \qquad [\, \gamma(m) = \beta(m) / \eta \,] \end{aligned} \tag{5}$$

where, η is a learning constant, δ_j is the same parameter used in Ref.[1]. The first term component in Eq.(5) corresponds to positive learning, and the second one corresponds to negative learning by Hebbian learning. $\gamma(m)$ is a decreasing function, because the unlearning component should be decreased as time increases.

3. Simulation results

In this section, we show computer simulation results to demonstrate the effectiveness of the proposed Hu-BP algorithm by comparing to the networks using the simple back propagation.

The following simulation conditions are used.

1) Three layer network is used.
2) Momentum α is zero in order to make clear the performance difference and reduce the influence by this parameter.(Parameter α is a constant which determines the effect of past weight changes on the current direction of movement in weight space.)
3) The number of input units is 15.
4) The number of output units is 20, namely 20 patterns which are shown in Figure 1 are used for learning and recognition.
5) The range of sigmoid function output is between -1 and 1.

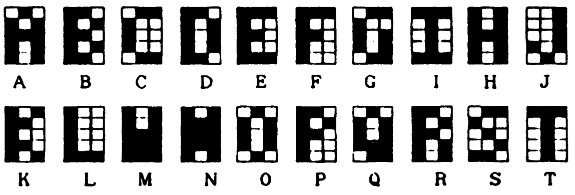


Fig.1 20 learning patterns used for simulation.

6) True recognition is regarded as the state where every sign of each output unit is same to that of the corresponding desired output.
7) The maximum number of learning sets is 10000 (200000 times learning in total).
8) The function $\gamma(m)$ is

$$\gamma(m) = \begin{cases} \frac{1}{2} & : m < 30 \\ \frac{1}{2}(0.8)^c \quad c=[\mathrm{int}(m/15)]-1 & : m \geq 30 \end{cases} \tag{6}$$

where, m is the number of learning sets (one learning set is one time learning of each pattern, so it means 20 times learning).

Table 1 shows the results of simulations. In the cases where the conventional back propagation algorithm is used, only half of the trials can be converged. On the other hand for the proposed Hu-BP, all of the trials can be converged. This fact means that the Hu-BP algorithm can not only reduce the learning time but also have an effect of avoiding local minima.

It also can be seen that the convergence time using the Hu-BP is reduced at least by two figures compared to the networks using the simple back propagation. And it is noteworthy that the variance of the convergence time by the Hu-BP is very small: all the trials using the Hu-BP are converged within one hundred and some.

η		Conv. BP	Ref.[6]	Hu-BP
0.1	Learning times [sets]	3186	444	146
	(Improvement ratio)	(1.0)	(7.2)	(21.8)
0.2	Learning times [sets]	1306	247	112
	(Improvement ratio)	(1.0)	(5.3)	(11.7)
0.3	Learning times [sets]	>10000	351	134
	(Improvement ratio)	(-)	(>28.5)	(>74.6)
0.4	Learning times [sets]	6334	716	138
	(Improvement ratio)	(1.0)	(8.8)	(48.0)

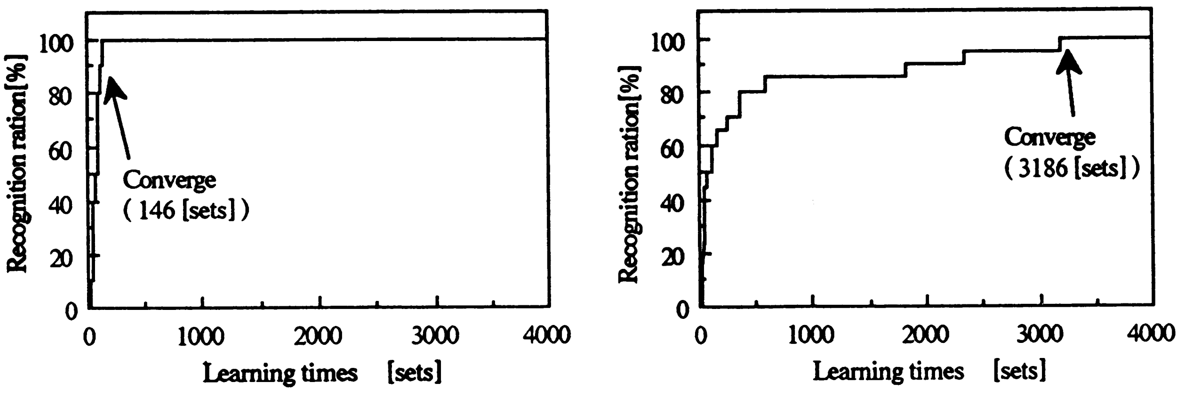
(a) 20 Hidden units

η		Conv. BP	Ref.[6]	Hu-BP
0.1	Learning times [sets]	>10000	912	113
	(Improvement ratio)	(-)	(>11.0)	(>88.5)
0.2	Learning times [sets]	2257	621	113
	(Improvement ratio)	(1.0)	(3.6)	(20.0)
0.3	Learning times [sets]	>10000	1546	125
	(Improvement ratio)	(-)	(> 6.5)	(>80.0)
0.4	Learning times [sets]	>10000	1199	141
	(Improvement ratio)	(-)	(> 8.3)	(>70.9)

(b) 30 Hidden units

Table 1 The times of learning.

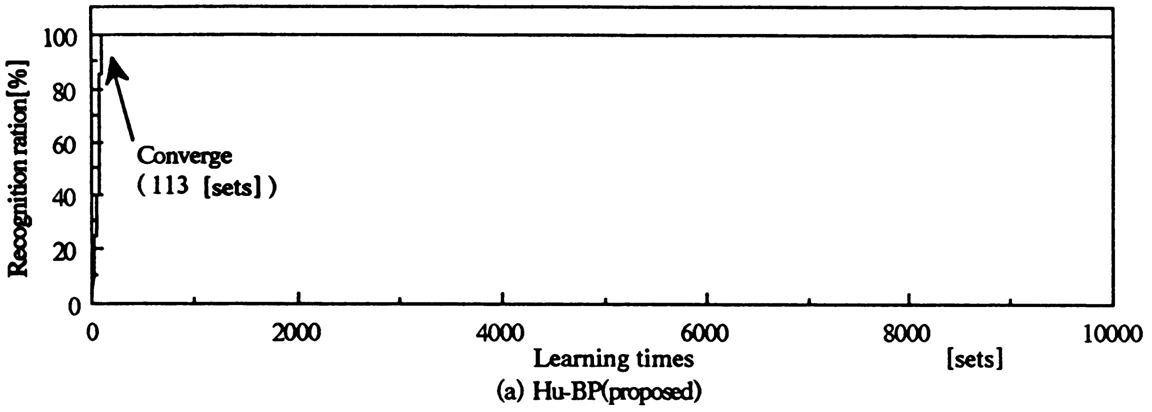
Figures 2 and 3 show the relations between learning times and recognition ratio characteristics, where η=0.1 and the numbers of hidden units are 20 and 30, respectively. From these figures we can see that the recognition ratios of the conventional BP increase very slowly. On the other hand, those of the proposed Hu-BP are very quick. And they are temporarily decreased at many points by unlearning. The reason of this phenomenon is that the network causes perturbation to find a global minimum quickly.



(a) Hu-BP (proposed)

(b) conventional BP

Fig.2 Learning times vs. recognition ratio characteristics (20 hidden units, η=0.1).



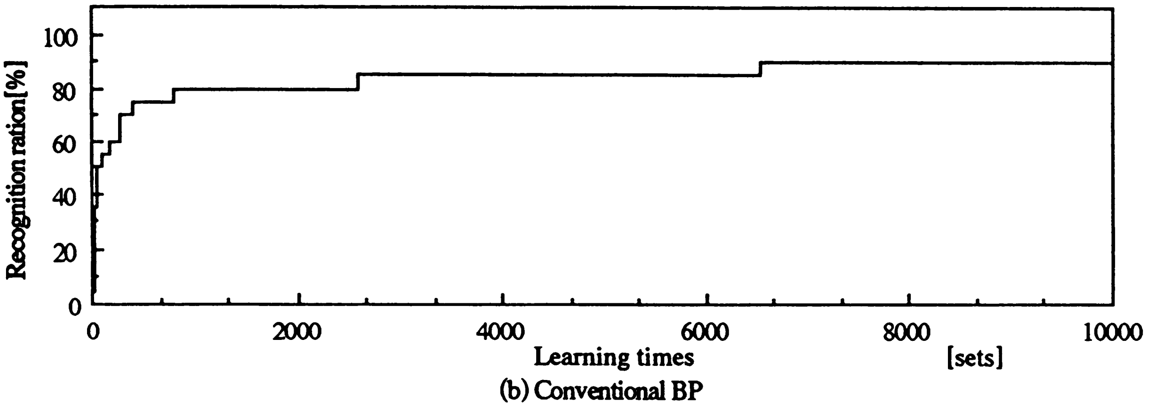


Fig.3 Learning times vs. recognition ratio characteristics (30 hidden units, $\eta = 0.1$).

4. Conclusions

Accelerated back propagation using unlearning based on Hebb rule (Hu-BP) has been proposed. The concept of "unlearning" is founded on a biological assumption and is used for the learning of the Boltzmann machines. The proposed Hu-BP algorithm utilizing unlearning is very simple and effective.

According to the simulation, the convergence time using the Hu-BP is reduced at least by two figures compared to the networks using a simple back propagation. And it has an effect of avoiding local minima.

The proposed Hu-BP algorithm can be used combined with other acceleration methods.

ACKNOWLEDGEMENT The author would like to thank Prof. Masao Nakagawa of Keio university.

REFERENCES

[1] D.E.Rumelhart, J.L.McClelland and the PDP Research Group : "Parallel Distributed Processing", MIT Press, 1986.
[2] R.P.Gorman and T.J.Sejnowski :"Analysis of hidden units in a layered network trained to classify sonar targets", Neural Networks, vol.1, 1, pp.75-89, 1988.
[3] A.Waibel, T.Hanazawa, G.Hinton, K.Shikano, and K.J.Lang : "Phoneme recognition using time-delay neural networks", IEEE trans. Accoust., Speech, Signal Processing, vol-37, 3, pp.328-339, March 1989.
[4] D.Nguyen and B.Widrow :"The truck backer-upper: an example of self-learning in neural networks", Proc. of IJCNN'89, II p.357-363, June 1989.
[5] H.Sawai, A.Waibel, P.Haffner, M.Miyatake, and K.Shikano :"Parallelism, hierarchy, scaling in time-delay neural networks for spotting Japanese phonemes/CV-syllables", Proc. of IJCNN'89, II p.81-88, June 1989.
[6] M.Hagiwara and M.Nakagawa:"Supervised learning with artificial selection", Proc. IJCNN'89, II p.611, June 1989.

Self-Organizing Autoassociative Dynamic Multiple-Layer Neural Net for the Decomposition of Repetitive Superimposed Signals

M.H.Hassoun, J.Song, S-M Shen
Department of Electrical and Computer Engineering

A.R.Spitzer
Department of Neurology

Wayne State University, Detroit, Michigan 48202

ABSTRACT

A multi-layer self-organizing neural network-based technique for the automatic identification and extraction of repetitive superimposed pulses in noisy signals is proposed. No *a priori* knowledge of the shape or amplitude of the pulses is assumed. The signal is composed of the superposition of an unknown number of statistically independent asynchronous pulse trains and noise. The identification of the underlying pulses is accomplished through a novel unsupervised learning technique based on the back-error propagation learning rule. During the learning phase, a multiple-layer autoassociative neural network is modified (weight amplitudes and number of neurons) such that a nominal representation is discovered for each underlying pulse. In a second phase (the retrieval or extraction phase), direct feedback between input and output is added to the trained network and the resulting dynamic network is used to map activity segments of the raw signal into their nominal or filtered representations. This is accomplished by iterating the network until it converges to one of its local minima representing learned pulse shapes.

1. INTRODUCTION

This paper reports on a neural network architecture and an associated learning strategy that combine the following concepts: back error propagation in multiple-layer neural nets (Amari, 1967; Werbos, 1974; Rumelhart, Hinton, & Williams, 1986), self-organization through constrained autoassociative learning (Kuczewsk, Myers, & Crawford, 1987), generalization through the adaptive generation of hidden-layer bottlenecks (Rumelhart, 1988; Kruschke, 1988; Chauvin, 1989; Hanson & Pratt, 1989; Mozer & Smolensky, 1989), and concept formation by means of convergent network dynamics (Amari, 1977; Hopfield, 1982; Hassoun, 1989).

The primary goal of the proposed neural network is the unsupervised recognition and decomposition of signals with superimposed repetitive pulses, as shown in Figure 1. The network has no *a priori* knowledge of the shape or amplitude of the underlying signal pulses. The network only assumes that all pulses to be learned have a pronounced peak, an estimated pulse duration upper bound, and a repetitive pulse pattern. (Figure 2 shows an example of three consecutive pulses used in the simulations reported later in this study.) Furthermore, the pulses are assumed to be asynchronous and nonperiodic. Finally, the diagnosed signal is assumed to last long enough so that a rich set of isolated and superimposed pulse waveforms is generated. Physiologic signals (EMG, ECG, EEG, etc.) and vehicle electronic sensor signals, in the presence of noise and undesired radiation coupling, are examples of the above signals.

Section 2 presents the proposed neural network architecture. Section 3 discusses the adaptive self-organizing algorithm employed. In Section 4, the net's learning capability and performance are reported based on a set of simulations. Conclusions are presented in Section 5.

2. NETWORK ARCHITECTURE

The problem at hand represents a difficult class of problems in signal processing and pattern recognition. This problem demands several processing stages: locating potential pulse activities within the given signal, discovering the number of different pulses (classes) present and their underlying nominal shapes (a pulse may be distorted and/or noisy in addition to being superimposed with other pulses), and using the nominal pulse shapes as templates or library vectors in order to classify signal activity segments according to some error (or distance) measure. In this work, we propose a three-layer (two-hidden and one-output layers) self-organizing neural network that is capable of capturing, representing, classifying, and reconstructing noisy superimposed pulses. No initial pulse representations are assumed; the proposed network relies only on the repetitive nature of a set of unlabeled signal activities.

Sampled signal segments (activity vectors) of forty samples each are used as inputs to the three-layer network. Activity vectors are selected such that signal peaks are aligned at the same position within the window of forty samples. Here, we assume that the signal duration covered by the forty activity vector samples represents an upper bound on all pulse durations present in the raw signal. An unlabeled training set consisting of all activity vectors with pronounced peaks is constructed. This is achieved by generating one forty-dimensional sampled training vector for each pronounced peak (peak amplitude exceeding 0.25) in the signal in Figure 1, with the peak amplitude located at a fixed location within the training vector. Here, prior knowledge about the sharp positive peaks of the underlying pulses is used effectively in locating and aligning potential pulse occurrence; this in turn reduces the computational load for shift invariance that would otherwise constrain the learning in the neural network.

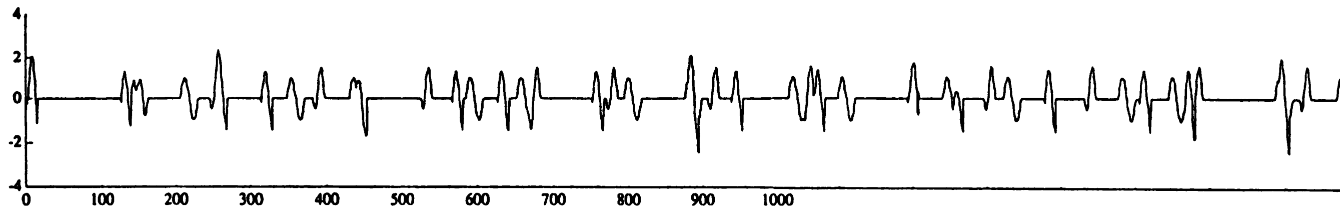


FIGURE 1. A signal of three superimposed pulse trains. Pulse occurrence is random.

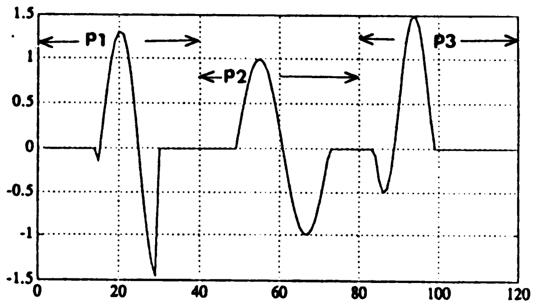


FIGURE 2. Three underlying pulse waveforms (p1, p2, and p3) of the signal in Figure 1.

Having identified a suitable training set, we now turn to the required neural network size and architecture. A multiple-layer neural network with two-hidden layers and one output layer is used. The first (from input) and second hidden layers have dimensions H and J, respectively, and employ neurons with sigmoidal activations operating over the activity interval [+1,-1]. The output layer has L=40 neurons with linear activations.

The choice of net architecture is closely related to the nature of the problem at hand. The network is applied in two phases: a learning/self-organizing phase and a retrieval phase. In the learning phase, activity vectors are presented one by one to the network (first interacting with the H-dimensional hidden layer) and a self-organizing learning strategy is used (this will be described in the next section) in order to capture the shapes of the underlying pulses. During this phase, the network minimizes its available neurons at the two hidden layers and forms a bottleneck at the second hidden layer (center layer). According to our training strategy, the neural network self-organizes such that the first layer acts as a feature discovery layer. The second hidden layer acts as an encoder which generates an efficient compact code (internal representation). Finally, during this phase, the output layer is constrained in a way such that an approximate mapping between the internal representation vectors (inside the cube $[-1,+1]^J$) and the sampled pulse space R^{40} is realized; i.e., the output layer is intended to reconstruct pulse waveforms.

In the retrieval phase, all learning is inhibited and the network is used to map training activity-vectors (as well as signal activity vectors not used during training) into their nominal superposition-filtered pulse waveforms (this is reminiscent of Amari's concept-forming nets). This is done by utilizing a dynamic architecture consisting of the trained three-layer net with a direct feedback connection from the output layer to the first layer. An investigation of the dynamics of the retrieval network reveals that stable internal representations lead to fast convergence to pulse-shape states having wide basins of attraction. (This will be shown in the simulations of Section 4.) This translates into having a network that starts with a corrupted (usually due to superposition) activity vector as its initial state and, later, dynamically converges to the closest pulse-shape state representing an underlying learned pulse waveform.

3. AN ADAPTIVE SELF-ORGANIZING STRATEGY

How is it possible to self-organize in a multiple-layer net? Kuczewsk, et al. (1987) and others have explored self-organization in multiple-layer networks through the use of the back-error propagation (BEP) learning algorithm.

Here, a multiple-layer neural network with a hidden-layer(s) bottleneck is used in an autoassociative learning mode employing the BEP algorithm. Input and target vectors are assumed identical (autoassociative learning) and are presented to the network simultaneously. In this configuration, and assuming an optimal number of hidden neurons, the network will evolve internal features represented in the middle layer that can be used to reconstruct the original training patterns.

In the present problem, a similar self-organizing strategy is used. However, a direct application of the BEP algorithm will fail to discover an internal hidden representation which will generalize from superimposed input pulses into underlying pulse-shape representations, not to mention the excessive training time of the standerd BEP algorithm. Another problem with the above self-organizing strategy is the difficulty of estimating the number of hidden neurons necessary for generalization. This latter problem has been recently addressed by several researchers (Rumelhart, 1988; Kruschke, 1988; Chauvin, 1989; Hanson & Pratt, 1989; Mozer & Smolensky, 1989) who have looked at modifying the cost function in the BEP procedure such that the network is constrained to discover a minimum number of generalizing hidden units. These techniques have used various penalty criteria for the elimination or suppression of redundant and/or slowly training units; weights (Rumelhart; Hanson & Pratt), activations (Chauvin), and energy (Mozer & Smolensky) based penalty criteria were employed.

In this work, we propose a BEP-based self-organizing/generalizing training strategy which addresses the above problems and is well suited for the problem at hand. First, consider the three-layer net signal/weight/error diagram shown in Figure 3. Here, x^k and $y^k = x^k$ represent the input and output activity vectors, respectively, o represents a layer's activation vector, and δ^k is the familiar back-propagated error vector due to the $\{x^k, y^k\}$ training pair. The indices i, h, j, and l correspond to input, first hidden, second hidden, and output signal components. The weights in all hidden units are updated according to the original BEP algorithm (with no momentum term): first hidden layer: $\Delta w_{ih} = \rho \delta_h x_i$ and second hidden layer: $\Delta w_{hj} = \rho \delta_j o_h$, where $\rho = \rho_o (\rho_m)^k$, $0 < \rho_o < 1$, ρ_m very close to but less than 1, and k is the training pattern number. The local update rule for the output layer weights is given by: $\Delta w_{jl} = \rho \delta_l o_j + \beta \rho w_{jl}$ with $\beta < \rho_o$.

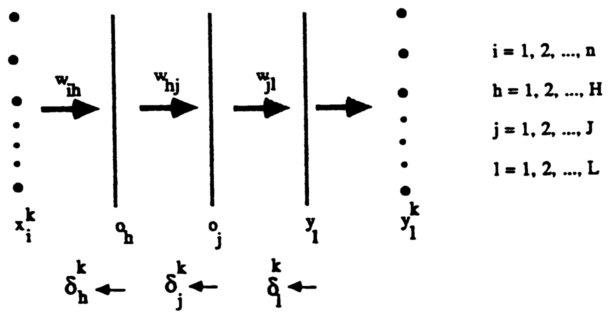


FIGURE 3. A three-layer neural network signal, weight, and back-propagated error diagram.

From the above weight update equations, it is seen that two modifications to the BEP algorithm have been incorporated: exponentially damped learning coefficients and output layer "forgetting" effects. The damping in the learning coefficient is very crucial in balancing learning against the emphasis of last-learned patterns (which could, in our problem, be undesirable highly-superimposed activity vectors). It also has the added advantages of learning "annealing" and allowing an initially relatively large learning coefficient to be used which accelerates the learning process. On the other hand, the forgetting effects at the output layer are very important in enhancing the learning of repetitive structured patterns as opposed to superimposed patterns and/or noise. This latter strategy also helps in realizing a more accurate mapping (reconstruction of pulse-shape patterns) between the second hidden layer and the output layer. It also has desirable effects on the generalization and self-organization of the hidden layers, indirectly through the propagation of more accurate δs from the output layer.

The only remaining problem that we have addressed is that of generalization through the dynamic optimization of hidden layer neurons. Here, we introduce a new strategy for hidden unit elimination based on back-propagated error magnitude distributions. This strategy allows for the elimination of undesired hidden units according to their history of error generation. The strategy also allows for unit elimination in networks with multiple hidden layers. More specifically, it allows for selective hidden layer reduction based on a prior knowledge of the hidden layer functionality; e.g., in our problem we employ the proposed unit elimination strategy such that both hidden layers are optimized, with the first layer being less constrained than the second layer, which leads to a gradual bottleneck in realizing feature extraction/encoder combinations layers. The following is the strategy we have employed for hidden layer size minimization:

0. Divide the training set into groups of K vectors each, with K being a small fraction of the total training set m.

1. Use the BEP algorithm as modified above and train the net with the first set of K sample. Set k_o to K.

2. Holding all network weights fixed, compute the accumulated error signals for all hidden neurons:

$$e_j = \sum_{k=k_o}^{K+k_o} \left(\delta_j^k\right)^2 \ ; \quad j = 1,2, \ldots, J \text{ for 2nd hidden layer neurons}$$

and

$$e_h = \sum_{k=k_o}^{K+k_o} \left(\delta_h^k\right)^2 \ ; \quad h = 1, 2, \ldots, H \text{ for 1st hidden layer neurons}$$

3. Compute the means and standered deviations for the accumulated error signals:

(μ_J, σ_J) for 2nd hidden layer and (μ_H, σ_H) for 1st hidden layer

4. One neuron j (from 2nd hidden layer) is deleted iff

$e_j > \mu_J + 1.4\sigma_J$ and $e_j > e_{i \neq j}$ for all i.

5. One neuron h (from 1st hidden layer) is deleted iff

$e_h > \mu_H + 1.8\sigma_H$ and $e_h > e_{i \neq h}$ for all i and no deletions are made in step 4.

6. After deletions, update the network weights by learning the present set of K patterns using the proposed learning algorithm. Set $k_o = k_o + K$. Stop if the limit on the number of training cycles is met, if not, Go To 2.

The first thing to note about the above redundent-hidden-unit elimination algorithm is the utilization of the accumulated back-propagated decision error signal δ over a sample subset of the training set. This differs from earlier-proposed penalty criteria for hidden-unit elimination. Here, a hidden unit is eliminated based 1) on a short history of its contribution to the output mapping inconsistencies and 2) on its behavior compared to all other units in a given hidden layer. Due to the nature of the training activity vectors, the choice of K in step 0 above is flexible. The larger K is, the more representative is the value of the accumulated error e in step 2. (One extreme is to choose K = m, the number of training vectors. However, this might not be possible in situations where only a fraction of the training vectors are available at a time.) The unit elimination recipe is then given in steps 4 and 5. Here, one unit in a given hidden layer is eliminated, after each K-vector presentation, if and only if it has the largest accumulated decision error e in that layer and if e exceeds a threshold determined by the distribution (mean and standard deviation) of all units' accumulated decision errors in that layer and a preset vigilance parameter (the 1.4 value used in the inequality of step 4). The above algorithm is also capable of realizing a gradual bottleneck when more than one hidden layer is used. This is accomplished according to step 5 above, where unit deletions may only be performed if no such deletions occur in the next higher level hidden layer. Another factor affecting the size of lower-level hidden layers is the magnitude of the associated vigilance parameter (set to 1.8 for the first hidden layer in our simulations). The vigilance value is set larger than that of the next higher-level hidden layer as seen in steps 4 and 5 above. This strategy allows the network to establish the desired cascade of a constrained feature extraction layer followed by a more constrained encoder layer/bottleneck.

4. NETWORK PERFORMANCE AND DISCUSSION

The performance of the above dynamic multiple-layer neural network and its proposed self-organizing training algorithm is evaluated using simulated signals of superimposed repetitive pulses of various degrees of complexity. Here, we report on the decomposition of one particular signal which is shown in Figure 1. The signal in Figure 1 represents the first segment of 1800-time units of a simulated 8000 time unit signal. This signal is constructed as the superposition of three pulse trains. Each pulse train is constructed from one fixed pulse waveform that repeats with a restricted random frequency. The three pulse waveforms used are labeled p1, p2, and p3 and are shown in Figure 2. The overall superimposed signal (refered to as the raw signal) of Figure 1 has a total of 192 pronounced signal peaks which are used to locate and align the activity-vectors of Section 2. An unlabeled training set of 192 activity-vectors is used in the following simulations. Visual inspection of the training set reveals the presence of 68 clean (nonsuperimposed) pulse waveforms: twenty-four, twenty-two, and nineteen waveforms belonging to p1, p2, and p3, respectively. Therefore, the training set has about 64% distorted (superimposed) pulse waveforms, each involving two or three pulses. It is to be noted here that the visual identification of the 68 clean pulses is made easy because of our prior knowledge of the shapes of the underlying pulses; this prior knowledge is not made available to the network. The network is supposed to discover these underlying pulse waveforms as part of its learning phase.

The network described in Section 2 is used. This initially had six neurons in the first and second hidden layers, respectively. An input window size of forty (+1 for bias) is used, with the leftmost activity-vector peak centered at the thirteenth input bit. The output layer has forty neurons with linear activations (slope =1). Only feed-next connections are allowed during the learning phase, with biases of +1 applied to all units. Initially, all weights are set randomly in the interval [+.1,-.1]. The BEP-based learning algorithm of Section 3 is used for training the network.

The following parameters were used: learning coefficient $\rho = .35(.995)^k$, number of learning cycles = 5 (a total of 192*5 = 960 presentations), K = 30 (i.e., 192/30 ~= 6 training vector subgroups), and output layer weight decay factor $\beta = .05$.

During the first cycle in the learning phase, the network eliminated three units from the second hidden layer, and thus reduced its neurons to three. In the second learning cycle, only one unit was eliminated from the first hidden layer. The learning proceeded from the second cycle to the fifth with no further hidden unit elimination. Learning stopped after the fifth cycle and the trained network was used in the dynamic retrieval phase as described in Section 2. Here, all 192 training activity vectors are tested and mapped by the trained network into "closest" underlying representations. Figure 4 (a-h) shows typical instances of pulse identification and reconstruction (the dashed graph corresponds to the input activity vector and the solid graph corresponds to the retrieved pulse waveform). In this simulation, the network has discovered four internal representations; three representations corresponded to the three underlying pulses, respectively, and one representation corresponded to a falsely identified pulse. In terms of network dynamics, each representation is manifested as a strong stable point in the R^{40} dimensional state-space. We were very impressed by the stability and speed of convergence of the retrieval network. The above simulation was repeated over thirty times, each time with new weight initialization and/or different learning parameters and K values (e.g., ρ_o =.15, .3, .4; K = 30, 40, 50). In most cases, the network discovered the five-three hidden layer arrangement and led to network dynamics comparable to the above. In some cases, the network learned two out of three representations (two pulse representations were merged into one) and one or two additional false representations (attractors) which attracted less than 10% of the activity-vectors. The network was also able to generalize and form meaningful internal representations when initialized between five and ten neurons/hidden layer. In all cases, five learning cycles seemed to be sufficient.

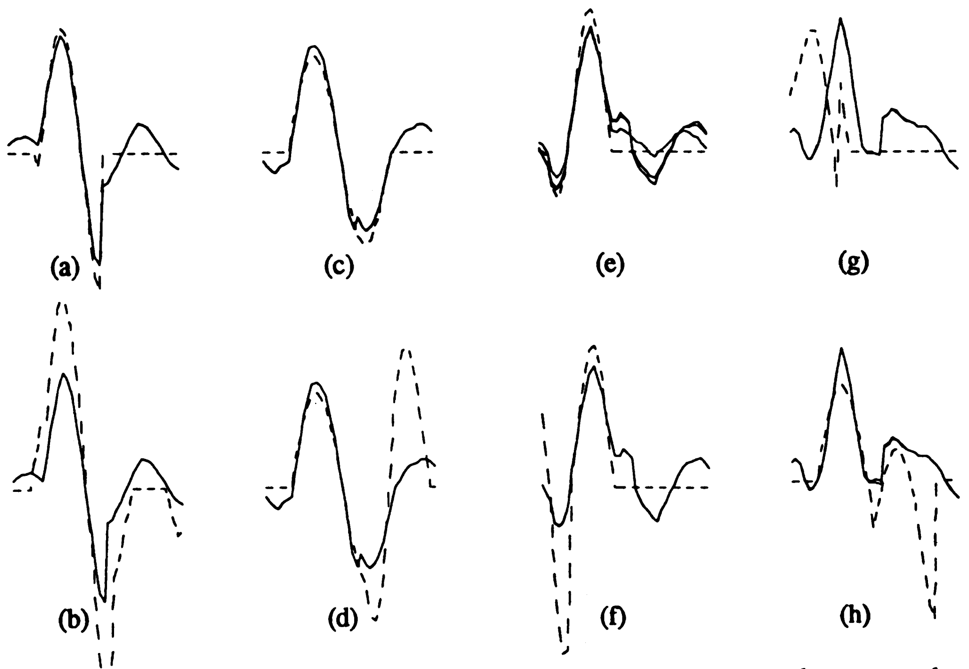


FIGURE 4. Typical instances of pulse identification and reconstruction. The dashed graph corresponds to the input activity-vector and the solid graph corresponds to the retrieved pulse waveform.

The quality of the reconstructed pulse waveforms (see Figure 4) and the classification/decomposition abilities of the above network are highly dependent on the complexity (degree of superposition of pulses and correlation between underlying pulse-waveforms) of the raw signal and the way the activity-vectors are represented at the input layer. In particular, one can easily notice the high degree of correlation between the three pulses of Figure 2, especially when aligned at their peaks. This alignment is important in the above simulation in order to alleviate difficulties with shifted signals. However, this same alignment strategy makes the underlying pulse waveforms highly correlated, which in turn affects the accuracy of the reconstructed pulse waveforms and the ability of the network to generalize. Despite these negative effects, the proposed network was still able to discover and resolve between all three pulse waveforms, and give relatively good reconstructions.

5. CONCLUSION

We have proposed a multiple-layer neural network for the identification and extraction of repetitive superimposed pulse signals, assuming no *a priori* knowledge of pulse shapes or repetition frequencies. The proposed network combines BEP learning, self-organization, constrained generalization, and concept formation processes in order to discover, decompose, and classify pulse waveforms from a set of unlabeled superimposed activity-vectors. A new hidden-unit elimination recipe was also provided and was shown to be effective in realizing generalizing hidden-layer bottlenecks. We have also demonstrated very interesting multiple-layer net dynamics,which were applied in superimposed pulse waveform decomposition through the dynamic retrieval of closest pulse shape matches. Overall, the proposed processing strategy is robust against network and learning parameter initializations. Finally, the proposed neural net and its associated self-organizing learning strategy can be very easily extended to problems involving the clustering and identification of unlabeled continuous-valued training vectors representing class features or actual sampled signals. Such extensions will be the subject of future work.

6. ACKNOWLEDGEMENT

This work was supported in part by a grant from the Whitaker Foundation.

7. REFERENCES

Amari S. (1967). A theory of adaptive pattern classifiers. IEEE Trans. Elec. Com., EC-16, 279-307.

Amari S., (1977). Neural theory of association and concept formation. *Biological Cybernetics*, **26**, 175-185.

Chauvin, Y. (1989). A back-propagation algorithm with optimal use of hidden units. In: *Advances in Neural Information Processing 1*, David S. Touretzky, ed., San Mateo, CA: Morgan Kaufmann, 519-526.

Hanson, S. J. and Pratt, L. Y. (1989). Comparing biases for minimal network construction with back-propagation. In: *Advances in Neural Information Processing 1*, David S. Touretzky, ed., San Mateo, CA: Morgan Kaufmann, 177-185.

Hassoun, M. H. (1989). Dynamic Heteroassociative neural memories. *Neural Networks*, 3(3).

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the U.S.A.*, **79**, 2445-2558.

Kruschke, J. K. (1989). Creating local and distributed bottlenecks in hidden layers of back-propagation networks. *Proceedings of the 1988 Connectionest Models Summer School*, D. Touretzky, G. Hinton, and T. Sejnowski (eds.), San Mateo, CA: Morgan Kaufmann, 120-126.

Kuczewiski, R. M., Myers, M. H., & Crawford, W. J. (1987). Exploration of backward error propagation as a self-organizational structure. *IEEE First International Conference on Neural Networks*, v. **II**, 89-95.

Mozer, M. C. and Smolensky, P. (1989). Skeletonization: A technique for trimming the fat from a network via relevance assessment. In: *Advances in Neural Information Processing 1*, David S. Touretzky, ed., San Mateo, CA: Morgan Kaufmann, 107-115.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323, 533-536.

Rumelhart, D. E. (1988). Parallel Distributed Processing. Plenary Session, *IEEE International Conference on Neural Networks*.

Werbos P. (1974). *Beyond regression: New tool for prediction and analysis in the behavioral sciences*. Ph.D. thesis, Harvard University.

Numerical Analysis and Adaptation Method for Learning Rate of Back Propagation

Junichi Higashino*, Bart L. de Greef, Eric H. J. Persoon

Philips Research Laboratories,
P.O.Box 80,000, 5600 JA, Eindhoven, The Netherlands

Abstract: Back Propagation (BP) normally requires two parameters: learning rate η and momentum α. Although these parameters play a very important role, few systematic selection rules have been reported. The authors have found two experimental facts. One is that the number of required iterations N is proportional to $(1-\alpha)/\eta$, and the other is that η can be normalized in terms of connectivity determined by the number of connections between a layer and not only its preceding layer but also its successive layer. If they are properly chosen, the number of iterations can be greatly reduced. A successful adaptation method for determining the learning rate has been obtained.

1. Introduction

Back Propagation (BP) requires a lot of repeated training calculations. In addition, since characteristics of its algorithm are not well understood, some empirical rules must be incorporated when setting parameters, in particular, learning rate η and momentum α. Having failed to select these parameters properly, BP requires iterations more than necessary or even fails to converge. It is therefore essential to clarify the basic properties of BP and to obtain a systematic rule for adjusting the parameter values, not only to realize a promising method but also to build a large-scale network for a practical system.

Since BP, including several variants, is basically a gradient method (steepest descent), it needs a step size or learning rate which can be determined using either fixed or adaptive values. Methods using fixed values were presented by Rumelhart[1], Sejnowski[2] etc., but no systematic rule for the selection of η was given. Methods using adaptive η were developed by Dahl[3], Vogl[4], Jacobs[5] etc. Dahl's method makes use of linear search for η, which is a typical one dimensional optimization method, but needs several iterations to determine an optimum η value. Methods of Vogl and Jacob, which are empirically derived, update η value without linear search. In addition to these steepest descent methods, it is also possible to incorporate other mathematical methods. Makram-Ebeid[6] presented an acceleration method using conjugate gradient method. Parker[7], Watrous[8] and Ricotti[9] introduced quasi-newton methods. These optimization methods are very powerful to reduce the number of iterations, however, it does not always imply to save computational time since BP should train a lot of different patterns.

For the practical applications we have studied with much larger scale networks (image recognition) than usual bench-mark problems such as XOR, we have observed a similar phenomenon to local minima. Errors at the output layer hardly propagate to the inner layers, when only one η value in a network is used. We have found that it is able to overcome this problem by changing hidden layer's η value. In order to clarify the role of hidden layer's η, we determine some basic properties through numerical simulations. Then we present a practical adaptive method for the learning rate.

2. Basic Properties of Back Propagation

2.1 Brief Description of BP

A multi-layer network model is designed so that a node belonging to a layer sums up outputs from the previous layer and propagates this summation to the following layer after applying a sigmoid function. That is, the output O_j of a node j is given by

$$O_j = f(net_j) = \frac{1}{1+e^{-net_j}}, \quad net_j = \Sigma_i W_{ji}\cdot O_i + \theta_j \tag{1}$$

where O_i is an output value of the previous layer and θ_j is a bias value of the node j.
The basic idea of training is to adjust every weight to minimize the energy defined as the summation of squared output errors between actual output O_{pj} and desired output T_{pj}, corresponding to an input pattern p. The updating rule of weights is given by

$$\Delta W_{ji}(n+1) = \eta\cdot\delta_j\cdot O_i + \alpha\cdot\Delta W_{ji}(n) \tag{2}$$

$$\delta_j = (T_{pj} - O_{pj})\cdot f'(net_j) \text{ for the output layer}, \quad \delta_j = \Sigma_k \delta_k\cdot W_{kj}\cdot f'(net_j) \text{ for the hidden layers}$$

where η and α are parameters called learning rate and momentum, respectively, and n is the number of iterations. The most important aspect of BP is to be able to modify internal weights using the chain rule.

* Present affiliation: Central Research Laboratory, Hitachi, Ltd. Kokubunji, Tokyo 185, Japan.

2.2. Network Topology and Input Patterns

We have selected the exclusive-or ("XORn", where n is the number of nodes in the hidden layer, 2-n-1 networks) task for a fully-connected network, and "IMAGE" ("IMAGEnxn", where nxn nodes are arranged in two dimensional array, and each node in the hidden layer has 5x5 connections to the input layer, 10x10-nxn-4 networks) task that classifies simple geometrical figures, such as lines or blocks, for a partially-connected network, which is especially important for image recognition applications since the number of inputs, or pixels, becomes so large that it is impractical to connect to every node in a previous layer.

2.3 Relation between Parameters and the Number of Iterations

The relation between η and N (the number of iterations required for convergence) is shown in Figure 1 for the task XOR2. It should be noted that both axes are logarithmic. Several lines have been plotted in the figure for different values of α (0.0, 0.2, 0.4, 0.6, 0.8, 0.9, 0.95). In all cases, when η is small N is inversely proportional to η, and when η reaches a certain value N becomes a minimum. As η further increases, N increases very rapidly. This relation is quite natural if one accepts for the fact that BP is basically a gradient method in which the number of steps is inversely proportional to the size of each step. Besides, N is also inversely proportional to $1 / (1 - \alpha)$, we can conclude the relation

$$N \propto \frac{1-\alpha}{\eta}. \tag{3}$$

We have confirmed that this relation also satisfies other examples. It is therefore desirable that η takes as large a value as possible within the range in which the gradient approximation is correct, and α is as near to 1 as possible before instability occurs. As 0.9 is usually used for α, one can expect to get ten times faster convergence than with $\alpha = 0.0$.

2.4 η in Hidden Layers

We observed the number of iterations in which η_{output} was a constant and η_{hidden} was a variable. Figure 2 (XOR) and Figure 3 (IMAGE) show the results. In both cases the relation

$$N \propto \frac{1}{\sqrt{\eta_{hidden}}} \tag{4}$$

holds. Moreover, it is worth mentioning that the optimal η_{hidden} which gives a minimum number of iterations N depends on the network topology. That is, for XOR2, XOR8 and XOR32 (η_{output} = 5.0, 1.25 and 0.3), N reaches a minimum for η values near 0.3, 2.0 and 30.0 respectively, and for IMAGE2x2, IMAGE3x3, IMAGE5x5 and IMAGE10x10 (η_{output} = 2.5, 1.1, 0.4 and 0.1), the optimal η values are near 1.0, 10.0, 100.0 and 1000.0.

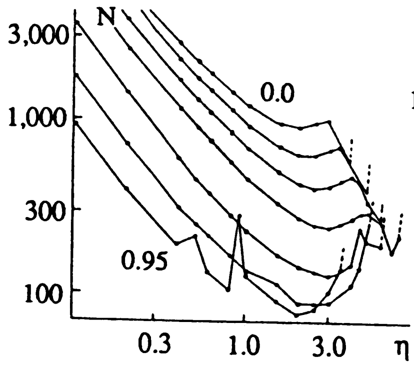


Figure 1. η–N relation, XOR2

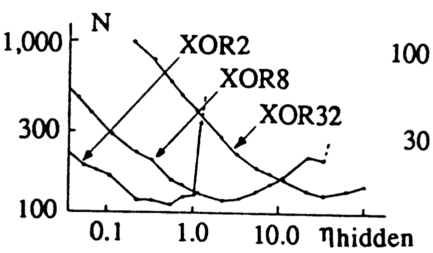


Figure 2. η_{hidden}–N relation, XOR

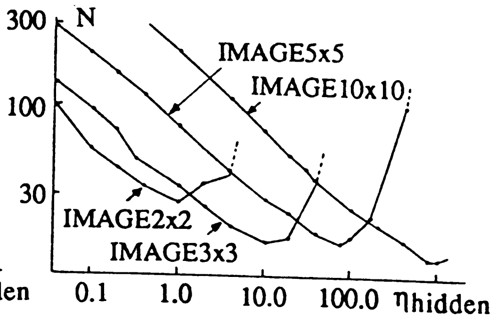


Figure 3. η_{hidden}–N relation, IMAGE

2.5 Normalized Parameter η

As we have seen, for IMAGE10x10, the hidden layer's η could be increased up to about 1000.0 without causing instability. The reason is that η of the basic BP equation (2) contains a different dimension than only that of a coefficient. Here we present a method which dissolves this dimension and normalizes the learning rate. The basic concept of this method is to normalize η depending on the number of node connections. When weights are updated during the convergence process, the difference change in the summation net_j of a node j becomes

$$\Delta net_j = \Sigma_i \, \Delta W_{ji} \cdot O_i = \Sigma_i \, \eta \cdot \delta_j \cdot O_i^2 \tag{5}$$

where $\alpha = 0$ is assumed for simplicity.

$$\eta = \frac{\Delta net_j}{(T_j - O_j) \cdot \Sigma_i \, O_i^2 \cdot f'(net_j)} \quad \text{for the output layer} \tag{6}$$

$$\eta = \frac{\Delta net_j}{\Sigma_k\ \delta_k \cdot W_{kj} \cdot \Sigma_i\ O_i^2 \cdot f\ '(net_j)}$$ for the hidden layers.

Equation (6) clearly shows that the value of η should be changed depending on the number of nodes connected to node j. In addition to this summation Σ_i, another summation Σ_k should be taken into account.

We now define the connectivity,

$$\chi = \frac{1}{\Sigma_k \frac{1}{M_k} \cdot I}, \text{ or } \chi = \frac{M}{K \cdot I} \text{ (when } M_k \text{ is a constant M).} \quad (7)$$

where I is the number of connections between node j and its preceding layer, K is the number of connections between node j and its successive layer and M_k is the number of connections between node k in its successive layer and the current layer. That is, comparing (7) with (6), K and I correspond to the summations Σ_k and Σ_i, respectively, and a weight W_{kj} can be thought to have a value in inverse proportion to M_k since if the number of connections is large, the corresponding weight should be small. Also, when the node belongs to the output layer, we can suppose that M=K=1, since each output has a one-to-one correspondence to the desired output. We can thereby rewrite η using the connectivity (7) as $\eta = \eta_0 \cdot \chi$, where η_0 is called the normalized η. Using this connectivity, we could rewrite the X-axis of Figure 2 and 3 as Figure 4 and 5, where the optimum range is almost same, regardless of the number of hidden nodes.

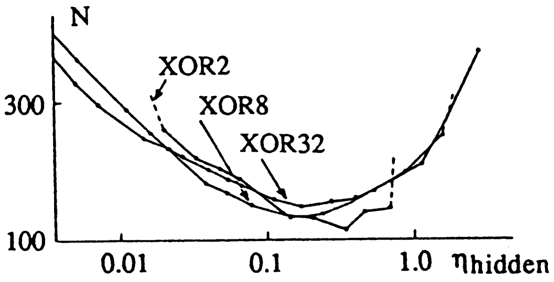


Figure 4. normalized η_{hidden}–N relation, XOR

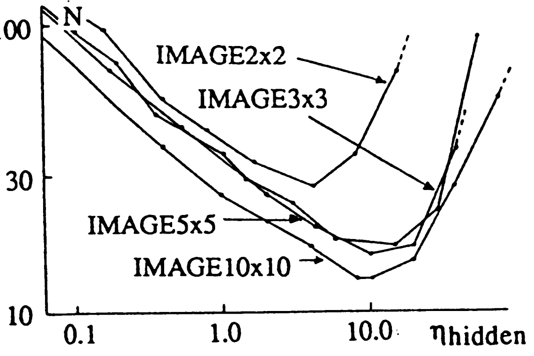


Figure 5. normalized η_{hidden}–N relation, IMAGE

3. Adaptation Method

We described how the normalized η could be obtained. However, the remaining problem is how η can be optimally adapted. This chapter describes an adaptation method that adjusts the learning rate, depending on characteristics of tasks, through a convergence process.

3.1 Algorithm

The basic idea of this adaptation method is to make η as large as possible within the region guaranteed by a gradient approximation. The magnitude of Δnet_j in equation (5) determines whether or not the gradient approximation is correct. We use Δnet_j, not ΔO_j, so that it prevents too much deviation in the saturated region of the sigmoid function. If the absolute value of Δnet_j, $|\Delta net_j|$, is smaller than a certain value, a step is regarded as correct and η is increased. Conversely if $|\Delta net_j|$ is greater than the value, a step is regarded as incorrect and η is decreased. In practice, we adopted a method in which the maximum $|\Delta net_j|$ within a layer determines the correctness, since every node in a layer has the same connectivity. This method is summarized as follows.

1. Forward Propagation: get outputs.
2. Backward Propagation: update all weights.
3. Forward Propagation: check the maximum deviation of $|\Delta net_j|$ and determine κ at each layer,
   If $|\Delta net_j|_{max} < \lambda$ then $\kappa = \kappa_1$, else $\kappa = \kappa_2$.

For instance, let us assume a network with one hidden layer. If the coefficient κ is κ_1 at the output layer and κ_2 at the hidden layer, η of the next iteration will be $\eta_0(n+1) = \eta_0(n) \cdot \kappa_1 \cdot \kappa_2$. We used $\kappa_1 = 1.1$, $\kappa_2 = 0.9$. It is reasonable to adapt η of each layer independently. However, we did not use this independent way but the coordinative way, since we had observed that the independent way tended to cause η for the hidden layers to become unstable.

3.2 Simulation Results

Figure 6 shows the behavior of η in terms of the iteration number in the case an XOR2 network where the iterations started with the initial value $\eta_0(0) = 0.001$ and $\alpha = 0.9$. The convergence process was terminated after 133 iterations with $\eta_0 = 5.6$. The behavior, in general, is such that η becomes large during the first several iterations, then decreases and remains constant ($\eta_0 = 1.5$) until after about 90 iterations, followed by two peaks around iteration number 100 and 110. This peak can be explained by the fact that the gradient approximation is still correct even if η is very large, because patterns other than pattern "10" begin to produce proper output. However, the larger the error pattern "10" begins to produce, the more η is decreased and thereby stabilized. Finally, four pattern begin to give proper outputs and η increases, and then the convergence process terminates.

This adaptation method requires three parameters, κ_1, κ_2 and λ. The effect of κ_1 and κ_2 is expected to be small, since η is stablized quickly. However, the determination level λ is crucial, since if too large a value is selected, η also becomes too large, although the gradient approximation is already incorrect. On the other hand, if too small value of λ is selected, η seldom becomes large. The influence of λ is shown in Figures 7, where XOR was used with $\alpha = 0.9$. Again, we observe the relation that N is inversely proportional to λ. The momentum effect, indicated by equation (3), is also observed. As a result, it can be said that the essence of this adaptation method is that the vague parameter η of BP is substituted by the well-characterized parameter λ.

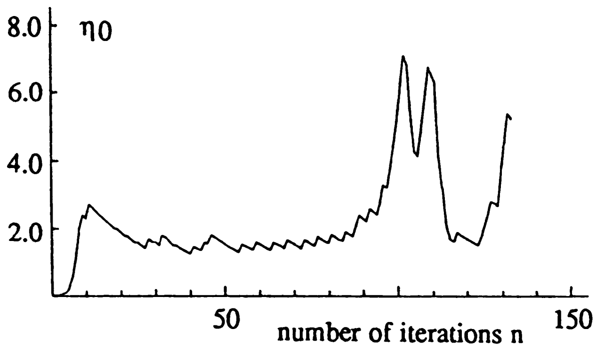


Figure 6. behavior of adaptive η, XOR

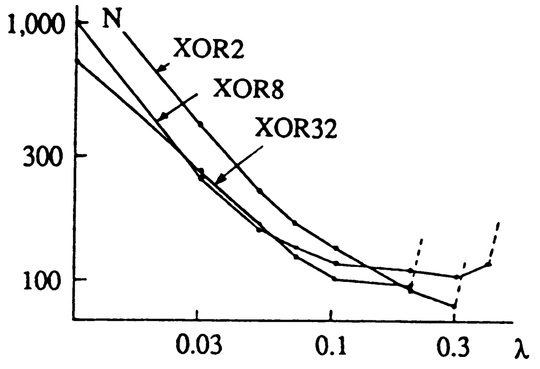


Figure 7. λ–N relation, XOR

4. Conclusion

We have clarified some basic properties of BP corresponding to convergence and derived an effective adaptation method based on the most primitive formulations. Through computer simulations, we have observed two experimental facts.

- The number of iterations N is proportional to $(1-\alpha) / \eta$ within the region in which a stable convergence is obtained, and in the case of a network with one hidden layer, N is proportional to $1 / \sqrt{\eta_{hidden}}$.
- η can be normalized in terms of connectivity determined by the connections between a layer and not only its preceding layer but also its successive layer. This is a remarkable characteristic of BP where errors at output layer are propagating layer by layer.

Finally, we have shown that the essence of this adaptation method is that the vague parameter η is substituted by a well-characterized parameter λ.

References

[1] Rumelhart,D.E., Hinton,G.E., Williams,R.J., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*, Chapter 8, The MIT Press, Cambridge, Mass, 1986.
[2] Sejnowski, T. J., Rosenberg, C. R, "Parallel Networks that Learn to Pronounce English Text," *Complex Systems* 1, pp.145-168, 1987.
[3] Dahl, E. D., "Accelerated learning using the Generalized Delta Rule," *ICNN'87*, pp.II523-530, 1987.
[4] Vogl, T. P., Mangis, J. K., Rigler, A. K., Zink, W. T., Alkon, D.L., "Accelerating the Convergence of the Back-Propagation Method," *Biological Cybernetics*, 59, pp.257-263, 1988.
[5] Jacobs, R. A.,"Increased Rates of Convergence Through Learning Rate Adaptation," *Neural Networks*, Vol.1, pp.295-307, 1988.
[6] Markram-Ebeid, S., Sirat, J.A., Viala, J.R.,"A Rationalized Error Back-Propagation Learning Algorithm," *IJCNN'89*, pp.II373-380, 1989.
[7] Parker, D. B.,"Optimal Algorithms for Adaptive Networks: Second Order Back Propagation, Second Order Direct Propagation, and Second Order Hebbian Learning," *ICNN'87*, pp.II593-600, 1987.
[8] Watrous, R. L.,"Learning Algorithms for Connectionist Networks: Applied Gradient Methods of Nonlinear Optimization," *ICNN'87*, pp.II619-627, 1987.
[9] Ricotti, L. P., Ragazzini, S., Martineli, G.,"Learning of Word Stress in sub-optimal Second Order Back Propagation, Neural Network," *ICNN'88*, pp.I355-361, 1988.

INTRODUCING EFFICIENT SECOND ORDER EFFECTS INTO BACK PROPAGATION LEARNING

D. M. Himmelblau

Department of Chemical Engineering
The University of Texas, Austin, Texas 78712

1. Introduction

An extensive literature within the area of artificial neural networks has already developed on the topic of learning via the backpropagation algorithm [1,2]. Within this general subject a few investigators [3,4,5,6,7,8,9] have examined how to enhance the learning rate by adding second order effects, i.e., curvature, to the first order (gradient) optimization procedure. However, in none of the cited references is it precisely clear how the introduction of second order effects influences the calculation of the δ's in the generalized delta rule, and how the different choices for approximates to the second order derivatives might be calculated. Here we show specifically how the backpropagation algorithm can be modified by taking into account second order derivatives, and list two procedures, one for serial and the other for parallel computation.

2. How the Second Order Effects are Introduced

In this section we sketch how the second order effects evolve for the adjustment of the weights on the connections of the artificial neural network. Figure 1 shows a typical three layer artificial neural network comprised

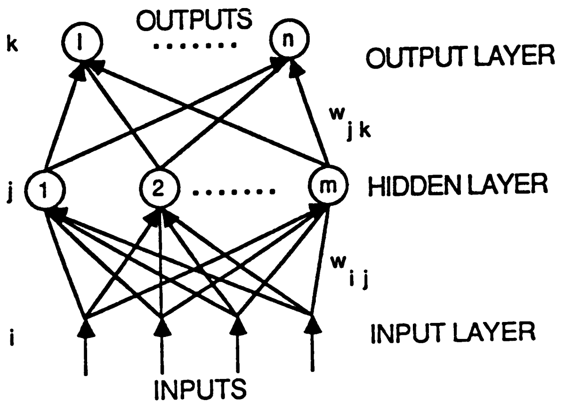


Figure 1. A feed forward artificial neural network

of one input layer, one hidden layer, and one output layer together with some of the notation to be used.

Let $E_{pk} = (t_{pk} - y_{pk})$ be the error for pattern p in an output node calculated by taking the difference between the target value t_{pk} and the output of the node y_{pk} in the output layer. We want to minimize the sum of the squares of the errors

$$F = \frac{1}{2}\sum_{p}\sum_{k} E_{Rk}^{2} = \frac{1}{2}\sum_{p}\sum_{k}\left(t_{pk} - y_{pk}\right)^{2} \quad (1)$$

In what follows we will suppress the subscript p to simplify the notation and consider just one pattern presentation.

Minimization of F in general is composed of a two phase interative calculation:

1. Calculation a search direction.
2. Calculate a step length (move some distance in that search direction before returning to phase 1).

Application of unconstrained Newton's method (as described in most books on solving nonlinear equations and optimization) gives the classical relation for updating the weights on the connections in the output layer of the net

$$w_k^{(k+1)} - w^k = \Delta w_k^{(k)} = -\left[H\left(w_k^{(k)}\right)\right]^{-1} \nabla F\left(w_k^{(k)}\right) \tag{2}$$

where the superscript (k) designates the iteration number, and (suppressing the (k) to keep the notation simple) $w_k = \left[w_{1k}\ w_{2k} \ldots w_{nk}\ w_{\theta k}\right]^T$ is the vector of weights associated with the connections entering node k in the output layer plus the weight associated with the threshold; $\mathbf{H}\left(w_k\right)$ is the Hessian matrix (matrix of second partial derivatives of F with respect to the w_{ik}) of $F(w_k)$; and $\nabla F\left(w_k\right) = \left[\partial F/\partial w_{1k}\ \partial F/\partial w_{2k} \ldots \partial F/\partial w_{nk}\ \partial F/\partial w_{\theta k}\right]^T$ is the gradient of $F(w_k)$. Alternately, instead of Equation (2), a set of linear equations could be solved for Δw_k if the inverse Hessian matrix is not used

$$\nabla F\left(w_k\right) = -\hat{\mathbf{H}}\left(w_k\right)\Delta w_k \tag{2a}$$

A related relationships is sometimes used termed the damped Newton method in which the following set of equations is solved for Δw.

$$\Delta F\left(w_k\right) = -\left[\hat{\mathbf{H}}\left(w_k\right) + \alpha \mathbf{I}\right]\Delta w_k \tag{2b}$$

or

$$\Delta w_k = -\left[\hat{\mathbf{H}}\left(w_k\right) + \alpha \mathbf{I}\right]^{-1} \nabla F\left(w_k\right) \tag{2c}$$

where α is a positive scalar and $\mathbf{I}$ is the identity matrix; α is adjusted by comparing the actual and predicted value of F. From the viewpoint of Equation (2b) or (2c), Δw_k is a compromise between the Newton direction and the steepest descent direction as adjusted by the weight α.

Newton's relation gives both the search direction and the step length, the later being unity ($\lambda = 1$), but other values of λ can be chosen, such as minimizing in a search direction or using a trust region, as explained in the literature on optimization. In equation (2) or (2a) a scalar λ would then be inserted before $\mathbf{H}^{-1}$ or $\mathbf{H}$. Also, we will not actually use the Hessian matrix itself, but only an approximate thereof obtained from elements in the gradient or values of the elements of Δw, so that we would replace the symbol $\mathbf{H}^{-1}$ by $\hat{\mathbf{H}}^{-1}$, or $\mathbf{H}$ by $\hat{\mathbf{H}}$.

For one interation for one pattern p for one node k in the output layer, from Equation (2) or (2a) the correction for a weight is

$$\Delta w_{jk} = -\lambda \sum_j \hat{h}_{rsk} \frac{\partial F\left(w_k\right)}{\partial w_{jk}} \tag{3}$$

where $\hat{h}_{rsk}$ is the proper element of $\hat{\mathbf{H}}^{-1}\left(w_k\right)$ associated with the output layer. To compare Equation (3) with the usual backpropagation development, let $\hat{\mathbf{H}}^{-1} = \mathbf{I}$ so that $h_{jlk} = 1$, and let $\lambda = \eta$. Curvature is introduced by the weighted sum of the gradient elements. If as usual $\partial F/\partial w_{ij} = (\partial F/\partial u_k)(\partial u_k/\partial w_{ij})$ and $u_k = \sum_j w_{jk} y_j + w_{\theta j}$, and we define $\delta k = \partial F/\partial u_k$, then

$$\Delta w_{jk} = -\lambda \sum_j \hat{h}_{rsk} \delta_k y_j \tag{4}$$

The delta for one node in the output layer would be calculated as

$$\delta_k = \left(\frac{\partial F\left(w_k\right)}{\partial y_k}\right)\left(\frac{\partial y_k}{\partial u_k}\right) = \left(t_k - y_k\right) f'\left(u_k\right) \tag{5}$$

where $\partial y_k/\partial u_k = f'(u_k)$ because y_k is a function of the node input u_k: $y_k = f'(u_k)$. For a **hidden layer**

$$\frac{\partial F\left(w_k\right)}{\partial w_{ij}} = \left(\frac{\partial F\left(w_k\right)}{\partial u_j}\right)\left(\frac{\partial u_j}{\partial w_{ij}}\right) = -\delta_j y_j$$

and by use of the chain rule several texts show that

$$\delta_j = f_j'\left(u_j\right) \sum_k \delta_k w_{jk} \tag{6}$$

so that

$$\Delta w_{ij} = \lambda \sum_{i} \hat{h}_{rsi} \, \delta_j \, y_i \tag{7}$$

If the input-output expression for a node is the sigmoidal function $y_j = 1/(1 + e^{-u_j})$, then the deltas are calculated as:

output layer: $\delta_k = (t_k - y_k)\, y_k\, (1 - y_k)$
hidden layer: $y_j\,(1 - y_j) \sum_{k} \delta_k w_{jk}$

3. Evaluate of the Elements in $\hat{H}$ or $\hat{H}^{-1}$

The evaluation of the elements of $\hat{H}^{-1}(w)$, or $\hat{H}(w)$, for each layer can be accomplished in innumerable ways as explained in the literature on nonlinear programming. In particular the BFGS secant update relations are generally favored (the argument w_k or w_j is suppressed)

$$\left[\Delta\hat{H}^{(k)}\right]^{-1} = \left[\hat{H}^{(k+1)}\right]^{-1} - \left[\hat{H}^{(k)}\right]^{-1} = \frac{\left[\Delta w^{(k)} - \left(H^{(k)}\right)^{-1}\Delta g^{(k)}\right]\left[\Delta w^{(k)}\right]^{T} + \left[\Delta w^{(k)}\right]\left[\Delta w^{(k)} - \left(H^{(k)}\right)^{-1}\Delta g^{(k)}\right]^{T}}{\left(\Delta g^{(k)}\right)^{T}\Delta g^{(k)}}$$
$$- \frac{\left[\Delta w^{(k)} - \left(H^{(k)}\right)^{-1}\Delta g^{(k)}\right]^{T}\Delta g^{(k)}\,\Delta w^{(k)}\left(\Delta w^{(k)}\right)^{T}}{\left[\left(\Delta g^{(k)}\right)^{T}\Delta w^{(k)}\right]\left[\left(\Delta g^{(k)}\right)^{T}\Delta w^{(k)}\right]} \tag{8}$$

$$\Delta\hat{H}^{(k)} = \hat{H}^{(k+1)} - \hat{H}^{(k)} = \frac{\Delta g^{(k)}\left(\Delta g^{(k)}\right)^{T}}{\left(\Delta g^{(k)}\right)^{T}\Delta w^{(k)}} - \frac{\hat{H}^{(k)}\Delta w^{(k)}\left(\Delta w^{(k)}\right)^{T}\hat{H}^{(k)}}{\left(\Delta w^{(k)}\right)^{T}\hat{H}^{(k)}\Delta w^{(k)}} \tag{8a}$$

where $\Delta g^{(k)} = \Delta F(w^{(k+1)}) - \Delta F(w^{(k)})$. The elements in all the vectors and matrices in Equations (8) and (8a) can be calculated from known values at stage (k).

The suggested procedure for serial calculations is

- start with the current values of $w^{(k)}$, $F\left(w^{(k)}\right)$, $\left[\hat{H}^{(k)}\left(w^{(k)}\right)\right]^{-1}$, $H^{(0)} = I$
- calculate $\Delta w^{(k)}$ by the equations above
- evaluate $\Delta F(w^{(k)})$ and calculate $\Delta g^{(k)}$
- terminate if tolerance is achieved, and if not
- update $\left[\hat{H}^{(k)}\right]^{-1}$ by Eq. (8) to get $[H^{(k+1)}]^{-1}$

For parallel calculation there are four variants of the updating relations for H or H^{-1} as indicated in Table 1 [10, 11]

Table 1 Updating H or H^{-1} for Parallel Calculation

	Matrix Stored Unfactored	Matrix Stored Factored
Update H	$H^{(k+1)} = H^{(k)}$ + rank-two update $H^{(k)}$ stored	$J^{(k+1)} = L^{(k)}$ + rank-one update $L^{(k)}$ (lower triangular) stored $H^{(k)} = L^{(k)}(L^{(k)})^T$
Update H^{-1}	$[H^{k+1}]^{-1} = [H^{(k)}]^{-1}$ + rank-two update $[H^{(k)}]^{-1}$ stored	$M^{(k+1)} = M^{(k)}$ + rank-one update $M^{(k)}$ stored $[H^{(k)}]^{-1} = M^{(k)}(M^{(k)})^T$

The four methods yield the same results in exact arithmetic, and although they differ in the number of arithmetic operations involved, in practice none appears to be superior to the others. By parallelizing the linear algebraic computations and or performing parallel evaluations of the objective function concurrently as described in references [10] and [11], an increase in the speed of calculation by a factor of almost 20 (at the expense of additional processor) can be achieved.

Acknowledgement

This work was supported by E. I. Dupont.

References

1. Werbos, P. J., "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences", Ph.D. Thesis, Harvard University, Cambridge, 1974.
2. Rumelhart, D. E., G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation", Chapt. 8 in *Parallel Distributed Processing, Vol. 1*, ed. D.E. Rumelhart, J. L. McClelland, et. al., MIT Press, Cambridge MA, 1986.
3. Hoskins, J., "Speeding Up Artificial Neural Networks in the Real World", paper presented at the Washington, D. C. ICNN Meeting, June, 1989.
4. Kollias, S. and D. Anastassious, "Adaptive Training of Multilayer Neural Networks Using a Least Squares Estimation Technique", *Proceed. IEEE Conf. on Neural Networks, San Diego, July 24-27*, pp. I 383-390, 1988.
5. Kung, S. Y. and J. N. Hwang, "An Algebraic Projection Projection Analysis for Optimal Hidden Units Size and Learning Rates in Back-Propagation Learning", *Proceed. IEEE Conf. on Neural Networks, San Diego, July 24-27*, pp. I 363-370, 1988.
6. Ricotti, L. P., S. Ragazzini, and G. Martinelli, "Learning Word Stress in a Sub-Optimal Second Order Back-Propagation Neural Network", *Proceed. IEEE Conf. on Neural Networks, San Diego, July 24-27*, pp. I 355-361, 1988.
7. Watrous, R. L., "Learning Algorithms for Connections and Networks: Applied Gradient Methods of Nonlinear Optimization", *Proceed. IEEE Conf. on Neural Networks, San Diego, July 24-27*, pp. II 619-627, 1987.
8. White, H., "Some Asymptotic Results for Back-Propagation", *Proceed. IEEE Conf. on Neural Networks, San Diego, July 24-27*, pp. III 261-266, 1987.
9. Parker, D. B., "Optimal Algorithm for Adaptive Networks: Second Order Back Propagation, Second Order Direct Propagation, and Second Order Hebbian Learning", *Proceed. IEEE Conf. on Neural Networks, San Diego, July 24-27*, pp. II 593-600, 1987.
10. Byrd, R. H., R. G. Schnabel, and G. A. Schultz, "Using Parallel Function Evaluations to Improve Hessian Approximation for Unconstrained Optimization", *Annals Oper. Res., 14*, 167-193, 1988.
11. Schnabel, R. B., "Concurrent Function Evaluations in Local and Global Optimization", *Computer Methods Applied-Mech. Engr., 64*, 537-552, 1987.

A NOVEL, ONE-STEP, GEOMETRICAL, SUPERVISED LEARNING SCHEME

Chia-Lun J. Hu
Electrical Engineering Department
Southern Illinois University
Carbondale, IL 62901

A B S T R A C T

When the input-output digital mapping relations are given for a two-layer Hopfield net which has hard-limited signal functions, the connection matrix can be solved in one step in terms of the given mapping relations. This solution is obtained by means of a novel discrete geometrical method applied to the N-space which is the state space of the neural system containing N neurons. The condition that solution exists for a given mapping is discussed. Numerical examples verifying the theoretical predictions are given. The geometrical method used here is simple, practical in designs, and it provides more perceptions to the physical insides of the learning mechanism.

I. INTRODUCTION

Supervised learning of a neural net generally requires an iteration process that minimizes the difference between the true outputs and the targeted outputs. The most common mathematical tool people used to achieve this minimization is the gradient descend method. Rumelhart, Hinton, and Williams [1] derived the most well known delta learning rule based on this gradient descend iteration approach. Widrow, al et, [2-4] derived another very efficient learning rule that forces W_{ij} to change along the most efficient path for reaching the equilibrium state. This rule is commonly known as the Widrow-Hoff rule. Barto, et al, [5-7], Williams [8], and Widrow [9] have also derived separately another class of very similar learning rules that Williams called it the reinforcement learning, while Barto, Widrow called it the reward-penalty algorithm. In place of the back-propagating errors, this scheme uses a reinforcement signal, derived from the degree of eliminating the least square errors. The scheme utilizes this reinforcement signal to award or to penalize the weighting coefficients to achieve the learning purpose.

While most of the learning studies are concentrated at the derivation and the performance of various learning rules, Widrow has addressed a very important point in the learning process. That is, the learning of new information should not disturb the old informations learned [2]. In the Widrow-Hoff learning rule, this disturbance is minimized because of the smallest ΔW_{ij} this rule provides for reaching the learning equilibrium. Also in [2], another important concept is addressed -- the use of the state-space geometrical method to analyze the change of mappings in the learning process. Geometrical method is also used very intriguingly by Mark, Oh and Atlas [10] in the study of a novel associative recall convergence property. In the present article, we also adopt the geometrical approach to analyze a novel learning process. This method appears to be very simple, very perceptive, and very practical in designs. In the following, section II will be devoted to the formulation of this supervised learning problem. In section III, we will solve the problem, and in IV, the general properties will be summerized.

II. A ONE-STEP SUPERVISED LEARNING MECHANISM

For a Hopfield net with all d.c. components separated, the controlling equation between the input pattern and the output pattern is the following:

$$R_n = \mathrm{Sgn}\left[\sum_{j=1}^{N} a_{nj} U_j\right], \qquad n=1 \text{ to } N \qquad (1)$$

where U_j is the j-th component of the input pattern, and R_n is the n-th component of the output pattern. a_{nj} is the connection matrix element. Sgn is the sign function which gives +1 or -1 value depending on the argument being positive or negative. N is the number of neurons. Now suppose there are M input-output pairs of patterns to be learned. Then the total error of the unlearned system is:

$$E = \sum_{m=1}^{M} \sum_{n=1}^{N} \left(V_{mn} - R_{mn} \right)^2 \qquad (2)$$

where R_{mn} and V_{mn} are the n-th compoents of the true output and the targeted output in the m-th mapping pair. Substituting (1) into (2), and forcing E to vanish by adjusting a_{ij}, we have the following equation.

$$V_{mn} = \mathrm{Sgn}\left[\sum_{j=1}^{N} a_{nj} U_{mj}\right], \quad m=1 \text{ to } M. \qquad (3)$$

U_{mj} and V_{mn} here are the components of the targeted input output mapping pair defined in the following.

Input pattern: $\{U_{mj}$: m=1 to M, j=1 to N, U_{mj}=+1 or -1.$\}$ (4-1)

output pattern: $\{V_{mn}$: m=1 to M, n=1 to N, V_{mn}=+1 or -1.$\}$ (4-2)

Since U_{mj}, V_{mn} take only +1 or -1 values, if we multiply -1 to those equations in (3) which have V_{mn}=-1, we will obtain the following equation.

$$\sum_{j=1}^{N} W_{mnj} a_{nj} > 0, \quad m=1 \text{ to } M \text{ and } n \text{ is a fixed number.} \qquad (5)$$

In (5), $W_{mnj}=V_{mn}U_{nj}=+1$ or -1 (6)

as determined by the given mapping in (4). We see now that to learn the required mapping relations in (4) is just equivalent to finding the solution of a_{nj} from the simultaneous inequality equations in (5).

III. METHOD OF SOLUTION AND NUMERICAL EXAMPLES

We see that the left hand side of (5) is the inner product of two N-dimensional vectors of fixed mn indices. One is a given vector $\bar{W}_{mn}$, the other is the unknown vector $\bar{a}_n$ which is the n-th row of the connection matrix $[a_{nj}]$. For a fixed n, we have M inequalities to solve in (5). From an N-dimensional geometrical point of view, solving (5) is the same as finding a vector $\bar{a}_n$ such that when it is "dotted" to <u>any</u> of the given vectors $\bar{W}_{mn}$ (m=1 to M), the result is always greater than zero. Consequently, we see now that

the condition that the solution $\bar{a}_n$ exists is that all W-vectors form a "convex cone" in the N-space. If this is met, then $\bar{a}_n$ is just any vector falling within that cone. .... (7)

The general solution of $\bar{a}_n$ is ANY positive, linear combination of all the W-vectors, or,

$$\bar{a}_n=\sum_{(m=1 \text{ to } M)} b_m\bar{W}_{mn}, \quad \text{with } \{b_m\}>0 \text{ and } n=1 \text{ to } N.... \quad (8)$$

When the targeted mapping pairs in (4) are given, $\bar{W}_{mn}$ can be calculated from (6). Then $[a_{nj}]$ can be solved from (8).

Following this method, five numerical examples are studied with five different mapping relations given. The first four give us legitimate solutions which lead us to some interesting properties as summerized in the following. The last example does not give us any legitimate solution because the W-vectors computed from the given mapping relations DO NOT form a CONVEX CONE as stated in (7).

IV. CONCLUSION

The summery of the general properties derived from this geometrical one-step learning process is the following.

1. Learning more information will not destroy old informations already learned.

2. Maximum number of informations (or mapping pairs) can be learned is 2^{N-1} where N is the number of neurons.

3. If the given mapping violates the condition stated in (7), that mapping is illegal and can not be learned.

4. Learning can be saturated. The connection matrix in this case is just equal to the unit matrix.

REFERENCES

[1] Rumelhart, D.E., Hinton, G. E., Williams, R. J., "Learning internal representations by error propagation," Parallel distributed processing: Explorations in the microstructures of cognition, Voll, D.E. Rumelhart, J. L. McClelland (eds.), MIT Press, 318-362, 1986.

[2] Widrow, B., Winter, R., Baxter, R., "Learning phenomena in layered neural networks," Proceeding of IEEE First International Conference on Neural Networks, San Diego, CA, Vol. II, 411-429, June 21-24, 1987.

[3] Widrow, B., Hoff Jr., M., "Adaptive switching circuits," IRE WESCON Con. Rec., pt. 4, 96-104, 1960

[4] Widrow, B., Sterns, S.D., Adaptive Signal Processing, Prentice Hall, Englewood Cliffs, NJ, 1985.

[5] Barto, A.G., Jordan, M.I., "Gradient Following Without Back-Propagation in Layered Networks," Proceeding of IEEE First International Conference on Neural Networks, San Diego, CA, Vol. II, 629-636, June 21-24, 1987.

[6] Barto, A.G., "Game-theoretic cooperativity in networks of self-interested units," in Neural Network for Computing, edited by J. S. Denker, American Institute of Physics, New York, 1986.

[7] Barto, A.G., Anandan, P., "Pattern recognizing stochastic learning automata," IEEE Trans. Systems, Man, and Cybernetics, 15: 360-375, 1985.
[8] Williams, R.J., "A class of grdient-estimating algorithms for reinforcement learning in neural networks," Proceeding of IEEE First International Conference on Neural Networks, San Diego, CA, Vol. II, 601-608, June 21-24, 1987.
[9] Widrow, B., Sterns, S.D., Adaptive Signal Processing, Prentice Hall, Englewood Cliffs, NJ, 1985.

[10] Marks II, R.J., Oh,S., Atlas, L.E., "Alternating projection neural network," IEEE Trans. Circuits and Systems, 36-6, 846-857, June 1989.

Speeding Up Back Propagation

Yoshio Izui[1] and Alex Pentland

Vision Science Group, The Media Laboratory
Massachusetts Institute of Technology
20 Ames St., Cambridge MA 02139

Abstract

We prove that the convergence speed of the back propagation learning algorithm is dependent on the sharpness D of the sigmoid function used to define the network's energy function. When using the simple gradient method for updating weights the time to convergence is $T_1 = C_1/D$, while for the momentum method the time to convergence is $T_2 = C_2/\sqrt{D}$ (note that normally $C_2 \ll C_1$). Thus a simple modification of existing code can markedly improve convergence speed.

1 Introduction

The use of back propagation in feed-forward neural networks has become very popular in experimental applications [1]. The slow convergence speed of back propagation, however, has proven to be a major obstacle in practical applications [2]. As a consequence researchers have investigated several methods of speed-up, including use of second-order update rules [3] and dynamic parameter adjustments [4]. Most of these methods have the disadvantage of being substantially more complex, and in addition often must be "tuned" to fit the particular application.

In this paper we prove that the convergence speed of the back propagation method depends directly on the sharpness of the sigmoid function used in defining the energy function. Thus a simple, general and yet surprisingly powerful method of speeding up the convergence is simply to increase the sensitivity of the sigmoid used to compute the energy function. In our simulations we have obtained up to eight-fold speedups of the momentum update rule in this manner.

2 Convergence Speed

2.1 Review of Back Propagation

Back propagation functions by minimizing an energy function E that involves a non-linear function $g(x)$ of the connection weights; $g(x)$ is sometimes called the *transfer function* between nodes. Normally $g(x)$ is the sigmoid function, $g(x) = 1/(1 + e^{-Dx})$ where D is the *sharpness* of the sigmoid function. In most treatments $D = 1$ and is therefore dropped from the sigmoid's definition.

[1]Current address is Industrial Systems Lab., Mitsubishi Electric Corp., 8-1-1, Tsukaguchi, Amagasaki, Hyogo 661 Japan

The energy function associated with a D-sharpened sigmoid function is,

$$E = \frac{1}{2}\sum_{p=1}^{P}\sum_{k=1}^{K}\left(g\left(D\sum_{j=1}^{J} W_{2kj} g\left(D\sum_{i=1}^{I} W_{1ji} V_i^p\right)\right) - d_k^p\right)^2 \tag{1}$$

where V_i^p is i-th component of the p-th training data, d_k^p is the k-th component of p-th desired output data, and W_{1ji} and W_{2kj} are weights between input and hidden, and hidden and output layers, respectively.

In the back propagation method this energy is minimized either by a simple gradient method,

$$\frac{dW_{nji}}{dt} = -\frac{\eta}{D}\frac{\partial E}{\partial W_{nji}} \tag{2}$$

where $n = 1, 2$ and η may be thought of as the "gain" of the system. To more closely model biological systems, as well as to minimize the effects of shallow local minima in E, one may alternatively use a momentum method,

$$\frac{d^2 W_{nji}}{dt^2} + (1-\alpha)\frac{dW_{nji}}{dt} = -\frac{\beta}{D}\frac{\partial E}{\partial W_{nji}} \tag{3}$$

where $(1-\alpha)$ may be thought of as the damping within the system and β as it's gain.

Note that the D in equation (2), and (3) only serves to cancel the D's appearing the the energy function, equation (1). Thus changing the sharpness D does *not* change the how system of weights will evolve over time — i.e., D does not affect the path in weight space — but rather only changes the *rate* at which they change.

2.2 The Gradient Descent Update Method

We start with the simple gradient descent method of updating the weights. First, let us define $\widetilde{W_{1ji}} = DW_{1ji}$, $\widetilde{W_{2kj}} = DW_{2kj}$, and $\widetilde{z_{1ji}} = \frac{d\widetilde{W_{1ji}}}{dt}$, $\widetilde{z_{2kj}} = \frac{d\widetilde{W_{2kj}}}{dt}$. The convergence time T_1 can be obtained by first re-writing equations (2) to obtain an expression for dt, and then by integrating dt:

$$T_1 = \int dt = \frac{1}{D}\int \frac{ds}{\sqrt{\sum_{j,i}\left(\eta\frac{\partial E}{\partial \widetilde{W_{1ji}}}\right)^2 + \sum_{k,j}\left(\eta\frac{\partial E}{\partial \widetilde{W_{2kj}}}\right)^2}} \tag{4}$$

where

$$ds = \sqrt{\sum_{j,i}(d\widetilde{W_{1ji}})^2 + \sum_{k,j}(d\widetilde{W_{2kj}})^2} \tag{5}$$

Once η and initial values are determined, then the integral part of equation (4) is a constant. Thus the convergence time T_1 is inversely proportional to D, the sharpness of sigmoid function.

Note, however, that these equations describe a continuous system whereas computer simulations employ a finite difference scheme that uses discrete time steps. Thus as D

becomes large at some point the validity of the finite difference approximation breaks down. The effect of this breakdown is, to a first approximation, to introduce a large amount of noise into the energy function. The value of D at which breakdown occurs is a function of the maximum weight velocities achieved in the particular network under consideration.

2.3 The Momentum Update Method

The more complex momentum update method may be similarly treated. The convergence time T_2 is

$$T_2 = \int dt = \int \frac{ds'}{\sqrt{A_{11} + A_{12} + A_{21} + A_{22}}} \tag{6}$$

where

$$ds' = \sqrt{\sum_{j,i} \left\{ \left(d\widetilde{z_{1ji}}\right)^2 + \left(d\widetilde{W_{1ji}}\right)^2 \right\} + \sum_{k,j} \left\{ \left(d\widetilde{z_{2kj}}\right)^2 + \left(d\widetilde{W_{2kj}}\right)^2 \right\}} \tag{7}$$

and for $n = 1, 2$,

$$\begin{aligned} A_{n1} &= D^2 \sum_{j,i} \left(\beta \frac{\partial E}{\partial \widetilde{W_{nji}}} + \frac{(1-\alpha)}{D} \frac{d\widetilde{W_{nji}}}{dt} \right)^2 & (8) \\ A_{n2} &= \sum_{j,i} \left(\frac{d\widetilde{W_{nji}}}{dt} \right)^2 & (9) \end{aligned}$$

If we assume that $\beta D \quad >> \quad (1-\alpha)$, i.e., that the gain of the system times the sensitivity of the sigmoid function is much larger than the amount of damping in the system, then the damping may be ignored to achieve the following approximation:

$$\frac{d\widetilde{z_{1ji}}}{d\widetilde{W_{1ji}}} = -(1-\alpha) - \beta D \frac{1}{\widetilde{z_{1ji}}} \frac{\partial E}{\partial \widetilde{W_{1ji}}} \approx -\beta D \frac{1}{\widetilde{z_{1ji}}} \frac{\partial E}{\partial \widetilde{W_{1ji}}} \tag{10}$$

The solution of (10) is

$$\widetilde{z_{1ji}} = \sqrt{2D\beta} \left(C_{1ji} - \int \frac{\partial E}{\partial \widetilde{W_{1ji}}} d\widetilde{W_{1ji}} \right)^{\frac{1}{2}} \tag{11}$$

where C_{1ji} be a constant. Equation (6) may then be simplified as follows:

$$T_2 \approx \frac{1}{\sqrt{2D\beta}} \int \frac{ds}{\sqrt{\sum_{j,i} \left(C_{1ji} - \int \frac{\partial E}{\partial \widetilde{W_{1ji}}} d\widetilde{W_{1ji}} \right) + \sum_{k,j} \left(C_{2kj} - \int \frac{\partial E}{\partial \widetilde{W_{2kj}}} d\widetilde{W_{2kj}} \right)}} \tag{12}$$

where ds is as in equation (5).

We may simplify equation (12) still further by noting that

$$E = \int \sum_{j,i} \frac{\partial E}{\partial \widetilde{W_{1ji}}} d\widetilde{W_{1ji}} + \int \sum_{k,j} \frac{\partial E}{\partial \widetilde{W_{2kj}}} d\widetilde{W_{2kj}} \tag{13}$$

We first use this relation to obtain $E_{initial}$, the energy at the initial state,

$$E_{initial} = \sum_{j,i} C_{1ji} + \sum_{k,j} C_{2kj} \tag{14}$$

assuming the standard initial values $\widetilde{z_{1ji}} = \widetilde{z_{2kj}} = 0$. We can then use equations (14) and (13) to reduce our expression for T_2 to the following:

$$T_2 = \frac{1}{\sqrt{D}} \int_{initial}^{convergence} \frac{ds}{\sqrt{2\beta(E_{initial} - E)}} \tag{15}$$

Thus for the moment method the convergence time is proportional to the inverse of square root D, the sharpness of sigmoid function.

3 Summary

We have obtained closed form solutions for the convergence time of the back propagation method using either the gradient or momentum update methods, assuming that a sigmoid function is employed as the transfer function between nodes.

The main results are that T, the time to convergence, is equal to C_1/D when using the gradient method and $C_2/\sqrt{D}$ when using the momentum method where D is the sigmoid function's degree of sharpness. In our simulations we have successfully employed values of up to $D = 100$, which has produced an eight-fold speedup when using the momentum update rule. Note that despite the fact that D has a greater effect upon the gradient rule than upon the momentum rule, C_2 is so much smaller than C_1 that even for large values of D the momentum rule still converges faster than the gradient rule.

The primary consequence of this result is that a simple modification of most existing back propagation code — increasing the sharpness D of the sigmoid function — can yield a substantial increase in convergence efficiency. Care must be taken, however, to choose D small enough that the finite difference calculations used to approximate the change in system energy E still accurately approximate the underlying continuous energy function.

References

[1] Rumelhart, McClelland and the PDP Research Group. (1986). Parallel Distributed Processing Vol.I. MIT Press.

[2] Robert Hecht-Nielsen. (1989). Theory of the Backpropagation Neural Network. *Proc. of the IJCNN.* **Washington D.C.,** I593-I605.

[3] L. P. Ricotti, S. Ragazzini, G.M Artinelli. (1988) Learning of word stress in a sub-optimal second order back-propagation neural network. *Proc. of the IEEE Second Annual ICNN.* **San Diego,** I355-I361.

[4] R.A. Jacobs. (1988). Increased Rates of Convergence Through Learning Rate Adaptation. *Neural Networks*, **1**, 295-307.

Explanation-Based Learning and Relevance

Bruce F. Katz
The Beckman Institute for Advanced Science and Technology
University of Illinois; Urbana, IL 61801

INTRODUCTION

The problem of relevance is crucial to tractable learning procedures. Features of high importance are often hidden in a sea of largely irrelevant distracters. To consider every feature is costly, and moreover, a large number of training examples may be required before it is realized that a particular feature has no bearing on the a set of outcomes. One solution is to have the teacher present the learner with a set of hand-picked events, such that the positive examples differ minimally from the negative examples (Winston, 1970). Needless to say, such a procedure is not always possible in unsupervised or loosely supervised environments.

Recently, non-connectionist learning theory has suggested an alternative approach to this problem, which it has termed Explanation-Based Learning (DeJong & Mooney, 1986). In the classical formulation of the EBL problem (Mitchell, Kellar, & Kedar-Cabelli , 1986), one is given a set of domain rules, a training example, and a goal that can be inferred by the application of the domain knowledge to the example. An explanation structure is then constructed, with the input features at the leaves of this tree, and the goal node at the top. This structure may then be generalized using goal regression or other related techniques (Mooney & Bennet, 1987). The resulting structure is then be "flattened", so that a new rule is formed with the left hand side being the generalized example, and the right hand side the original goal. If the left-hand side is readily observable, then one will have a quick and easy way of predicting the goal concept given the appropriate inputs, without the need to produce what may be an extensive inference chain. To take a simple example, let us assume one has access to two rules: a) that all professors are absent-minded, and b), that all absent-minded people misplace things. Suppose one sees Professor X misplacing his glasses. After forming an explanation of this event with the help of a) and b), one emerges in the end with the general rule that professors will tend to misplace things. One may question the role of the example in the above, since, from a strictly logical point of view, it is unnecessary. The standard response to this objection (Mitchell, Kellar, & Kedar-Cabelli, 1986) is that the example indicates which type of knowledge it may be profitable to chunk; the full deductive closure of one's current knowledge is not readily computable given spatial and temporal limitations.

EBL, then, differs primarily with SBL in that it is a knowledge intensive approach. It eliminates features irrelevant to the classification task not by noticing their joint occurrence in both positive and negative examples, as there is typically a small number of positive examples, but by noting which features are necessary for the generalized explanation. E.g., in the above example, the fact that Professor X's specialty was medieval history was not part of the explanation structure, and was therefore deemed irrelevant. Another nice effect of EBL is the optimization of the knowledge base via its compression, resulting in faster response times on similar training events. Previous papers (Katz, 1989a, and Katz, 1989b) have shown how a relatively simple connectionist system can achieve the same sort of optimization. This paper will concentrate on the way in which this neural network model can bring prior knowledge to bear in determining relevance.

ARCHITECTURE

Refer to figure 1, Panel C for this description of the system architecture. Input nodes are activated by features in the environment. These inputs are buffered by another layer, with one node for each corresponding node in the input layer. Unlike input nodes, which are clamped on or off by the environment, nodes in the input buffer may be affected by top-down control. This will prove important in the mechanism for EBL.

Activation flows from the input buffer to sets of nodes in one or more hidden layers. Solid lines represent symmetric excitatory connections, while shaded lines represent symmetric inhibitory connections. In addition, the dotted boxes in the figure are shorthand representations for sets of mutually inhibitory nodes at the same layer. Nodes in these layers also have excitatory connections to themselves. This sub-architecture has been shown to produce winner-take-all networks (Rumelhart & Zipser, 1986), that is, the node in the set receiving the most activation will reach maximum value, while all others will be driven to zero activation. Activation spreads in parallel in all directions until one unit in the output layer "wins" and becomes the decision. The relaxation process is described more fully in the next section.

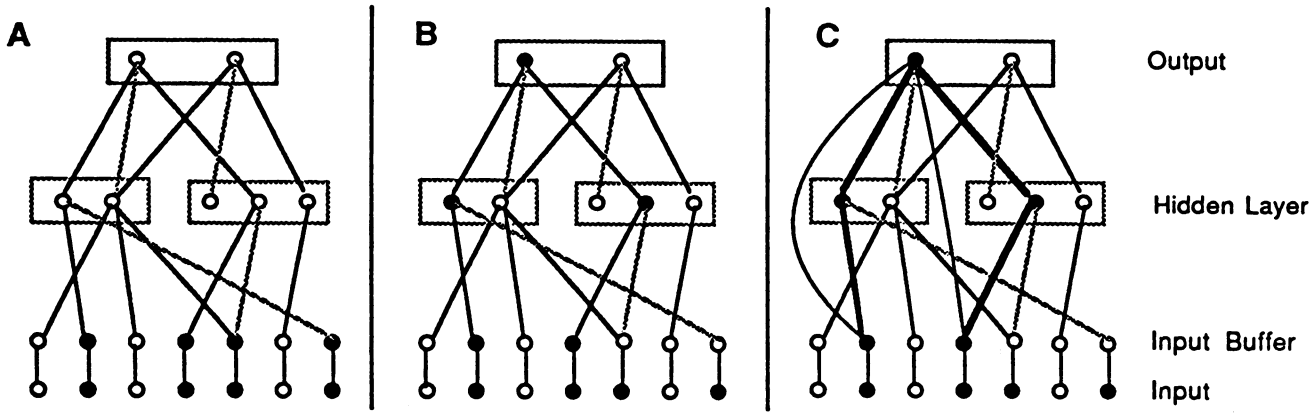


Figure 1. Achieving EBL in a neural network architecture

INFERENCE and LEARNING

Inference is accomplished by the spread of activation. The activation of a unit is a weighted sum of its inputs, as in (1). In this equation, a_i represents the net activation level of unit i, w_{ij} is the weight between units i and j, and o_j is the output of unit j.

$$a_i = \Sigma\, w_{ij}\, o_j \qquad (1)$$

Weights may be either positive (excitatory), or negative (inhibitory), and are unbounded. In contrast, the output of a unit is held between 0 and 1 by the sigmoidal function in equation (2). In this formula, T is a free parameter representing the "temperature" of the network, and θ is a constant threshold. Activation spreads until the network reaches a steady state.

$$o_i = 1 / (1 + \exp(- (a_i - \theta)/T)) \qquad (2)$$

Learning is accomplished by (3), a Hebbian associative rule modified to allow the the decrease in weight strength between two disassociated units. In conjunction with the winner-take-all decision procedure, this rule is able to classify sets of linearly separable examples. No learning is permitted directly from the input layer.

$$\Delta\, w_{ij} = \lambda\, o_i\, o_j - \delta\, |o_i - o_j| \qquad (3)$$

EXPLANATION-BASED LEARNING IN THE MODEL

Figure 1 is a highly schematic view of how EBL is accomplished in the model. Panel A represents the state of the network before relaxation. Note that the input buffer is a veridical representation of the input vector. Panel B represents the network after relaxation. Descending inhibition has turned off the two rightmost units in the input buffer (the threshold in equation 2 can also be adjusted so that merely the lack of excitatory confirmation also results in a dampened unit.). The network has "decided" that these features were not crucial in the determination of the final decision, or in the final activated state of the intermediate units leading to this decision. Unlike symbolic techniques, where relevance is determined by the explicit computation of a proof structure, in this model it is an emergent property of top-down attentional control.

Panel C represents the state of the network after learning. No correlations are made between data that was originally present in the input buffer, but turned off during relaxation (no direct learning is permitted from the input layer to other layers). Existing connections between units active at relaxation are strengthened, and new connections may also form. These "bypass" connections can be seen in C as new lines between the input buffer and the activated output node. These new connections, along with strengthened old ones, cause the network to relax at a much faster rate given a similar input pattern.

Figure 2 shows in more detail how input feature relevance may be determined in such a network. In this simplified diagram of the network used in the experiments described in the next section, only two input features are shown, one relevant, and one irrelevant. In this case, the hidden unit clusters correspond to items of *functional* significance. Prior learning with other examples has enabled the connections to the hidden layers to form, but not the connections from the functional units to the output decisions.

Processing up to the point shown in the network indicates that the object has a flat-surface, and is red. The flat-surface, along with other features (not shown) such as the fact that the surface is a comfortable temperature turn on the can-sit-on(yes) and can-lie-on(yes) units. Since weights are bi-directional, this reinforces the original input. If these weights are large enough, this input is maintained in the input buffer at relaxation, which will then develop a strengthened connection with the indicated object. The color(red) unit, however, is not maintained in the input buffer during the relaxation process. This is because the higher-order functional unit it is connected to, attracts-bull, did not receive sufficient activation from other lower-order units (not shown), such as the fact that the object is flexible enough to be waved. Recall that the threshold in equation 2 can be set such that a positive input *alone* is not sufficient to fire the corresponding unit in the input buffer. Thus previous knowledge, encoded in the form of functional information, can serve to reduce the amount of input features, and thereby simplify the resulting inductive task.

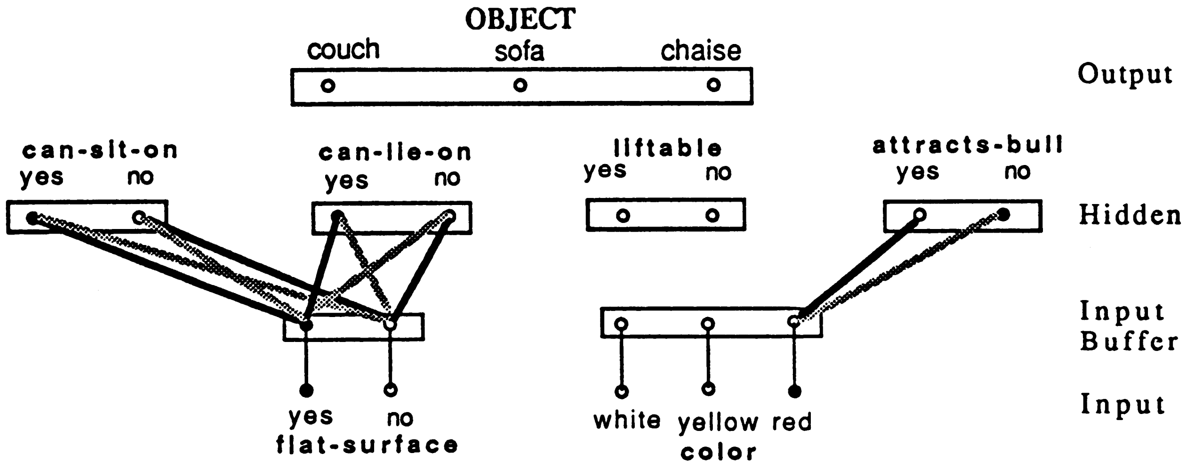


Figure 2. Determining relevance by the functional significance of the inputs

EXPERIMENTAL RESULTS

In the first set of experiments, the potential usefulness of the suppression of irrelevant features to learning was demonstrated. A network was created similar to that pictured in figure 2 with five sets of input features and three higher-order functional features. The task of the network was to correctly classify each of six randomly presented examples, two for each category of sofa, chair and chaise. Connections from the input buffer to the hidden units were hard-wired, but no connections were made to the decision units. In addition, a varying number of binary input features were added to network. The value of the feature, off or on, was determined randomly at the start of each relaxation cycle. The threshold was set such that these extraneous features were *not* suppressed by the lack of descending confirmation, to measure the deleterious effect of their addition. The graph in A of figure 3 summarizes these results. The number of sweeps through all examples before perfect recognition increased roughly linearly with the the proportion of irrelevant to relevant features.

In the second set of experiments, the number of irrelevant features was held constant (at 2.5 times the number of relevant features) but their threshold was varied. The graph in part B, figure 3 shows these results. As the threshold increases, the activation value of the feature at relaxation is lowered, and the more the network "ignores" these features. An pleasant (and unexpected) result was that the activation of the irrelevant unit at a threshold of 1.5 (when the network performed almost as well as if the irrelevant attributes did not exist), was approximately .80. That a unit can be maintained at a high level of activation and not affect learning is important since it is not known *a priori* whether the unit is relevant of not. Therefore one would like to suppress it as little as possible and yet not have it perturb the learning process if it is indeed irrelevant.

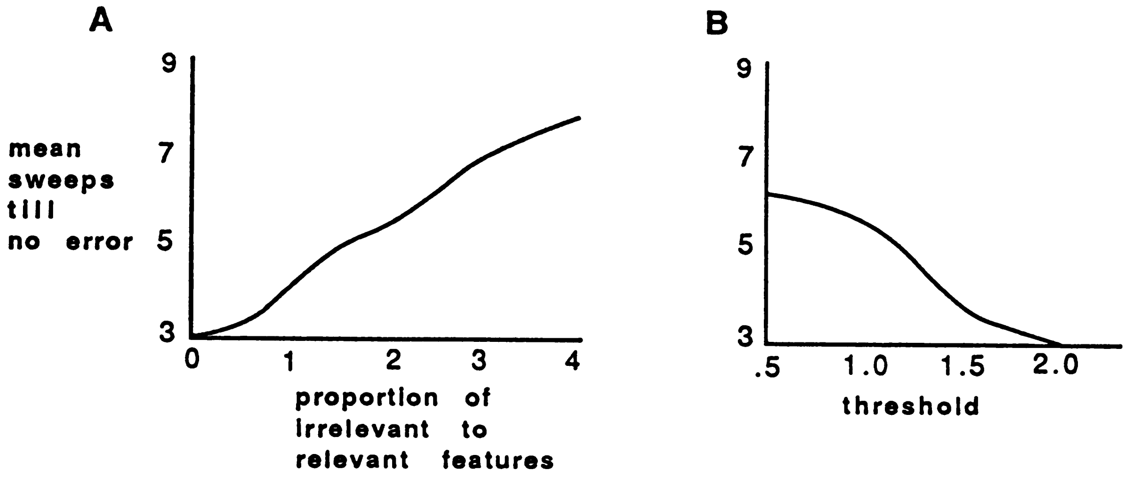


Figure 3. Experimental results

DISCUSSION

The importance of eliminating irrelevant features to assist learning has been discussed and has demonstrated in a simple connectionist system. In addition, it has been shown how varying the threshold of units in the input buffer can effect a trade-off between top-down and bottom-up influences during the learning process. The ultimate goal of this research remains similar to those of the EBL researchers working in a non-connectionist framework; viz., determining how one brings prior knowledge to bear on future inductive tasks.

REFERENCES

DeJong, G., & Mooney, R. (1986). Explanation-Based Learning: An alternative view *M.. Learning 2.*

Katz, B.F. (1989a). Integrating learning in a neural network. *Proceedings of the Sixth International Workshop on Machine Learning.*

Katz, B.F. (1989b). EBL and SBL: A neural network synthesis. To appear in *Proceedings of the 11th Annual Conference of the Cognitive Science Society.*

Mitchell, T.M., Keller, R.M., & Kedar-Cabelli, S.T. (1986). Explanation-based generalization: A unifying view. *Machine Learning 1.*

Mooney, R. & Bennet, S. (1986). A domain independent EBG. *Proc. of AAAI..*

Rumelhart, D.E., and Zipser, D. (1986). Feature discovery by competitive learning. In Rumelhart, et. al. (Eds.), *Parallel Distributed Processing, Vol. I.* MIT Press.

Merging Hebbian learning rule and least-mean-square error algorithm for two-layer neural networks

Sang-Ho Koh, Soo-Young Lee, Ju-Seog Jang, and Sang-Yung Shin
Department of Electrical Engineering
Korea Advanced Institute of Science and Technology
P.O. Box 150 Cheongryang, Seoul, Korea

Introduction

Learning algorithm has been one of the major research areas in neural networks. While Hebbian learning rule has been employeed for fixed learning neural networks[1], gradient-based least-mean-square (LMS) error algorithms have been extensively studied for adaptive learning.[2] However the most popular error-back propagation algorithms is notorious for enormous learning time. In this paper we propose a new LMS learning algorithm based on Hebbian learning rule for two-layer networks.

Formulation

Let's consider a two-layer network as shown in Fig. 1. The input layer, hidden layer, and output layer are represented by **x**, y, and **z**, respectively. In general different node numbers may be assigned to each layer.

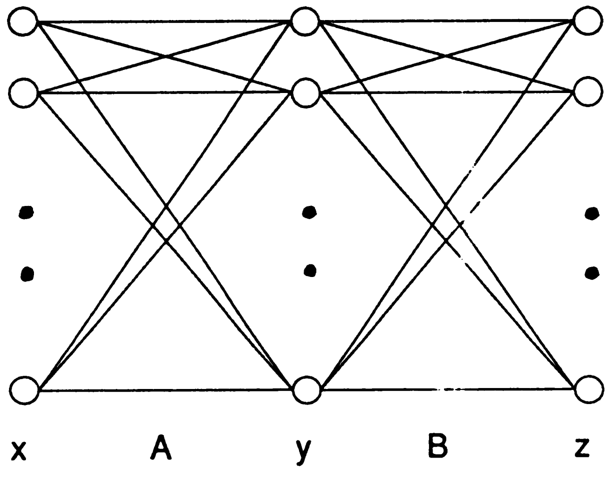


Fig. 1 Two-layer neural networks with Hebbian learning rule, i.e. $A_{ji} = \sum_s x_i^s y_j^s$ and $B_{kj} = \sum_s y_j^s z_k^s$

Suppose M sets of input $\mathbf{x}^s$ and output $\mathbf{z}^s$ (s = 1, 2, . . . , M) need to be learned. Provided corresponding hidden layer value $\mathbf{y}^s$ were known, one might determine interconnection matrices with Hebbian learning rule as

$$A_{ji} = \sum_{s=1}^{M} x_i^s y_j^s , \tag{1}$$

and

$$B_{kj} = \sum_{s=1}^{M} y_j^s z_k^s . \qquad (2)$$

The hidden layer $\mathbf{y}^s$ (s = 1, 2, . . . , M) is selected to minimize output error defined as

$$E = \sum_{s=1}^{M} |\mathbf{z}^s - \mathbf{z}(\mathbf{x}^s)|^2 , \qquad (3)$$

where $\mathbf{z}(\mathbf{x}^s)$ denotes the output vector corresponding to input $\mathbf{x}^s$ and its kth element may be represented as

$$z_k(\mathbf{x}^s) = f[\sum_j B_{kj}\, g(\sum_i A_{ji}\, x_i^s)] \qquad (4)$$

with proper nonlinear functions f(·) and g(·) for output and hidden layer, respectively. It is worthy noting that both **A** and **B** are functions of $\mathbf{y}^s$ (s = 1, 2, . . , M) and, from Eq. (3), the error E may be minimized for appropriate choice of hidden layer $\mathbf{y}^s$. We adopted the steepest descent method for the minimization as

$$y_j^s[n+1] = y_j^s[n] - \eta\frac{\partial E}{\partial y_j^s} \qquad j = 1, 2, . . , J \ , \ s = 1, 2, . . , M \ . \qquad (5)$$

Here y_j^s[n] denotes the nth iterative solution of the jth element of hidden layer vector $\mathbf{y}^s$. η is an learning coefficient affecting convergence of the iterative learning.

For auto associative memory, i.e. $\mathbf{x}^s = \mathbf{z}^s$ (s = 1, 2, . . , M), one may feedback the output into the input for improved recalls, and the two-layer network becomes similar to bidirectional associative memory (BAM) presented in Ref. [3] except the iterative learning.

Error Correction Performance

Error correction performance of this model is demonstrated by computer simulation. Ten images are learned in 6 × 8 node autoassociative memory, and error correction probabilities are plotted versus Hamming distance, i.e. number of different bits with a stored image, in Figs. 2. Results of (pseudo) random images are plotted in Fig. 2(a), and those of highly correlated images, i.e. numbers "0" to "9", in Fig. 2(b).

In the figures 1000 input images are randomly generated to satisfy required Hamming distance with each of the stored images, fed to this model and the 2-layer perceptron model, and their overall convergence characteristics are collected. For small Hamming distances the 2-layer perceptorn model has slightly higher error correction performance which may be resulted from higher degree-of-freedom of the perceptron model, i.e. whole elements of the two interconnection matrices instead of hidden layer vectors. However, for larger Hamming distances, the feed back nature of this auto-associative model greatly improves error correction performance and this new model works much better than the other. Correlation of stored images greatly decreases error correction performance as shown in Fig. 2(b). In this case optimization of bit-significance may be used to reduce the correlation.[4]

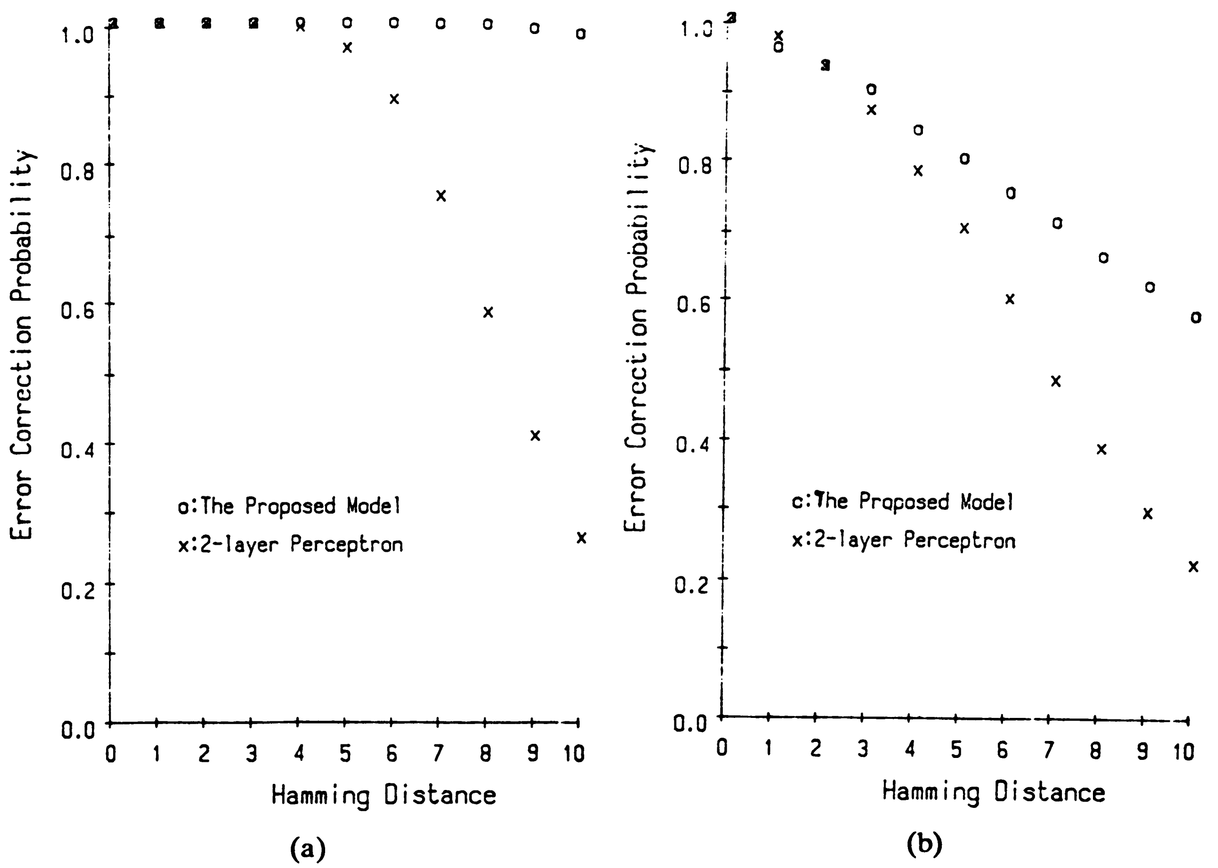


Fig. 2 Error correction probabilities versus Hamming distance.
(a) (pseudo) random images, (b) highly correlated images

Conclusion

In this paper we proposed a new adaptive learning algorithm for two-layer neural networks. In this model the two interconnection matrices are determined by Hebbian learning rule, and the output error is minimized by the steepest descent iteration for the hidden layer vectors. For auto assoeative memories excellent error correction performances of this model are demonstrated by computer simulations.

Acknowledgement : This research was supported by the Korea Science and Engineering Foundation.

References

[1] J.J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Natl. Acad. Sci. USA*, Vol. 79, pp. 2554-2558, 1982.

[2] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning representations by back-propagating errors," Nature 323 : 533-536

[3] B. Kosko, "Bidirectional associative memories," *IEEE Trans. Sys. Man* and *Cyber.*, Vol.18, pp.49-60, 1988

[4] Soo-Young Lee, Ju-Seog Jang, Jin-Soo Park, and Sang-Yung Shin, "Modification of the Hopfield model and its optical implementation for correlated images," *SPIE Pro.* 963 *Optical Computing*, pp.504-511, 1899.

Modular Neural Networks: Combining the Coulomb Energy Network Algorithm and the Error Back Propagation Algorithm

Won Don Lee[*], *Kyunghee Lee*[**], *Jongwook Jang*[**]

[*]Department of Computer Science College of Natural Sciences
ChungNam National University, Daejeon, KOREA
[**]Electronics and Telecommunications Research Institute
Daejeon, Chungnam, KOREA

ABSTRACT

Multi-layer feed-forward networks can be splitted into single-layer modules and can be trained separately if the requirement of fixing the target values in the error back-propagation(EBP) is relaxed. We propose a general form of the energy function for such networks, and show how this algorithm can be naturally combined with EBP algorithm. We also show that coulomb energy network fits the form of the modular neural networks and a demonstration is given in recognizing a Korean consonant.

INTRODUCTION

There are some problems in the error back propagation(EBP) algorithm which is used extensively as a neural network learning algorithm [Rumelhart et al., 1986]. First, since it uses a descent method to find an energy minimum, it can somtimes trap into a local minimum instead of a global minimum. Second and more serious problem comes from the fact that the algorithm adjusts the synaptic weights of the hidden layers by observing input and the corresponding output values. Because weights of the hidden layers are adjusted through propagation of errors between desired output and actual output values for a given pattern, hidden layer learning is done indirectly compared with the direct output layer learning. Therefore, when we apply the EBP algorithm to a complex network, there can be a danger that hidden layers might not have inductive capability to capture input patterns effectively. The first problem might not be a serious one depending upon network structures, but the second one is important in that it is directly linked with the problem of increasing network efficacy.

The energy function in the EBP network is described as

$$E = \sum_{p} E(p) = \sum_{p}\sum_{j} (t(p,j)-o(p,j))^2$$

where t(p,j) and o(p,j) are the target and the actual j-th bit values of the output, respectively, when input pattern p is entered to the system. The reason that EBP adjusts the hidden layer weights indirectly through propagating errors is that the target values of the output patterns are already determined as fixed points.

If we relax the requirement of the fixed target values in the output space, then it becames possible to adjust the weights in the middle layers directly. The requirement in such a network would be the property that patterns belonging to the same class be attracted together, and those belonging to different class be repelled each other. There can be no more "hidden" layers, as we can directly adjust the weights in the middle layers.

Since this algorithm does not rely on the propagation of errors through layers, the system becomes "modular" in the sense that each single-layer module can be trained independently. Multi-layer system can be made naturally by layering trained layers one by one(See Figure 1).

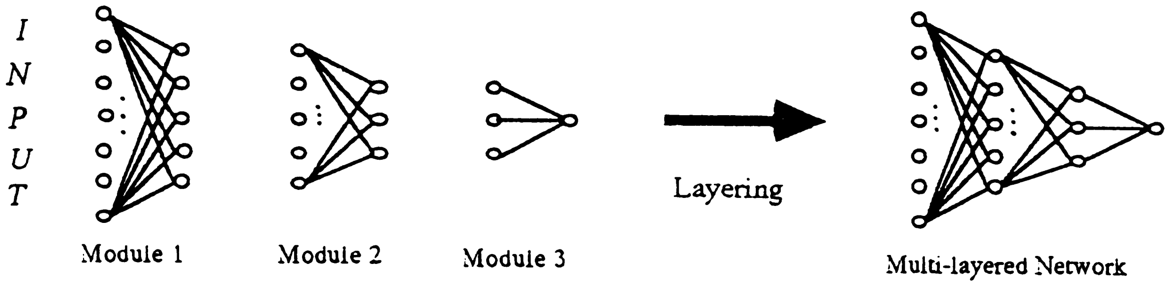


Figure 1 : Concept of modular networks. Multi-layered network is formed by layering trained modules

In general, module i is a single-layer network whose input comes from the output of the module (i-1), and whose output becomes the input of the module (i+1). Therefore, the learning of the whole system is done sequentially by first training the first module, and then training the second module until the last uppermost module is trained. Training each module separately can have advantage besides those already discussed just because it is easier to train a simple single-layer module than to train the whole multi-layer network at once.

The general form of the modular neural network energy function would be:

$$E \propto \sum_{\substack{\textit{pairs of}\\ \textit{same class}}} M_1(X_i, X_j) + \sum_{\substack{\textit{pairs of}\\ \textit{diff. class}}} M_2(X_i, X_j)$$

where M_1 and M_2 are some measure functions to determine the distance between recoded output patterns X_i and X_j. As we can see, there is a separate measure function for pairs of output patterns belonging to the same class, and another for pairs of output patterns belonging to the different class. For simplicity, we only consider the dichotomy system. The measure functions are constructed in such a way that the total energy is minimized as output patterns of same class are attracted each other and output patterns of different class are repelled each other. When we do this, the resulting vectors will be grouped together in the output space, and there may be more than or equal to one group in the space in general for each class. There has been attempts to make such an enegy function [Psaltis and Neifeld,1988], and one of them is the coulomb energy network [Scofield, 1988].

The energy function of the coulomb energy network is

$$E = 1 \,/\, (2L) \sum_{i=1}^{M} \sum_{j=1}^{M} sign(X_i, X_j)\ |X_i - X_j|^{-L}$$

Here, sign(X_i, X_j) is '-' when patterns X_i, X_j belong to the same class, and '+' when patterns belong to different class. After defining specific measures, then it is easy to find learning algorithm by gradient descent method:

$$\delta\omega_{nm} = -\eta\, \partial E/\partial\omega_{nm}$$

where eta is the learning rate. In the coulomb energy network, this becomes

$$\delta\omega_{nm} = (+/-)\,\eta\, |X(t) - X(t+1)|^{-(L+2)}\, \Delta_{nm}(p(t), p(t+1))$$

where negative sign is for subsequent patterns of the same class, and the positive sign for patterns of different class, and Δ_{nm} is defined as

$$\Delta_{nm}(p(t), p(t+1)) \equiv (X(t) - X(t+1)) \cdot \partial/\partial\omega_{nm}(X(t) - X(t+1))$$

COMBINING MODULAR NEURAL NETWORK ALGORITHM WITH EBP ALGORITHM

When single-layer modules are trained sequentially from the lowermost to the uppermost to make a multi-layer network, then for a given pattern, a specific output value will be assigned by the converged weight set. Although the target output values are not fixed by the designer, the system automatically fixes the target values during the learning process. Therefore, after training the modules by the modular learning algorithm, and making a multi-layered system, we can further train the system by the EBP algorithm by exploiting the fact that target output values are fixed after modular learning phase. Further learning by EBP algorithm can be useful when the designer wants the output gathered around a certain point. In this way, modular neural network learning algorithm and EBP learning algorithm can be combined naturally.

The combined learning algorithm can be described as follows:

Phase 1:
- (a) Train single-layer modules from the lowermost to the uppermost sequentially by modular learning algorithm.
- (b) Make a multi-layer system by layering modules.

Phase 2:
- (a) Observe and set the target values for each class.
- (b) If the output value of a pattern is far from target values of the class, then set the target value for that pattern as the one closest to it.
- (c) Use EBP algorithm to move the output value to the target value determined in (b).

SIMULATION

To demonstrate the combination of the modular neural network algorithm with EBP algorithm, we set up a network for recognizing Korean consonant "ㄱ". The input consists of 16 bits, and the two middle layers 8 and 4 bits respectively, and the output 1 bit. In Figure 2(a), the training input patterns along with the output values after phase 1 learning is shown. Here we used coulomb energy network with L=2 for each module. Noisy test "ㄱ" patterns along with the outputs of the network with the weight sets learned in phase 1 are shown in Figure 2(b). Note that the output of some test patterns are not close to target value('1'). In Figure 2(c), the output values of the patterns in Figure 2(b) are shown after phase 2 learning by EBP algorithm. This shows that those patterns which were not close to the target value become closer to it('1').

	Training Pattern	Output (Phase 1)	Test Pattern	Output (Phase 1)	Output (Phase 2)
"7"	1111 0001 0001 0001	0.897342	1111 0001 0001 0011	0.871588	0.914869
	1110 0010 0010 0010	0.888008	1110 0010 0010 0110	0.866925	0.931059
	0111 0001 0001 0001	0.892196	0111 1001 0001 0001	0.903699	0.928059
	0000 1111 0001 0001	0.893201	0000 1111 0001 0011	0.865227	0.906250
	1110 0010 0010 0000	0.908003	1110 0010 0010 1000	0.426068	0.834320
	0000 0111 0001 0001	0.878635	0000 0111 1001 0001	0.589643	0.810455
	0000 1110 0010 0010	0.861448	1000 1110 0010 0010	0.871759	0.921667
	0111 0001 0001 0000	0.895743	1111 0001 0001 0000	0.901857	0.935792
	0111 0001 0001 0010	0.868973	0111 0001 0001 0011	0.865518	0.900577
	1110 0010 0010 0100	0.894385	1110 0010 0010 1100	0.348479	0.790407
Not "7"	0000 0000 0000 1111	0.172287	0000 0000 0000 1111	0.176576	0.176535
	1000 1000 1000 1111	0.087116	1000 1000 1000 1111	0.089216	0.079293
	1000 1000 1111 0000	0.076874	1000 1000 1111 0000	0.078625	0.075550
	0100 0100 0111 0000	0.087248	0100 0100 0111 0000	0.089311	0.101868
	1111 1000 1000 1111	0.089773	1111 1000 1000 1111	0.092149	0.092205
	0111 0100 0100 0111	0.080508	0111 0100 0100 0111	0.082409	0.072183
	1110 1010 1010 1110	0.101767	1110 1010 1010 1110	0.104827	0.186210
	0111 0101 0101 0111	0.081790	0111 0101 0101 0111	0.083782	0.076785
	1111 1001 1111 0000	0.083144	1111 1001 1111 0000	0.085262	0.097414
	1110 1010 1110 0000	0.090545	1110 1010 1110 0000	0.092984	0.148327
	0111 0101 0111 0000	0.111303	0111 0101 0111 0000	0.114273	0.160169
	0000 1110 1010 1110	0.080612	0000 1110 1010 1110	0.082599	0.089475
	0000 0111 0101 0111	0.096213	0000 0111 0101 0111	0.098586	0.105523
	1111 1001 1001 1111	0.091624	1111 1001 1001 1111	0.094128	0.099939
	(a)		(b)		(c)

Figure 2 : Experimental result for demonstrating the combination of modular neural network algorithm(a,b) and the EBP algorithm(c).

DISCUSSION

We have described a general form of the energy function for modular neural networks and shown that the coulomb energy network fits the form. We have also proposed a general scheme to combine modular neural network algorithm and the EBP, and demonstrated the experimental result. By relaxing the requirement of the fixed target values, the whole feed-forward networks can be splitted into single-layer modules to be trained separately. This helps to control the middle layer weights directly and therefore increases the chance of making complex systems. EBP is shown to be helpful to train the network further after each module is separately trained by the modular learning algorithm.

REFERENCES

Rumelhart,D.E., Hinton,G.E., Williams,R.J.: Learning internal representations by error propagation, in D.E. Rumelhart and J.L. McClelland (Eds.), Parallel Distributed Processing, MIT Press, 318-364(1986).

Psaltis,D., Neifeld,M.: The emergence of generalization in networks with constrained representations, Proceedings of the IEEE International Conference on Neural Networks, Vol.1, 371-381(1988).

Scofield,C.L.: Learning internal representations in the coulomb energy network, Proceedings of the IEEE International Conference on Neural Networks, Vol.1, 271-276(1988).

ANALYSIS OF DECISION CONTOUR OF NEURAL NETWORK WITH SIGMOIDAL NONLINEARITY

Ho Chung Lui
Institute of Systems Science, National University of Singapore
Kent Ridge, Singapore 0511.
BITNET: ISSLHC@NUSVM

Abstract

The decision contour of multi-layer feedforward neural networks with sigmoidal nonlinearity is analysed. Unlike the linear threshold case, the decision boundary can be a curved surface. Moreover, a family of contours can be obtained, each corresponding to a particular output value. The contour typically consists of straight line segments joined by a smooth rounded surface. It is also possible to identify which hidden unit is affecting which part of the contour. In addition, networks with a single hidden layer are found to be capable of forming disjointed decision regions.

1 Introduction

A linear classifier can be implemented using simple neural network with threshold nonlinearity[1]. By joining several linear classifiers together with AND/OR logic, Lippmann observed that a network with two hidden layers can form arbitrarily complex, disjointed decision boundaries[1]. Recently, Makhoul et al. reported that networks with one hidden layer can form disconnected regions[4]. Other researchers also analysed neural networks using the geometric approach and suggested alternative learning algorithms[7,6]. However, all these results were based on the linear threshold function. On the other hand, the smooth sigmoidal nonlinearity is widely used for the popular Backpropagation algorithm. This paper analyses the formation of decision contour for the sigmoidal nonlinearity. In this case, a family of decision contours emerges, each contour corresponding to a particular output value. The contour typically consists of linear segments. Unlike the threshold case, segments are joined together by smooth, rounded surfaces. Experimental results indicate that complex, disconnected decision regions can be formed by a network with a single hidden layer.

2 Theory

Consider the following sigmoidal nonlinear function:

$$f(x) = \frac{1}{1+e^{-x/T}} = \frac{1}{2}[1+\tanh(\frac{x}{2T})] \tag{1}$$

Notice that as $T \to 0$, $f(x)$ approaches the threshold function. Thus the results of the following analysis also apply for the threshold nonlinearity as a limiting case. For the rest of the discussion, T is taken to be $1/2$ for clarity.

The output z of a simple two-layer (input and output) network is:

$$z = f(\sum_{i=1}^{N} w_i x_i + \theta) = \frac{1}{2}[1+\tanh(\sum_{i=1}^{N} w_i x_i + \theta)] \tag{2}$$

where w_i are the connection weights from input unit x_i, θ is the bias and N is the dimension of the input space. At any output value $z_0 \in [0,1]$, we have

$$z_0 = \frac{1}{2}[1+\tanh(\sum_{i=1}^{N} w_i x_i + \theta)] \tag{3}$$

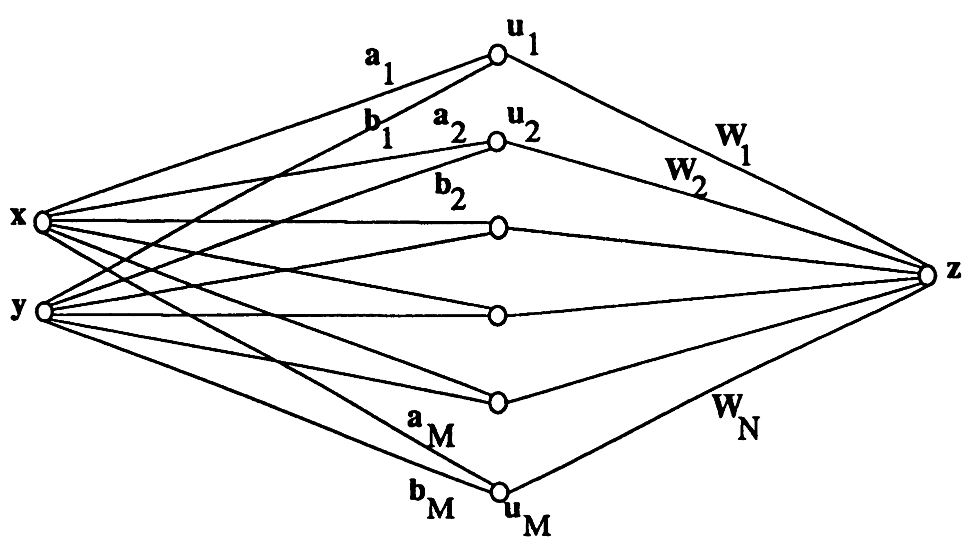


Figure 1: Neural network with one hidden layer.

which is equivalent to

$$\sum_{i=1}^{N} w_i x_i = \gamma \tag{4}$$

where $\gamma = \mathbf{arctanh}(2z_0 - 1) - \theta$. Thus the decision contour is a linear hyperplane in R^N for each z_0. For a two-input, one-output network with one hidden layer as shown in Fig. 1, we have from eq. 4

$$\sum_{i=1}^{M} W_i u_i = \gamma \tag{5}$$

where M is the number of hidden nodes in the network, W_i is the weight between the i^{th} hidden node to the output node, and the response of the i^{th} hidden node u_i is:

$$u_i = \frac{1}{2}[1 + \tanh(a_i x + b_i y + \theta_i)] = \frac{1}{2}[1 + \tanh(\xi_i)] \tag{6}$$

where a_i and b_i are weights from the input units to the i^{th} hidden node. Substituting eq. 6 to eq. 5 and after simplifying, we obtain

$$\sum_{i=1}^{M} W_i \tanh(\xi_i) = \gamma' \tag{7}$$

Thus the greater the magnitude of $|W_i|$, the more influential this hidden node will be on the overall network response. Notice that $\tanh(\xi)$ is a monotonic increasing function bounded between -1 and $+1$. Its range can be roughly partitioned into two 'satuation' regions, (when $|\xi| \gg 5$, $\tanh(\xi) \sim \pm 1$ and is insensitive to ξ) and an 'active' region in between them. In other words, if $\tanh(\xi_j)$ is in the saturation region, then $W_j \tanh(\xi_j) \simeq \pm W_j$. Hence, for those (x, y) coordinates which satisfy eq. 7 and all but one of the $\tanh(\xi_i)$ term are in the saturation region, eq. 7 can be approximated by:

$$W_i \tanh(\xi_i) \simeq \sigma \tag{8}$$

Eq. 8 describes a straight line, whose slope is determined by $(-a_i/b_i)$. Hence, when only one hidden node is in its active region, it dominates the shape of the decision contour and the contour becomes a straight line. In fact, it can be considered that each hidden node induces a set of straight lines at different response levels to graduately separate the input space into two parts. The output node then combines the regions together, through the weights W_i and the output bias θ, to form a proper decision boundary to separate the input patterns. This is not unlike the threshold nonlinearity case. In the present case, however, every output value $z_0 \in [0,1]$ defines a decision contour. The family of these contours forms a smooth decision surface.

When more than one hidden node enter the active region, they interact with one another to form a smooth, rounded contour. The analysis is more complex but the function in eq. 7 is a well-behaved function and high-order derivatives exist. Let us consider the slope of the contour m:

$$m = \frac{dy}{dx} = -\frac{\sum_{i=1}^{M} W_i a_i \operatorname{sech}^2(\xi_i)}{\sum_{i=1}^{M} W_i b_i \operatorname{sech}^2(\xi_i)} \tag{9}$$

Notice that $\operatorname{sech}(x) \in (0,1]$ for $x \in (-\infty,\infty)$, so $\operatorname{sech}^2(x) \in (0,1]$ also. When the i^{th} node becomes dominant, we have from eq. 9 :

$$m = \frac{dy}{dx} \simeq -\frac{W_i a_i \operatorname{sech}^2(\xi_i)}{W_i b_i \operatorname{sech}^2(\xi_i)} = -\frac{a_i}{b_i}$$

which reiterates that the trajectory is a straight line with slope $(-a_i/b_i)$. Notice that the slope is basically a continuous function, except for those points where the demoninator is zero [1]. When the dominance starts to shift from node i to node j, the slope move gradually from $(-a_i/b_i)$ to $(-a_j/b_j)$. As a result, the trajectory is a smooth, continuous curve which connects the two straight lines together.

It is therefore possible to identify which hidden unit is responsible for which part of the contour. The overall influence, however, depends also on the magnitudes and signs of W_i, their relative strength against one another, and against the output bias θ. When $|W_k|$ is comparatively much smaller, the node can be eliminated without affecting the contour. Thus the parameters $\{a_i, b_i$ and $\theta_i\}$ determine the slopes and placements of straight lines while $\{W_i\}$ determine their orientations and relative strengths.

3 Experiments

A neural network software package from McClelland and Rumelhart[5] was used in these experiments. The software was further enhanced by a color graphic package which displayed the network structure and the decision region[3]. Information such as weights, biases, neural responses and decision contours are all color-coded for ease of visualization. However, the code to implement the Backpropagation algorithm was not modified. Several network structures similar to Fig. 1 were used. Input data were manually generated and were confined within the unit square bounded by (0,0), (1,0), (0,1) and (1,1). Fig. 2 and 3 depict the results after training. Training data are represented by symbols '+' and 'x' in the unit square. A '+' means that the target output should be '1' whereas an 'x' corresponds to a target value of zero.

In Fig. 2, the '+' data are concentrated at the center of the unit square, while the 'x' symbols are scattered along the perimeter. The decision contours clearly reveal that, for low output values, the main components are four straight line segments. For high output values, the hidden units interact with one another to form an oval shaped contour. Fig. 3 shows that the same network topology is capable of forming complex, disjointed decision regions. The data set in this case consists of two disjointed '+' sets at the upper left and lower right corners of the unit square. They are surrounded by the 'x' data. Again the contours consist of mainly straight lines, joined together with rounded corners. Futher analysis of the network parameters shows that hidden units 2, 3, 5 and 6 (unit 1 = top) are mainly responsible for the formation of the decision surface.

It should be noted that while the multi-layer neural network is capable of forming complex decision regions to separate patterns, the Backpropagation learning algorithm may not always be able to find the right solution. It depends heavily on the initial conditions. A more thorough set of experiments have been done to study the effectiveness of different network configurations against five sets of manually generated pattern[2].

[1] In this case, we can consider $1/m = dx/dy$, which is a smooth function within this neighborhood.

Conclusion

The decision contour of neural networks based on sigmoidal nonlinearity is analysed. For networks with one hidden layer, the contour becomes a linear segment when a single hidden node is in the 'active' region. When more that one node is active, the contour is a smooth, continuous curve. Experimental results show that complex, disjointed regions can be formed on networks with only one hidden layer. However, the popular Backpropagation learning algorithm may not be able to construct them reliably.

Acknowledgement

The author would like to thank Dr. K.P. Choi and Prof. Teh for their valuable comments.

References

[1] Richard Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, 4(2):4, 1987.

[2] H. C. Lui. Decision boundary formation from the back-propagation algorithm. In *Proceedings of International Symposium on Computer Architecture and Digital Singal Processing*, Hong Kong, 1989.

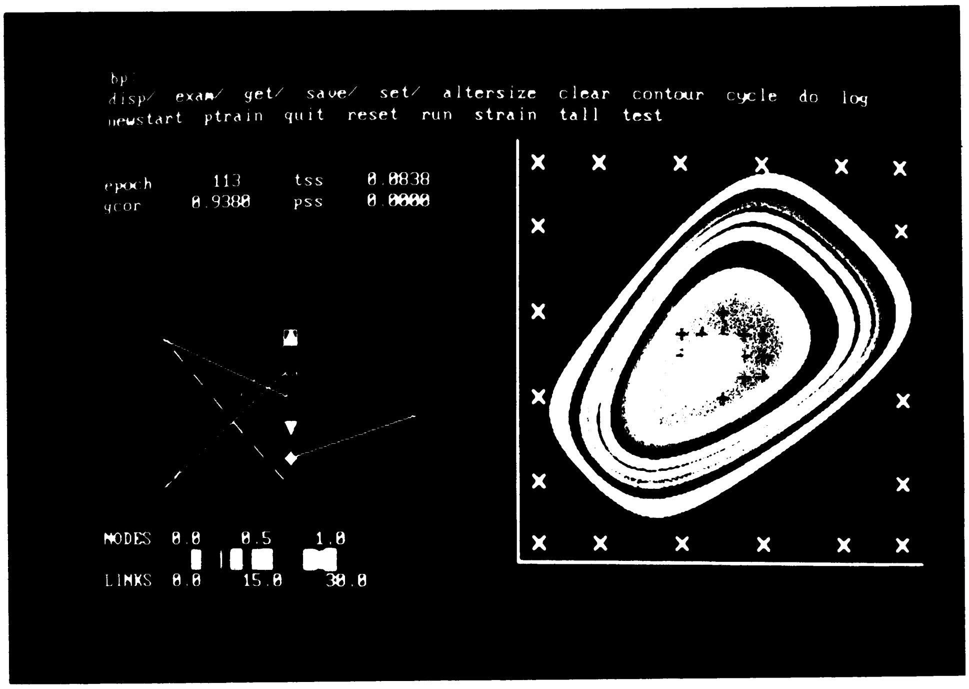
[3] H. C. Lui and Melvin Cheong. Graphical visualization of multi-layer neural network. In *Proceedings of the Inter Faculty Seminar on Neuronet Computing*, page 33, Natinal U. of Singapore, 1989.

[4] John Makhoul, Amro El-Jaroudi, and Richard Schwartz. Formation of discounnected decision regions. In *Proceedings of International Joint Conference on Neural Network*, page 455, IEEE, 1989.

[5] James L. McClelland and David E. Rumelhart. *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises.* MIT Press, Cambridge, MA, 1988.

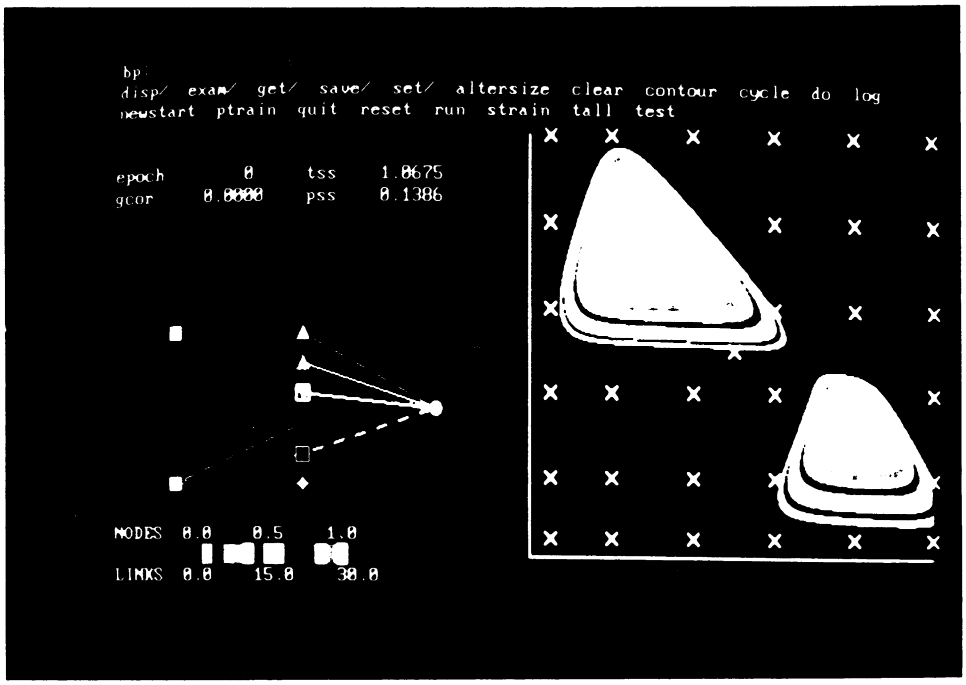
[6] U. Ramacher and Wesseling M. A geometrical approach to neural network design. In *Proceedings of International Joint Conference on Neural Network*, page 147, IEEE, 1989.

[7] Pal Rujan and Mario Marchand. A geometric approach to learning in neural networks. In *Proceedings of International Joint Conference on Neural Network*, page 105, IEEE, 1989.


bp:
disp/ exam/ get/ save/ set/ altersize clear contour cycle do log
newstart ptrain quit reset run strain tall test

```
epoch    113      tss    0.0038
gcor     0.9380   pss    0.0000

NODES 0.0    0.5    1.0
LINKS 0.0    15.0   30.0
```

Figure 2: Neural network topology and decision contour.


bp:
disp/ exam/ get/ save/ set/ altersize clear contour cycle do log
newstart ptrain quit reset run strain tall test

```
epoch      0      tss    1.0675
gcor     0.0000   pss    0.1386

NODES 0.0    0.5    1.0
LINKS 0.0    15.0   30.0
```

Figure 3: Disjoint decision contours.

A Learning Algorithm based on Prediction

Akihiko MACHIZAWA
Communications Res. Lab. MPT
2-1, Nukui-Kitamachi 4-chome, Koganei-shi, Tokyo 184, JAPAN

1. Introduction

Animals extract necessary information from a lot of indiscriminate outside information with their neural systems. Linsker presented a learning algorithm which preserves maximum information.[1] Its neuron can not extract information. There are some learning algorithms of neural synapse. Although, conventional algorithms need either teach signals (desired output) or no teach signals, biological neuron can learn both with teach signals and without teach signals. This article presents a new learning algorithm which extracts information based on Wiener's information. The algorithm can learn both with teach signals and with no teach signals. From simulation results, with teach signals the neuron model has performance to categorize patterns and with no teach signals connections of synapse are formed like retinal neuron.

2. Information extraction

Signals are divided to two parts (Fig.1); one is possible to be predicted with others, and another is impossible. Let $X_1, X_2, \cdots, X_n$ as information sources, and from information theory,

$$H(X_1)+H(X_2)+\cdots+H(X_n) \geq H(X_1,X_2,\cdots,X_n) \tag{1}$$

with equality only if the events are independent. Though, the information quantity is usually defined as Shannon's entropy[2] , Wiener said the part which could not be predicted was real information.[3] We consider the information extraction based on

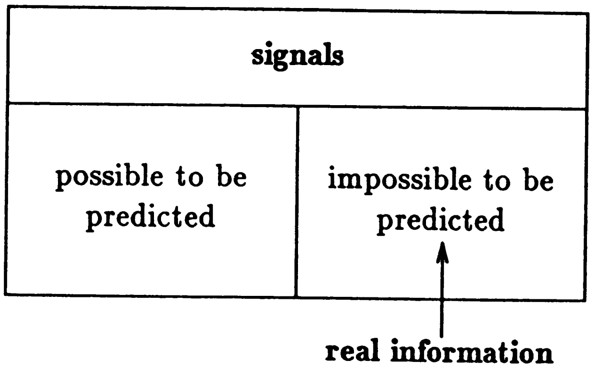


Fig.1 Wiener's information.

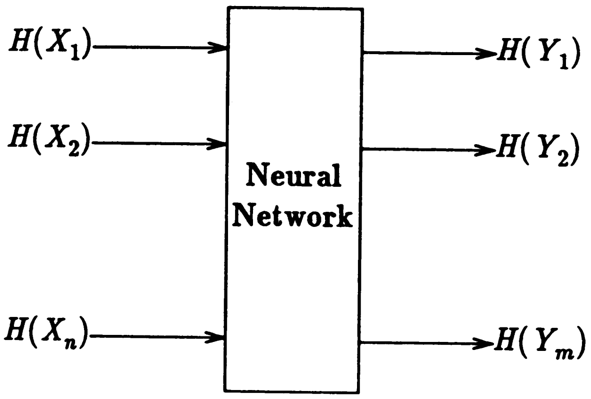


Fig.2 Neural network architecture.

Wiener's information definition. Total information $H(X_1)+H(X_2)+\cdots+H(X_n)$ is divided to real information $H(X_1,X_2,\cdots,X_n)$ and other redundant information.

Let the predictive value $\overline{x_i}$ of x_i as weighted sum without itself given by,

$$\overline{x_i} = \sum_{\substack{j=1 \\ j\neq i}}^{n} w_j x_j \tag{2}$$

and the predictive error y_i is given by

$$\begin{aligned} y_i &= x_i - \overline{x_i} \\ &= \sum_{j=1}^{n} w_j x_j. \end{aligned} \tag{3}$$

3. Learning Algorithm

The weights $\mathbf{w}_i=\{w_{ij}\}$ are defined as minimizing mean square of predictive error to extract information,

$$I = \min_{\mathbf{w}_i} \mathrm{E}[\, y_i^2 \,] \tag{4}$$

and larger weights are suitable to transmit more information,

$$C = \min \frac{1}{\sum_j (w_{ij})^2}. \tag{6}$$

Therefore the neurons learn minimizing (4) and (5) eq. simultaneously with gradient method.

$$E = \mathrm{E}[\, \alpha I + \beta C \,] \tag{7}$$

The neurons learn with no teach signals when all weights w_{ij} varies freely.

$$\begin{aligned} \delta w_{ij} &= \varepsilon \frac{\partial}{\partial w_{ij}}[\, \alpha I + \beta C \,] \\ &= \varepsilon \left[2\alpha x_j \sum_k w_{ik} x_k - \frac{2\beta w_{ij}}{\{\sum_k (w_{ik})^2\}^2} \right] \end{aligned} \tag{8}$$

The other hand, the neurons learn with teach signals when one weight w_{ih} is large and fixed.

$$\delta w_{ij} = \begin{cases} \varepsilon \left[2\alpha x_j \sum_k w_{ik} x_k - \dfrac{2\beta w_{ij}}{\{\sum_k (w_{ik})^2\}^2} \right] & (j\neq h) \\ 0 & (j=h) \end{cases} \tag{9}$$

4. Simulation results

4.1. With no teach signals

A neuron has 15×15 (height × width) synapses, training data is a set of variety parts of a gray scale picture (Fig.3(a)), and no teach signals are desired. After learning, a center-surround neuron shown as Fig.3(b) is formed. Fig.3(c) shows an output of an array of such neurons. This artificial neuron is similar to $\nabla^2 G$ known as biological retinal neuron model[4] shown in Fig.3(d),3(e).

4.2. With teach signals

The task is categorizing three input patterns (size: 4×4) shown in Fig.4 and activating three output neurons for each pattern. Presented algorithm has almost same categorizing performance to Widrow-Hoff algorithm,[5] because of similar evaluation function.

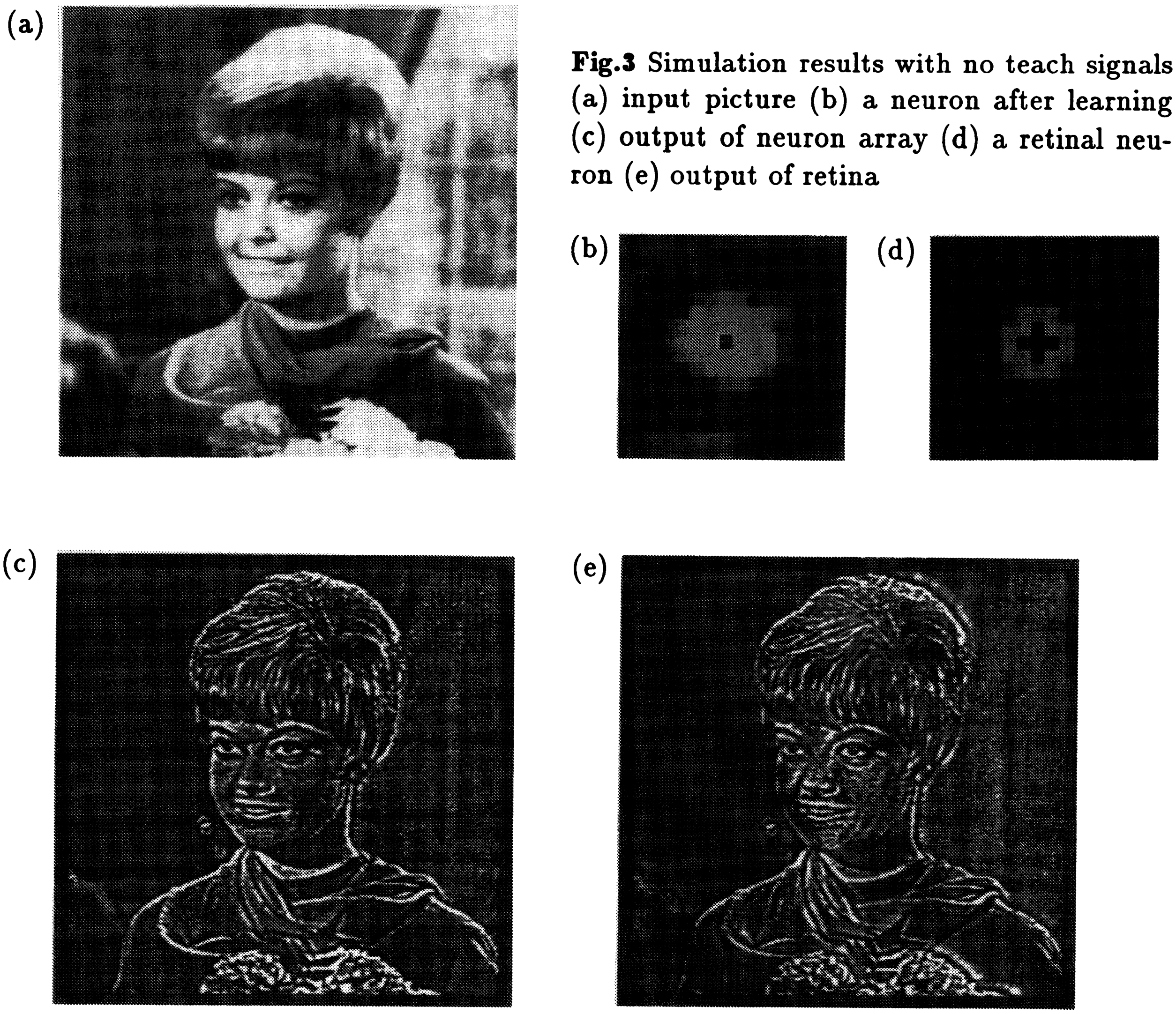


Fig.3 Simulation results with no teach signals (a) input picture (b) a neuron after learning (c) output of neuron array (d) a retinal neuron (e) output of retina

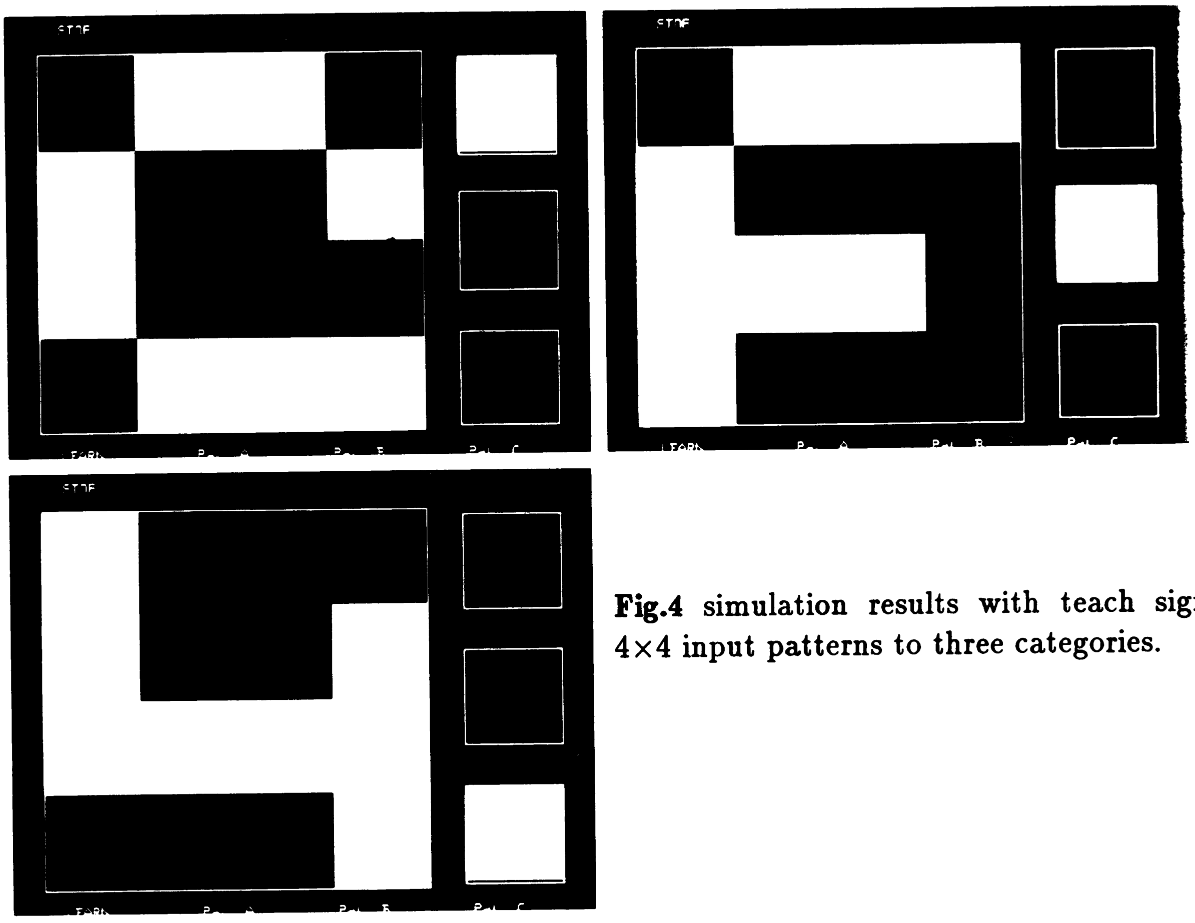


Fig.4 simulation results with teach signals. 4×4 input patterns to three categories.

5. Conclusion

This article presents a new learning algorithm which extracts information based on Wiener's information. The neuron makes its output minimum to predict one input with others. This principle contribute to not only information extraction but also biological energy saving. This article, though, discusses a learning algorithm for only one neuron. Competitive learning and self-organization are future studies.

References

1. LINSKER, Ralph, "From Basic Network Principles to Neural Architecture: Emergence of Spatial-opponent Cells," *Proc. Natl. Acad. Sci. USA*, vol. 83, pp. 7508-7512,8390-8394,8779-8783, Oct. 1986.
2. SHANNON, C. E., "A Mathematical Theory of Communication," *Bell Syst. Tech. J*, vol. 27, pp. 379-423, 623-656, 1948.
3. WIENER, Nobert, in *Cybernetics, 2nd ed.*, John Wiley & Sons, Inc., New York, 1961.
4. MARR, David, in *Vision*, W.H. Freeman and Company, New York, 1982.
5. WIDROW, B. and M. E. HOFF, "Adaptive switching circuits," *Conv. Rec. of Inst. Radio Eng., West. Electr. Show and Conv.*, vol. Part 4, pp. 96-104, 1960.

A SYSTEM IN CONTROL OF ITS KNOWLEDGE THAT PROVIDES ALTERNATIVE AND DIFFERENT SOLUTIONS FROM ONE INPUT SET

By
Oscar Martinez and Craig Harston
Computer Applications Service
6207 Forest Trail
Signal Mountain, TN 37377

Abstract

This is an associative neural network that can store a large number of associations and can provide many solutions to an individual input pattern. Responses are controllable making the system very powerful. The sensitivity of the input is measured in the system exposing several matches stored on memory. The system does not look for a matching solution to the input, instead the input is used as a clue to finding many solutions. The system can provide enormous reduction of the memory size, and shows control of its knowledge.

Introduction

We still find many limitations for what neural networks can do:

1) Neural networks are limited to an optimal goal and that is to find an exact match to the given input. Finding the exact match does not always mean getting the right answer. We could have an incomplete or noise input misleading the search for a response.

2) Saturation is another problem that limit the number of associations that can be stored in memory. Saturation also forces us to increase unnecessarily the size of the memory in order to increase the number of associations even for a small fixed size application.

3) Most neural networks do not grow as intelligent systems, because they do not make use of their knowledge. They do not show to the user the relations among similar experiences that occurred during the training.

We have a system which capacity is $m = +.6n$ (figure 1) where n is the number of neurons (m has been determined by experiments). Since it is possible to store several associations in a small memory, many of them could have similar patterns, so finding the best match may not be the right solution. The goal is to find all the possible answers. The system provides the necessary information for us to make a decision. The right answer always depends on the type of application for which the system is used. For example: in medicine, we need to find information about symptoms and diseases. Instead of just having a confirmation of a disease by giving the symptoms, the system could also list the consequences that could follow from the current condition of the patient.

SYSTEM IN CONTROL OF ITS KNOWLEDGE

Informative Associative Network Development

The system was designed to provide more than one correct answer to a given input pattern in an organized manner. The memory is built using the outer product of a vector with itself.

vector Z(n)
matrix T(n,n)

$T_{i,j} = T_{i,j} + (Z_i * Z_j)$

Memory-Input interaction

We are measuring all the levels of reactions between each input neuron and the memory finding different frequencies.

We know that neurons react at different levels or intensities of stimulus producing a greater or smaller interaction (excitation) among them. These stimulus levels are controlled in the system and reflected in the output.

Martinez-Harston (MH) Frequency

The calculated interaction between memory and input is called MH frequency. For every output neuron there is a MH frequency calculated in the first recall. Some neurons may have the same MH frequency. The MH frequency is stored in the output neuron array. A duplicate of these values are sorted and stored in tables (figure 2).

MH Frequency Table: The tables are used to control the output responses. All the MH frequencies are stored in an ascending or descending order to control how high or low we want the intensity of the stimulus to be. As a result, the frequency values are stored in two tables.

REDUCTION TABLE: The best solution corresponds to the highest frequency in the table. The alternative best solution follows the highest frequency in a descending order.

AMPLIFICATION TABLE: The best solution corresponds to the lowest frequency in the table. The alternative best solution follows the lowest frequency in an ascending order.

Bypassing the memory

After storing the MH frequency values in the tables, it is unnecessary to recalculate the MH frequencies because the input is not changed. Response time is almost zero for every output neuron, since we only have to compare the value in the output array with the one given from the frequency table (figure 2).

THE HIDDEN OBSERVER

Different Answers

The system can find different responses at two different recalls even when the input is the same.

There is a weak side that shows the reaction of neurons that have been less involved than other neurons during the learning process, but they do stimulate the system for a response. However, there is also a strong side that shows the reaction of neurons that were more active in the process of learning. Figure A and B show two different responses for a single input.

The Hidden Observer

A similar behavior has been found in our brain. [5] "According to the researchers, it seems to be a prime example of a mental dissociation, two parts of consciousnesses split off from one another".

"Even more intriguing are a small but growing number of studies that show that during hypnosis the brain's right hemisphere becomes more active in relation to the left hemisphere, perhaps because the left side is somehow inhibited (Banyai, Mesaros and Csokay, 1985; Edmonston, 1985; Gruzelier et al., 1984, 1985; Karlin, Cohen, and Goldstein, 1984 ...)."

REFERENCE

[1] J. A. Anderson, "A simple neural network generating an interactive memory," Mathematical Biosciences, vol. 14, pp.197-220, 1972.

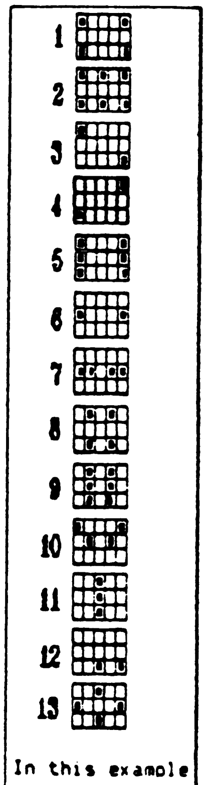
[2] T. Kohonen, "Correlation matrix memories," IEEE Transactions on Computers, vol. C-21, pp.353-359, 1972.

[3] S. Grossberg, "A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine," Computer Vision, Graphics, and Image Processing, vol. 37, pp.54-115, 1987.

[4] J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," Proceedings of the National Academy of Sciences, vol. 70, pp.2554-2558, 1982.

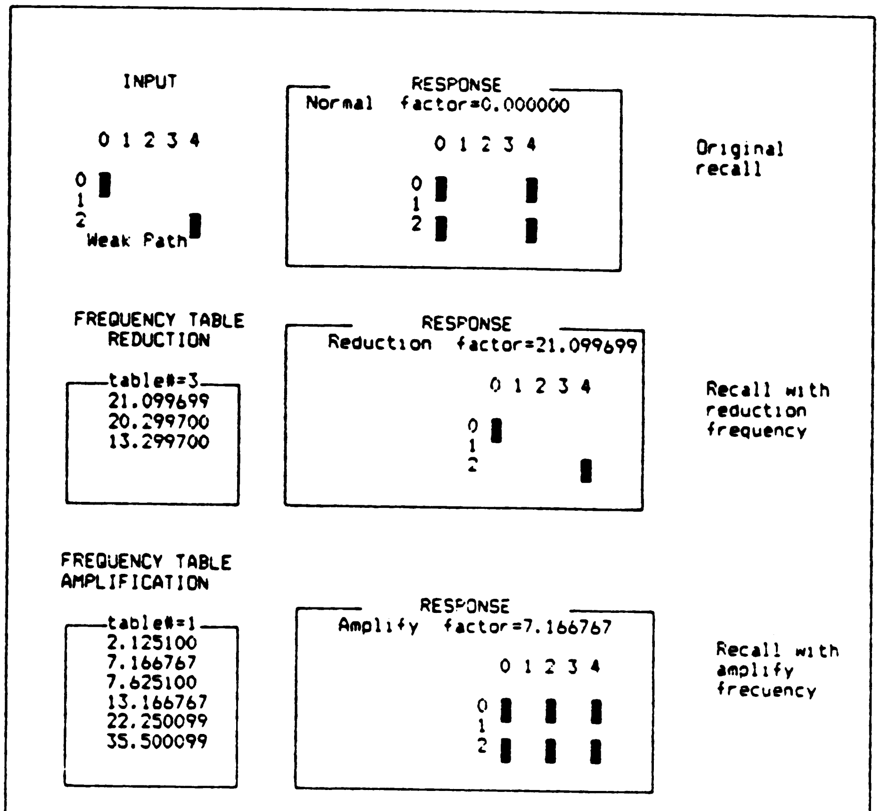
[5] Camille B. Wortmam and Elizabeth F. Loftus, "Studying The Hidden Observer" , Psychology, pages 328-329, 1988.

[6] Ed Reitman " The Hebb Learning Rule," Experiments in Artificial Neural Networks, pp. 46-53.



In this example
m= .8 x 15

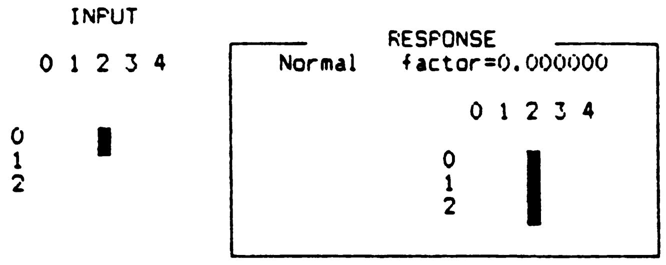
FIGURE 1



```
INPUT                    RESPONSE
                         Normal  factor=0.000000
 0 1 2 3 4
                           0 1 2 3 4                Original
0 ▮                                                 recall
1                        0 ▮   ▮
2          ▮             1
 Weak Path               2 ▮   ▮

FREQUENCY TABLE          RESPONSE
REDUCTION                Reduction  factor=21.099699
 table#=3                  0 1 2 3 4                Recall with
 21.099699                                          reduction
 20.299700               0 ▮                        frequency
 13.299700               1
                         2         ▮

FREQUENCY TABLE          RESPONSE
AMPLIFICATION            Amplify  factor=7.166767
 table#=1
 2.125100                  0 1 2 3 4                Recall with
 7.166767                                           amplify
 7.625100                0 ▮ ▮ ▮                    frecuency
 13.166767               1
 22.250099               2 ▮ ▮ ▮
 35.500099
```

FIGURE 2

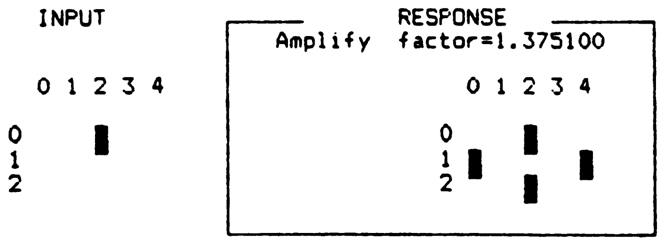
```
FREQUENCY TABLE
 AMPLIFICATION
* Strong Path *

 table#=1
 4.000100
 4.888989
 5.600100
 7.333433
```



```
INPUT                RESPONSE
                     Normal   factor=0.000000
```

Correspond
to figure 1
association #11

```
FREQUENCY TABLE
AMPLIFICATION
* Weak Path *

 table#=1
 1.375100
 1.500100
 2.300100
 2.375100
```



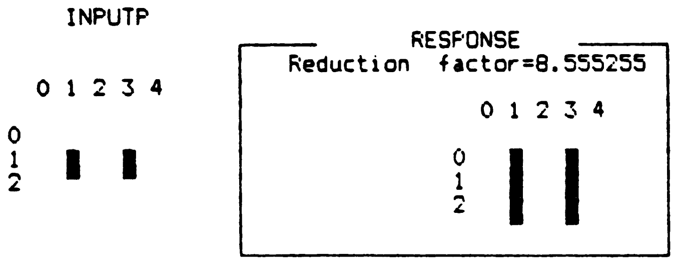
```
INPUT                RESPONSE
                     Amplify  factor=1.375100
```

Correspond
to figure 1
association #13

This is example #1 of two different answers from
two different recalls using the same input.

FIGURE A

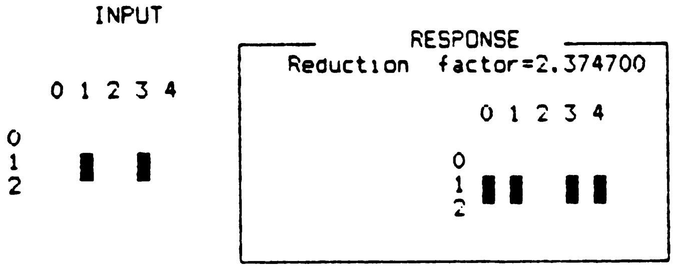
```
FREQUENCY TABLE
   REDUCTION
*  Strong Path *

 table#=3
 24.428270
 8.555255
 4.636064
 0.199700
```



```
INPUTP               RESPONSE
                     Reduction   factor=8.555255
```

Correspond
to figure 1
association #9

```
FREQUENCY TABLE
  REDUCTION
* Weak Path *

 table#=3
 24.166367
 2.374700
 1.749700
```



```
INPUT                RESPONSE
                     Reduction   factor=2.374700
```

Correspond
to figure 1
association #7

This is example #2 of two different answers from
two different recalls using the same input.

FIGURE B

FAST QUADRATIC SEPARATION USING A SINGLE-LAYER INTERCONNECT MODEL

Jack L. Meador
Electrical and Computer Engineering
Washington State University
Pullman Wa. 99164-2752

ABSTRACT

This paper details a performance comparison between a single-layer axo-axonic neural network and a two-layer backpropagation network. The axo-axonic interconnect model used is based upon axo-axo-dendritic connections between neurons. This kind of interaction is modeled as a multiplicative dependency between neuron inputs. These dependencies allow for the single-layer representation of input pattern classes that are separable by a generalized quadratic function. Even though the single-layer axo-axonic model is computationally more complex than conventional single layer networks, the specific results reported suggest that it is less complex than a multi-layer network using backpropagation. It not only requires fewer computations per epoch, it also converges in fewer epochs.

Introduction

Neurobiological research has established the existence of several kinds of synaptic junctions. That which is the most studied in artificial neural networks is the axo-dendritic variety where the axon of the pre-synaptic neuron excites the dendritic membrane of the post-synaptic neuron. Another variety known to exist is the axo-axonic synapse. In this case, an axon process terminates upon another axon process instead of a dendrite. When the former process releases neurotransmitter, the effectiveness of an axo-dendritic synapse associated with the latter process is modulated. Axo-axonic synapses have been documented in a wide variety of natural neural circuits [1,2]. Another type of connection is the dendro-dendritic synapse. Sigma-pi units [3] and product units [4] for multi-layer networks are inspired by this third kind of synapse. Even though a more detailed model of natural networks would ultimately include the effects of all three kinds of connections and perhaps others, the focus of this work is strictly upon a combination of axo-dendritic and axo-axonic varieties.

Interconnect Model

An axo-axo-dendritic synapse between two inputs is modeled as a multiplicative modulation of one input by the other [5]. Given two inputs x_j and x_k, axo-axonic control of x_j by x_k can be expressed by the second-order term:

$$d_{ij}x_j(1+a_{ijk}x_k) \tag{1}$$

Where a_{ijk} represents the strength of the axo-axonic synapse from x_k to x_j and d_{ij} represents the axo-dendritic synapse strength between x_j and receiving neuron i. This equation expresses the accentuation or attenuation of x_j's effectiveness upon the receiving neuron when x_k is active. The axo-axonic synapses are constrained such that $a_{ijk} \geq -1$ for greater biological plausibility.

The axo-axonic model employed here accounts for all possible axo-dendritic and axo-axo-dendritic connections. The computation performed by a second-order I-input network can be expressed as:

$$O_i = \sum_{j=1}^{I}[w_{ij}x_j + d_{ij}(1+\sum_{k=1}^{I}a_{ijk}x_k)\,x_j] \tag{2}$$

Where the w_{ij} represent axo-dendritic connection weights having no axo-axonic influence. The d_{ij} represent axo-dendritic connection weights controlled by the axo-axonic connections a_{ijk}, $1 \leq i \leq N$. N and I are the total number of neurons and inputs respectively. All connections expressed by this equation are depicted by the two-input single-neuron pattern associator illustrated in Figure 1.

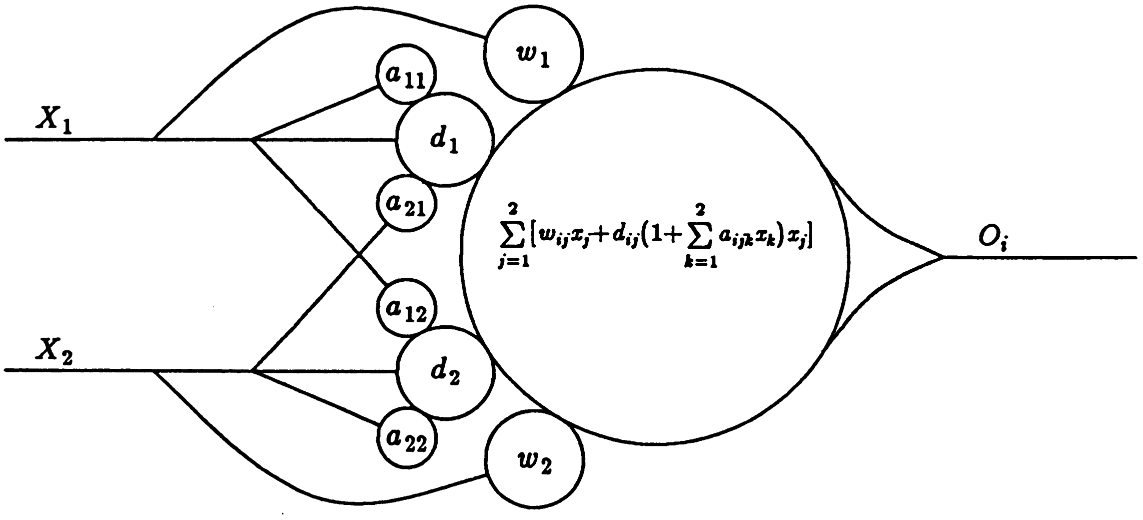


Figure 1. A two-input single-output axo-axonic interconnect model.

Equation (2) can be rewritten in the form of a second-order generalized polynomial by collapsing connection weight products into single variables:

$$O_i = \sum_{j=1}^{n} D_{ij}x_j + \sum_{j=1}^{n}\sum_{k=1}^{n} A_{ijk}x_kx_j \tag{3}$$

where $D_{ij}=w_{ij}+d_{ij}$ and $A_{ijk}=d_{ij}a_{ijk}$. For any combination of D_{ij} and A_{ijk}, $d_{ij}=A_{ijk}/a_{ijk}$ and $w_{ij}=D_{ij}\text{-}A_{ijk}/a_{ijk}$. This means there will always exist some combination of w,d and a for every possible choice of D_{ij} and A_{ijk}. It follows that equation (3) may be used in place of the earlier expression without loss of generality.

Adaptation algorithm

In this experiment, adaptation of axo-axonic connections was based upon a LMS procedure similar to that used for single-layer linear pattern associators. Weight changes are correlated to the error between trial output and target training data. A gradient descent in weight space is guided by a sum-square error measure. It can be shown that gradient descent is performed when the coefficients of equation (3) are adapted according to the following rules:

$$\Delta D_{ij} = -\eta(T_i-O_i)X_j \tag{4}$$

$$\Delta A_{ijk} = -\eta(T_i-O_i)X_jX_k \tag{5}$$

η is the learning rate, T_i represents the i^{th} element of an output training pattern, and O_i represents the network response to the corresponding input training pattern.

A simulation of a single neuron network described by (3) using these adaptation rules has successfully converged to XOR solutions from a variety of initial states. Hyperbolic and elliptic solutions typically result. Figure 2 shows convergence to a hyperbolic solution over a period of 50 epochs. Each line represents the decision boundary where $O_i=0.5$ and is labeled by the epoch number from which the weights were obtained.

The activation function has thus far been assumed to be linear. This assumption however, does not preclude the use of a nonlinear activation function. It has been found that linear activation is sufficient to learn the XOR association upon which current simulation data is based. It has yet to be established whether there exist problems for which a nonlinear activation would offer better results, although such problems are expected to exist.

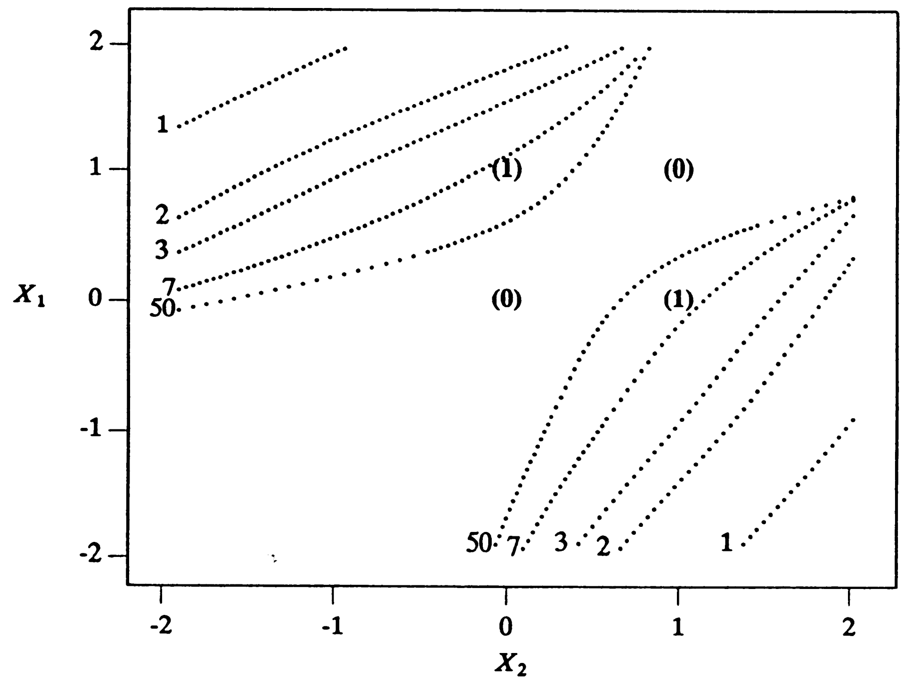


Figure 2. Convergence to a hyperbolic XOR solution.

Experimental Results

Figure 3 provides a tabular comparison of simulation results obtained using an established backpropagation simulator (BP) [6] and a 2nd-order axo-axonic simulation (AA). Both were configured as 2-input single-output networks. Mean convergence times (in epochs) and convergence probabilities (%) were observed. Each observation was based upon twenty simulations with the convergence probability given as the percentage of those trials which converged to a solution. The mean convergence time corresponds to the number of epochs required for convergence, averaged over the simulations which met convergence criteria. Solutions needing more than 4,000 epochs to converge to 0.01 total sum square error were recorded as nonconvergent. Simulations of both networks were performed with initial connection strengths uniformly distributed over [-1,1], patterns presented sequentially, zero momentum, and weight update by pattern. The networks were trained to perform a standard XOR classification to evaluate their relative performance with a quadratically-separable problem. The XOR problem is perhaps the simplest pattern association problem to require a nonlinear solution such as this.

Learning rate (η)	0.125	0.25	0.5	1.0	2.0
2-layer Back- propagation (BP)	* 0%	* 0%	2967 90%	1503 70%	830 85%
Single-layer Axo-Axonic (AA)	29.6 100%	13.1 100%	3.0 100%	* 0%	* 0%

Figure 3. Mean convergence time (epochs) and convergence probability (%) observed for various learning rates using a backpropagation and axo-axonic network. (*see text)

As the table shows, the convergence time for the axo-axonic simulation is significantly less than that of the backpropagation network. The best result obtained using the AA network requires two orders of magnitude fewer epochs than the fastest BP result. It has also been found to converge much more consistently for learning rates below some "stability threshold." While conducting these experiments, BP was observed to fail convergence many times by falling into a local error minimum. (BP convergences for $\eta \leq 0.25$ all occurred in greater than 4000 epochs, so are counted as nonconverging by the previous convergence definition.) AA was observed to not converge only when the learning rate (η) was chosen to be too large. With η too large, the system becomes unstable and connection weights tend to infinity. The stability threshold was found to occur somewhere between $\eta = 0.6$ and $\eta = 0.7$ for the XOR problem. With all observed learning rates below this threshold, AA was found to *always* converge. AA was *never* observed to become stuck in a local minimum.

These results are especially interesting when considering one relative advantage provided the BP program. The multilayer configuration virtually requires that activation biases be allowed to adapt -- fixing the biases at zero, for example, can cause BP to never converge to an XOR solution for a 0,1 pattern set. The BP XOR simulation uses five neurons and six connections arranged in two layers. AA on the other hand, is configured as a single neuron having six connections in a single layer. Activation biases for AA are fixed at zero offset, so the AA results are obtained with fewer degrees of adaptive freedom. Even with adaptive biases, AA would be computationally simpler, since it employs fewer neurons. At least for the XOR problem, the axo-axonic network exhibits better performance in terms of fewer epochs to converge, fewer computations per epoch, and more consistent convergence.

Future Directions

Much work remains to better understand the mechanisms by which axo-axonic connections lead to this level of performance, and to understand what the limitations of these mechanisms are. Preliminary indications from related experiments suggest that convergence improvements will also be observed for other problems having the same dimensionality. Experiments with higher dimensional networks and practical data are also planned.

Many other issues also need to be investigated. The utility of a nonlinear activation function needs to be better identified. The concept of higher-order axo-axonic networks needs to be developed. Higher-order networks could include the use of higher-order axo-axonic connections (axo-axo-axo-dendritic, for example), and the investigation of multi-layer extensions to this model. The generalization ability of networks based upon this model is also an open question suitable for future investigation.

Conclusion

A performance comparison between a multi-layer backpropagation network and a single-layer axo-axonic network has been presented. The results obtained show that (for the XOR problem) the axo-axonic interconnect model exhibits a higher convergence rate and better convergence probability than a simple backpropagation network . These results indicate that the axo-axonic interconnect model provides an interesting and potentially highly productive avenue of future investigation.

References

[1] Shepherd, G.M., **The Synaptic Organization of the Brain**, Oxford University Press, 1979.

[2] Hawkins, R.D., and Kandel, E.R., "Is There a Cell-Biological Alphabet for Simple Forms of Learning?" **Psychological Review**, Vol. 91 No. 3, pp. 375-391, 1984.

[3] Rumelhart, D.E, G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation," **Parallel Distributed Processing Vol. 1**, pp. 352-353, 1986.

[4] Durbin, R. and D.E. Rumelhart, "Product Units: A Computationally Powerful and Biologically Plausible Extension to Backpropagation Networks," **Neural Computation**, Vol. 1 No. 1, 1989.

[5] Meador, J.L., "Nonlinear Separation in a Single Neuron Pattern Associator Using an Axo-Axonic Interconnect Model," **IEEE Conf. on Neural Information Processing Systems**, 1989 (*In review*).

[6] McClelland, J.L., and D.E. Rumelhart, **Explorations in Parallel Distributed Processing**, pp. 121-159, 1988.

A PERCEPTRON BASED AUTO-ASSOCIATIVE MEMORY

A.MEKKAOUI, P.JESPERS
Microelectronics Laboratory
Place du Levent 3, B-1348 L.L.N. BELGIUM

1. INTRODUCTION:

The perceptron concept is the oldest and the most studied neural paradigm. After the pioneering work of Rosenblatt [1] this model has induced a lot of enthusiasm among researchers interested in building brain-like machines.

The power of the perceptron method lies in the existence of a convergence theorem by which one can determine the coefficients of the separating hyperplane equation whenever the classes are separable [2].

In this paper we propose a method to realize an auto-associative memory, where use is made of the perceptron theorem to determine the connection matrix **W**. The proposed method allows an arbitrary number of patterns to be fixed points. The issue to address then, is the size of attraction basins of these memory points which get smaller and smaller as more fixed points are added. When the memory is overloaded the size of the basins tends to zero but the reference prototypes are never washed out.

Throughout the paper the term convergence is used to indicate convergence to the correct expected state. Bold characters represent vectors or matrices. The terms exemplar and prototype, associative and auto-associative, are considered as synonymous. $\mathbf{x}^t$ denotes the transpose of vector **x**.

2. THE ASSOCIATIVE MEMORY PARADIGM:

A "neural" associative memory is usually implemented as a network of n mutually interconnected units(neurons). Each neuron receives n binary signals $(x_0, x_1, \ldots, x_{n-1})$ and emits a binary output signal by evaluating a non-linear function f of the weighted sum of its inputs. The state transition of each unit can be written as:

$$x_i(t+1) = f\left(\sum_{j=0}^{n-1} W_{ij} x_j(t)\right)$$

The system state-transition can be written as :

$$\mathbf{x}(t+1) = \mathbf{T}\mathbf{x}(t)$$

$\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})^t$ and T being a nonlinear operator defined by

$$\mathbf{T}\mathbf{x} = f(\mathbf{W}\mathbf{x})$$

$\mathbf{W} = \{W_{ij}\}$ denotes the connection matrix of the net. W_{ij} is the weight from unit j to unit i. Our aim is to find a matrix **W** such that for every pattern **x**, we want to store, the following is to hold:

$$\mathbf{x} = \mathbf{T}\mathbf{x}$$

An x for which the above equation holds is said to be an equilibrium state [4] or a fixed memory point [5] or more expressively a reference pattern [6]. Not only we want the **x** patterns to be fixed points but we want to achieve large attraction basins B(x) as well. Amari [4] defines B(x), the attraction basins of an equilibrium state **x**, as the set of those states each of which fall in the state **x** after N state transitions. Formally:

$$B(\mathbf{x}) = \{\mathbf{x0} \;/\; \mathbf{T}^N\mathbf{x0} = \mathbf{x}\} \quad N < \infty$$

In practice this translates to: Any reference pattern **x** can be associatively recalled from any vector **x0** belonging to the set B(x). It this property that makes an associative memory attractive for pattern recognition applications. The **x0** vector may represent a noisy input pattern to be recognized and **x** the non-distorted pattern to which the system converges after a finite number of transitions.

From the above one can infer that both the number of equilibrium states that can be achieved and the size of the attraction basins are important capacity criteria. The former is sometimes referred to as the absolute capacity criterion. A common measure of this capacity is the ratio m/n, m being the achievable number of fixed memory points.

Recent papers have reported different upper bounds of the m/n ratio. McEleice et-al [7] reported a figure of 1/(4logn) for the Hopfield network. Hopfield reported an experimental value of around 0.15 [8]. For the same network Amari [4] reports an upper bound of 1/(2logn-log(logn)).

3. A PERCEPTRON BASED ASSOCIATIVE MEMORY:

The proposed network is similar to the well known Hopfield one except that self-connection is allowed and weight symmetry is not imposed.

The key idea of this method is to consider each unit as a two-class machine receiving input from all other units and from itself through feedback. Each element in the network represents one bit in the input pattern, i.e,unit i represents bit i. Unit i (noted u_i) is assigned the job of separating input patterns into two classes. Class 1 contains patterns in which bit i is "1" and for which the u_i must be ON. On the other hand class 2 contains the patterns in which bit i is "0" and consequently u_i must be OFF. For example, in a four bit net u_1 will classify the pattern 1010 as belonging to class 1 while it will classify 0110 as class 2 pattern; on the other hand u_2 classifies the first pattern as being of class 2 and the second one as of class 1.

It can easily be shown that these two classes are linearly separable. And use is made of a perceptron convergence procedure to determine the weight matrix. The procedure is applied to each unit in turn. The W matrix is determined column after column.

One way to see that these 2 classes are linearly separable is to consider each pattern as located on a corner of an n-dimensional hypercube. It is obvious that the vectors having a "1" at the i^{th} position will be on the same side of the hypercube and the existence of a non-trivial separating hyperplane is obvious.

Determining the weight matrix by applying a perceptron procedure to each unit ensures That the network never fail to converge to a stored exemplar if it is presented undistorted. This is true even if all the 2^n possible patterns are stored. This is an important advantage over Hopfield associative memory where cross-talk is always present whenever the number of stored patterns exceeds a certain limit. The ratio m/n achieved by this method is $2^n/n$. This measure of capacity is to be taken with care because it does not give information on the size of the attraction basins of the equilibrium states. In fact the greater the number of stored patterns is the smaller the attraction basins of individual memory points are.

In order to achieve good attraction basins, the method has been improved by adding to each prototype its 1-bit distant (in Hamming sense) neighbors that are not prototypes themselves. Then an attempt is made to classify each pattern and its added neighbors as being the same exemplar. Assume, for example, that we have the two patterns 110 and 001 to store. After addition of the pattern neighbors we will have to classify 110, 010, 100 and 111 as being of the same category: u_1 and u_2 will then classify all these patterns as being of class 1 (110 has a 1 at its first and second position), while unit 3 will classify them as being of class 2.

This technique seems to be a kind of noise prediction. The considered neighbors will play the role of traps where distant input patterns are likely to go, during the relaxation process, before settling down on the right corner of the hypercube. Personnaz et-al have introduced a similar principle which consists in imposing intermediate state transitions as a way to tailor the size of the basins [9].

One might suspect that the added patterns may violate the linear separability condition. In the simulations a test is provided bywhich the added patterns causing such a problem are not considered. A possibly better solution is to use a procedure that converges when the set is linearly separable and is optimal when it is not [3],[10],[11]. Another problem that is likely to occur is the fact that two different exemplars may have the same neighbor. We have taken the decision to classify such a pattern with the exemplar that have the greatest binary magnitude. This has improved the degree of attractiveness considerably. In the next section the simulation results are presented.

4. SIMULATION AND RESULTS:

We have experimented this method by using the simple fixed increment algorithm [10] with a fixed threshold of 0. The units having boolean nonlinear transfer function. The net updates asynchronously and in a deterministic manner.

Figure 1 shows the obtained convergence results for a 25 bit network. In case of wrong convergence the output patterns are found to be close to the expected result. A plot of the % error versus noise is reported in figure 2.

Figure 3 shows the performance of this method as compared to the Hopfield net and to the simplex method described in [12]. This comparison considers a 12-bit net with 3 stored exemplars.

This method has been used for a practical pattern recognition problem. The input are 25-bit long binary vector representing digitized alphanumeric characters. After training, patterns corrupted by random noise were presented to the network for recognition. See Box 1. The results obtained are more than encouraging despite the extreme simplicity of the algorithm used to train each unit in the net.

Finally, we should mention that in these simulations we have never observed limit cycles nor a relaxation process that took more than six cycles.

5. CONCLUDING REMARKS:

We have proposed a simple idea to implement the auto-associative paradigm. We don't expect the idea to be used as such. Allowing variable threshold will certainly improve the performance. The application of the convergence procedure to each unit makes the learning process very slow, consequently the on line learning of this method is out of question. But it may not matter if the learning is done once per application.

REFERENCES:

[1] R. Rosenblatt, *Principles of Neurodynamics*, New York, Spartan Books (1959).
[2] M. Minsky, S. Papert, *Perceptrons: an introduction to computational geometry*, MIT press, (1988)
[3] B.Widrow, M.E Hoff, "Adaptive switching circuits", *1960 IRE WESCON Conv. Record, Part 4, 96-104, August 1960.*
[4]S.I Amari, "Statistical Neurodynamics of Associative Memory", *Neural Networks*, Vol 1, 63-73. (1988)
[5]H. Bourlard, C.I. Wellekens, "Neural Network Models and Application to Speech Processing", Lecture given at the Vrije Universiteit Brussells (VUB) April 1988.
[6]T. Kohonen, *Associative Memory : A system-theoretical Approach*, Springer-Verlag Berlin Heidlberg New-York 1977.
[7]McEleice,R.J, Posner, E.C, Rodmich, E.R, Venkatesh, S.S, "The Capacity of the Hopfield Associative Memory", *IEEE Trans. Info. Theory*, IT-33, 461-482.
[8] J.J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Properties " *proc. Natl. Acad. Sci.* USA, Vol. 79, 2554-2558, April 1982.
[9]Personnaz *et-al*, "Neural Networks for Associative Memory design", in *Computational Systems: Natural & Artificial*, Ed. H.Haken, springer 1988.
[10] R.O Duda, P.E Hart, *Pattern Classification and Scene Analysis*, John Wiley & Sons, New York (1973).
[11] J.T Tou, R.C Gonzales, *Pattern recognition principles*, Addison-Wesley co., Reading, Mass, (1974)
[12] M. Verleysen, P. Jespers, "Neural networks for high-storage CAM: VLSI circuit and learning algorithm." To be published in the IEEE JSSC, June 1989, Special Issue On ESSCIRC'88 Conference.

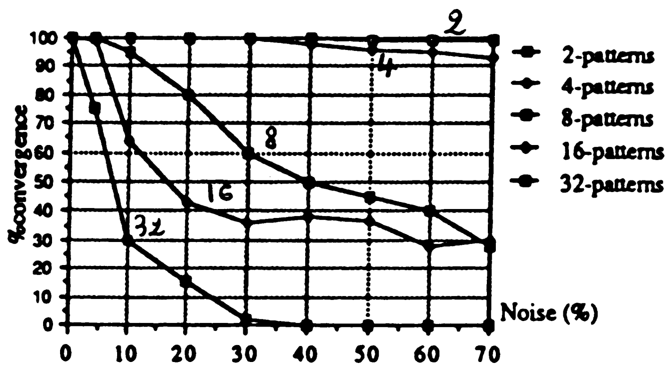


Figure 1. %convergence vs introduced noise.
Noise is expressed in terms of % bits randomly inverted.

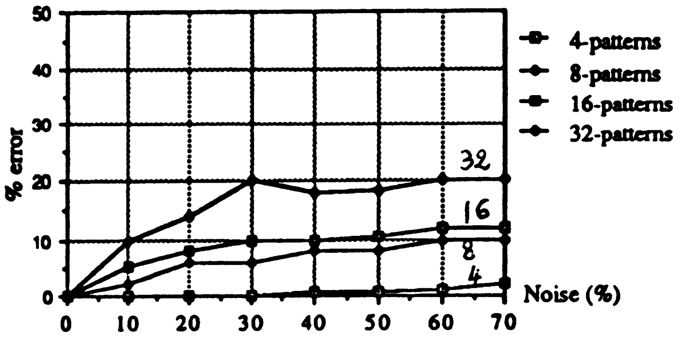


Figure 2. %error vs %noise introduced in the input patterns. Note the 32-patterns case, though the % convergence was poor, the %error remains low.

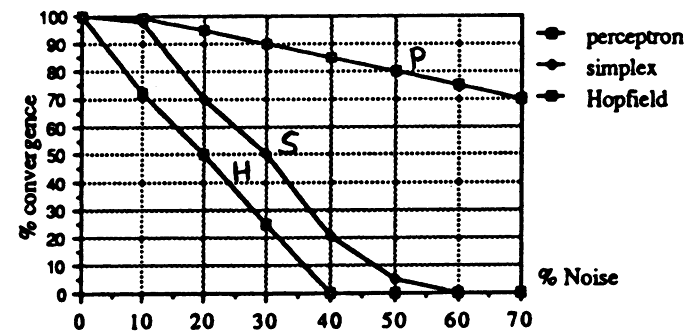
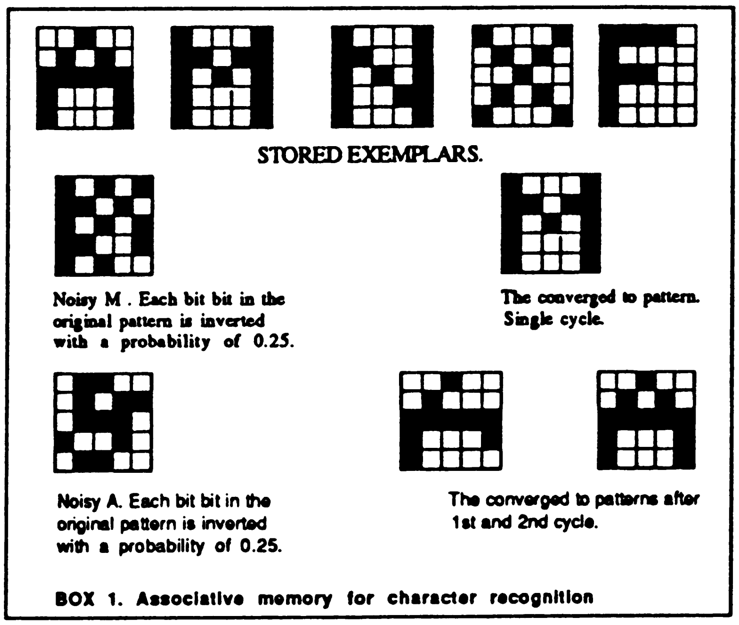


Figure 3. Comparison of Hopfield, simplex, and the proposed methods. Results for The Hopfield and the simplex methods are from [12].



STORED EXEMPLARS.

Noisy M . Each bit bit in the original pattern is inverted with a probability of 0.25.

The converged to pattern. Single cycle.

Noisy A. Each bit bit in the original pattern is inverted with a probability of 0.25.

The converged to patterns after 1st and 2nd cycle.

BOX 1. Associative memory for character recognition

Acceleration of Back-Propagation through Learning Rate and Momentum Adaptation.

Ali A. Minai & Ronald D. Williams
Center for Semicustom Integrated Systems
Department of Electrical Engineering
University of Virginia, Charlottesville, VA 22903

1. Introduction

The potential of highly parallel distributed systems for pattern recognition, control and artificial intelligence has long been recognized, but with the recent development of realistic training algorithms, artificial neural networks (ANN's) have really come into their own. Feed-forward ANN's have been intensively studied and models abound in the literature. By far the most popular training procedure for these networks, however, is the **back-propagation algorithm** [7]. Basically a gradient descent procedure, back-propagation has been quite successful, but there are several problems. The most important of these is slow speed. The research presented here is one of the many attempts at alleviating this difficulty.

2. Back-propagation Fundamentals

Consider a feed-forward network of neurons with transfer characteristic $o_i = f_i(net_i)$, where net_i is the net input to the *i*th neuron. Define an error metric E_p for an input/output pair p, and the overall error as

$$E = \sum_p E_p \tag{1}$$

Then, in order to minimize error, the weights need to be modified as

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} \tag{2}$$

where w_{ij} is the weight from the *j*th to the *i*th neuron, and η is a *learning-rate* parameter. The calculation of the partial derivative for multiple layers requires the backward propagation of error at the outputs --- hence the name back-propagation.

Since back-propagation uses only local gradient information, η must be kept very small to prevent jumps to undesirable areas of weight space. However, this slows the process down considerably, especially over long flat "plateaus" which, as Hecht-Nielsen [3] and others have suggested, abound in most ANN search spaces. The question, thus, is how to reconcile the opposing imperatives of stability and efficiency.

3. Speeding Up Back-Propagation: Methods and Previous Work

Basically, there are two ways to approach the problem of speeding up back-propagation: analytically incorporating more information about the error surface into the algorithm, or using heuristic methods that work most of the time. The former approach has had its most prominent success in *second order back-propagation* [6, 13], where second order gradient information is used to guide the search for a suitable minimum. These methods, while certainly useful, are difficult to implement and often produce little real-time speedup. This has led to the development of several "intuitive" heuristics that produce empirically acceptable results. These include attempts to damp local oscillations by the use of a *momentum* term [8], varying the learning rate [5, 1], modifying neuronal characteristics [12], trying line-search techniques [2], controlling the sequence in which training examples are presented [4], using different error metrics [11], controlling problem difficulty [10], and incorporating statistical information into the training procedure [9]. Of these, momentum has been so universally successful that it is now an integral part of almost all back-propagation networks. It modifies the basic learning equation to be:

$$\Delta w_{ij}(t) = -\eta \frac{\partial E}{\partial w_{ij}} + \mu \Delta w_{ij}(t-1) \tag{3}$$

where μ (momentum) is an inertial parameter that damps out local oscillations and provides additional acceleration [13].

4. Varying the Learning Rate: the Delta-Bar-Delta Algorithm

Jacobs, in [5], introduces a method, called the *delta-bar-delta (DBD) algorithm*, for varying the learning rate in response to local gradient information. The four basic aspects of this scheme are: 1) Each weight (i.e. each dimension of the search space) has its own learning rate. 2) These learning rates are varied based on error surface information. 3) When the error surface gradient $\partial E/\partial w_{ij}$ has the same sign for many iterations, the corresponding learning rate is increased, since this indicates that a minimum lies ahead. 4) When the error surface gradient flips signs for several consecutive time steps, the learning rate is decreased, since this indicates that a minimum is being jumped over. Based on these heuristics, the scheme for modifying learning rates is:

$$\eta_{ij}(t+1) = \eta_{ij}(t) + \Delta\eta_{ij}(t) \tag{4}$$

$$\Delta\eta_{ij}(t) = \begin{cases} \kappa & \text{if } \bar{\delta}_{ij}(t-1)\delta_{ij}(t)>0 \\ -\phi\eta_{ij}(t) & \text{if} \bar{\delta}_{ij}(t-1)\delta_{ij}(t)<0 \\ 0 & \textit{otherwise} \end{cases} \tag{5}$$

$$\delta_{ij}(t) = \frac{\partial E(t)}{\partial w_{ij}(t)} \tag{6}$$

$$\bar{\delta}_{ij}(t) = (1-\theta)\delta_{ij}(t) + \theta\bar{\delta}_{ij}(t-1) \tag{7}$$

κ, ϕ and θ are parameters specified by the user. The quantity $\bar{\delta}(t)$ is basically an exponentially decaying trace of gradient values. The reason for increasing learning rates additively is to prevent them from becoming too large too rapidly, while that for the exponential decrementing is to keep the rates positive at all times, and to allow rapid decrease [5].

5. Problems with the Delta-Bar-Delta Algorithm

The DBD algorithm was tried on several problems, and was found to be excellent for increasing learning speed. However, some serious limitations were noticed. The first is that the algorithm does not use momentum, even though results cited by the author indicate that momentum *does* give further speedup [5]. The reason is that momentum, along with a varying learning rate, can sometimes cause the search to diverge drastically. One possible solution is to keep the κ factor very small, but this leads to slow increase in η and little speedup.

Another related problem is that, even with a small κ, the learning rate can sometimes increase so much that the small exponential decrease is not sufficient to prevent wild jumps. Increasing ϕ exacerbates the problem instead of solving it because it makes return from bad search-space areas difficult. Making ϕ small, on the other hand, lets η get out of hand. The heuristic algorithm presented here tries to overcome these problems to a certain extent.

6. The Extended Delta-Bar-Delta Algorithm (EDBD)

Several changes were made to the DBD algorithm to make it more robust and efficient. These are as follows:

(1) The learning rate increase was made an exponentially decreasing function of $|\bar{\delta}(t)|$ instead of being a constant κ. This meant that learning rate would increase faster on very flat areas but slower on areas of greater slope. Standard DBD overlooked this, increasing η by a constant even when the slope was quite steep. The new scheme allowed the use of higher increment factors where it really mattered.

(2) Momentum was used as a standard part of the algorithm, but was varied just like the learning rate. Thus, momentum was increased on plateaus and decreased exponentially near minima. The DBD criterion was used for this purpose too, but the increment factor was again a decreasing exponential function of $|\bar{\delta}(t)|$.

(3) To prevent either the learning rate or momentum from becoming too high, a *ceiling* was defined for both at which they were hard-limited. This further facilitated the use of large increments, since their effect was not unbounded.

(4) Memory and recovery were incorporated into the algorithm. Thus, the best result seen until the current time step was saved. A tolerance parameter λ was used to control recovery. If the error became greater than λ times the lowest seen so far, the search was restarted at the best point with attenuated learning rate and momentum. To prevent thrashing, this was done stochastically, so there was a small probability that the search would restart at a totally new point.

The equations for the EDBD algorithm can be written as follows:

$$\Delta w_{ij}(t) = -\eta_{ij}(t)\frac{\partial E}{\partial w_{ij}(t)} + \mu_{ij}(t)\Delta w_{ij}(t-1) \tag{8}$$

$$\eta_{ij}(t+1) = MIN[\eta_{\max}\ ,\ \eta_{ij}(t) + \Delta\eta_{ij}(t)] \tag{9}$$

$$\mu_{ij}(t+1) = MIN[\mu_{\max}\ ,\ \mu_{ij}(t) + \Delta\mu_{ij}(t)] \tag{10}$$

$$\Delta\eta_{ij}(t) = \begin{cases} \kappa_l \exp(-\gamma_l \,|\bar{\delta}_{ij}(t)|) \\ -\phi_l \eta_{ij}(t) \\ 0 \end{cases} \quad \Delta\mu_{ij}(t) = \begin{cases} \kappa_m \exp(-\gamma_m \,|\bar{\delta}_{ij}(t)|) & \text{if } \bar{\delta}_{ij}(t-1)\delta_{ij}(t)>0 \\ -\phi_m \mu_{ij}(t) & \text{if} \bar{\delta}_{ij}(t-1)\delta_{ij}(t)<0 \\ 0 & \textit{otherwise} \end{cases} \tag{11}$$

where $\delta_{ij}(t)$ and $\bar{\delta}_{ij}(t)$ are as in equations (6) and (7). $\kappa_l, \phi_l, \gamma_l, \eta_{\max}, \kappa_m, \phi_m, \gamma_m$ and $\mu_{\max}$ are specified by the user. Note that EDBD reduces effectively to DBD with appropriate parameter settings.

7. Results

The EDBD algorithm was tested on two problems: a simple binary one (Exclusive-OR), and a more complex real-valued one (Quadratic Function). The algorithms are characterized by their parameters as follows:

SDM: η, μ ;   DBD: $\eta_i, \mu, \kappa, \phi, \theta$
EDBD: $\eta_i, \eta_{\max}, \mu_{\max}, \kappa_l, \phi_l, \gamma_l, \kappa_m, \phi_m, \gamma_m, \lambda, \theta$

where η_i is the initial value of η, and SDM is the standard back-propagation algorithm with momentum, included here for comparison. All algorithms were run in a "batched" fashion, each epoch representing a complete pass through the training set. Weights were modified at the end of each epoch. All hidden neurons had sigmoid transfer functions. κ_m=0.1,ϕ_m=0.3 and γ_m=1.0 were used for all experiments. Each graph represents data averaged over 20 independent runs *including failures*. For each individual run, all algorithms started at the same point.

7.1. The Exclusive-OR Problem (XOR)

A network with one 2 unit hidden layer and a linear threshold output unit was used for this problem. The settings were $\kappa=\kappa_l=0.095$ and $\phi=\phi_l=0.1$. These were taken from [5]. The DBD algorithm was tried both with (μ=0.9) and without momentum. In the former case, 10 out of 20 runs failed to converge in 2000 epochs, while all converged in the latter. EDBD performed better than both using $\eta_{\max}$=10.0, $\mu_{\max}$=0.9, and all runs converged to 0 error (figures 1 & 2). The "Performance" figures give **#converged/#did_not_converge.** All figures are for 2000 epochs. The recovery feature was turned off in this case by setting λ very high.

7.2. The Quadratic Function Problem (QUAD)

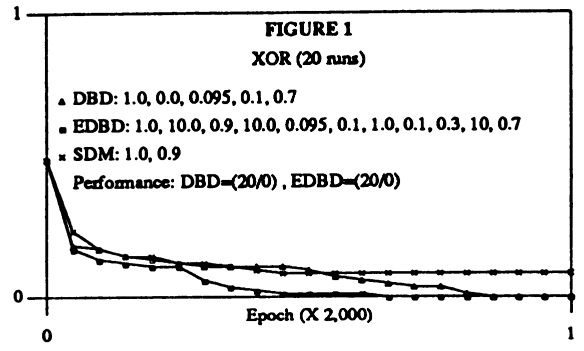
Here, a network with 2 hidden layers, of 2 and 1 units respectively, and one linear output neuron was trained to approximate the quadratic function

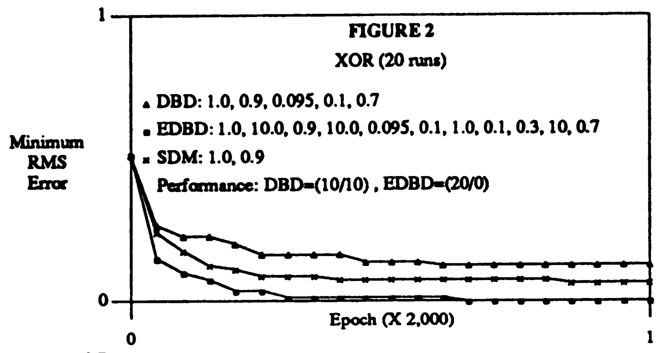
$$y = Px(1-x)\ ; \qquad P = 3.95,\ 0.0 < x < 1.0 \tag{12}$$

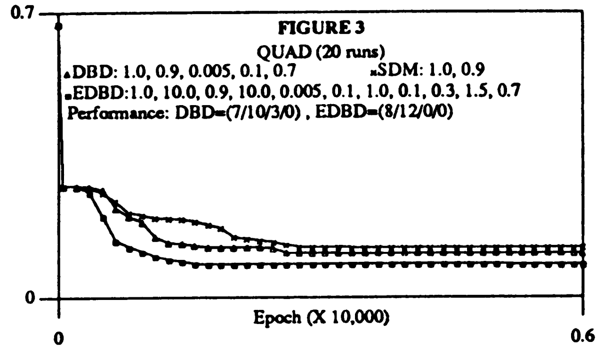
given 20 data points. Unlike binary problems, real-valued ones like this have error-spaces that are *extremely* sensitive to minor weight changes, and require deft parameter manipulation. Several parameter values were tried for both DBD and EDBD, with the latter producing clearly superior results. Indeed, it was found that DBD was able to operate well only in a very narrow range of κ and ϕ. The upper limit on κ was about 0.1, beyond which the algorithm either got trapped on a plateau or diverged to extremely high errors, depending on ϕ. EDBD, on the other hand, proved *much* more robust, operating better than the best of DBD with κ as high as 1.0, thus making parameter settings far less critical (see figure 6). No EDBD runs diverged, though descent did become more erratic with increasing κ. Sometimes the search got stuck on a plateau at an error of 0.13, though still moving downhill. Longer simulations give reason to believe that these latter situations are not permanent, and do lead to good solutions, albeit slowly. Some of the results from this experiment are shown in figures 3 to 6. The "Performance" figures represent: **#succeeded/#stuck_at_0.13/#stuck_at_0.27/#diverged,** since two very large plateaus exist at error = 0.27 and 0.13. A solution with error less than 0.02 is counted as a success. All figures are for 10000 epochs. Tests with novel data points showed excellent generalization for both algorithms.

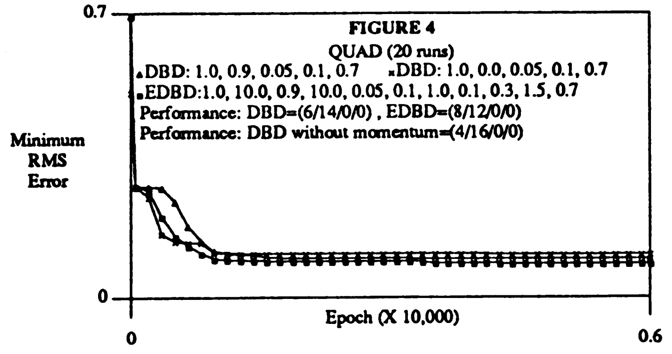
References

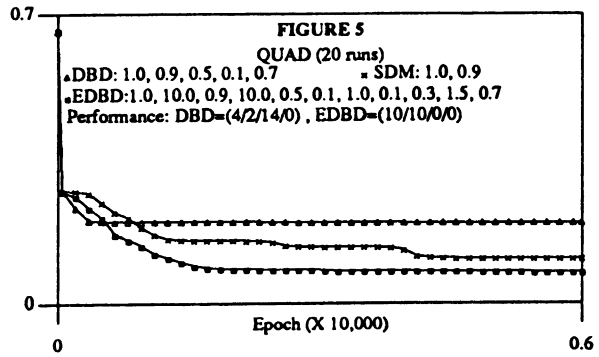
1. J.P. Cater, *Proc. of the 1st ICNN*, vol. II, pp. 645-651, 1987.
2. E.D. Dahl, *Proc. of the 1st ICNN*, vol. II, pp. 523-530, 1987.
3. R. Hecht-Nielsen, *Proc. of the 1989 IJCNN*, vol. I, pp. 593-605, 1989 .
4. J.C. Hoskins, *MCC Tech. Rep. # STP-049-89*, January 1989.
5. R.A. Jacobs, *Neural Networks*, vol. 1, pp. 295-307, 1988.
6. D.B. Parker, *Proc. of the 1st ICNN*, vol. II, pp. 593-600, 1987.
7. D.E. Rumelhart, G.E. Hinton, and R.J. Williams, in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, ed. D.E. Rumelhart & J.L. McClelland, vol. 1, pp. 318-362, 1986.
8. J.L. McClelland & D.E. Rumelhart (eds.), *Explorations in Parallel Distributed Processing: Models, Programs and Exercises*, MIT Press, 1988.
9. T. Samad, *Neural Networks*, vol. 1, Supp. 1, p. 216, 1988.
10. F.J. Smieja and G.D. Richards, *Complex Systems*, vol. 2, pp. 671-704, 1988.
11. S.A. Solla, E. Levin, and M. Fleisher, *Complex Systems*, vol. 2, pp. 625-640, 1988.
12. W.S. Stornetta and B.A. Huberman, *Proc. of the 1st ICNN*, vol. II, pp. 637-643, 1987.
13. R.L. Watrous, *Proc. of the 1st ICNN*, vol. II, pp. 619-627, 1987.

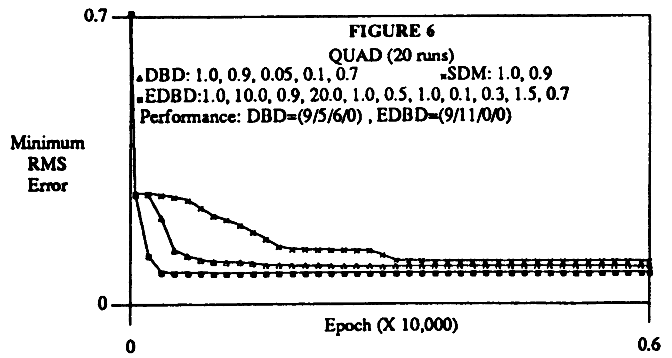












BACKPROPAGATION LEARNING WITH HIGH-ORDER FUNCTIONAL NETWORKS AND ANALYSES OF ITS INTERNAL REPRESENTATION

Akira Namatame
Department of Computer Science
National Defense Academy
Yokosuka, 239 JAPAN

ABSTRACT: We obtain the necessary conditions of the networks architecture for learning the structures of continuous mappings. We propose a new network architecture, high-order functional networks with some nonmonotonic functionals as input units. It is shown that high-order functional networks trained with backpropagation can generalize and infer the highly nonlinear structures of continuous mappings. Nonlinear mappings can be characterized by the features of their extremums and curvatures. It is shown the combination of the input units with nonmonotonic functionals and the hidden units makes it possible to realize a proper internal representation for extracting those features.

1 INTRODUCTION

It has been widely recognized that inductive inference capability or generalization is a fundamental issue for machine learning, and there has been a great deal of theoretical and experimental work done on inductive learning in symbolic AI [6]. A connectionist learning may provide an alternative method for inductive learning and provide a number of important properties not found in symbolic AI. The process of inferring general regularities requires the construction of complex mappings, which utilizes internal representation, from a high-dimensional input space to a lower dimensional output space. We choose a set of continuous mappings as a task domain for learning. Depending on how a continuous mapping to be encoded in the network, some architecture models can be utilized. It is shown that conventional models are inadequate to represent the continuous nonlinear mapping. We propose a new network architecture and show it can generalize a highly nonlinear structural relation defining a given continuous mapping.We also investigate the internal representation capability of a new network architecture.

2 REPRESENTATION & GENERALIZATION OF CONNECTIONIST MODELS

The internal representation of a multi-layer network is believed to capture high-order correlations which embody the invariant relationships between the input data and the output data. However, those innate properties can be only realized only if the representation model and the corresponding network architecture that fit the task domain for learning are appropriately designed [6]. As the task domain for learning, we choose a set of continuous mappings defined $D=\{(x,y);\ x \varepsilon K,\ y= F(x)\}$, where K is a compact set of R. We assume the function F(x) is a class of C^2 with many extremums. Nonlinear mappings can be characterized by the number and position of their extremum and the curvature.We denote the set of those extrmums by G that contains S elements.We consider the following specific functions defined on K=[0, 1] as the test cases:

Case 1; $F(x)= 0.5 + 0.3\sin 10\pi x,$ (2-1)

Case 2; $F(x)=(1+4x-4x^2)(1.1+\cos 4\pi x)(1.1+\sin 3\pi x)$ (2-2)

3 NECESSARY CONDITIONS OF AN NETWORK ARCHITECTURE

It is known that any continuous function can be approximated to arbitrary precision by a three-layer neural network [3]. However, there is no systematic method how such a network should be constructed. A basic and commonly used model of representing continuous pattern in a neural network is to encode input and output variables by continuous input and output units as shown in Fig.1a [5] and we term this network architecture as a first-order network. We denote the output from each hidden unit (that is input to the output unit of the network) by $f_i(x)$, $i=1,2,\ldots,k$. The network output function $f(x)$ is then given as $f(x)= 1/[\ 1+ \exp\{-\ \Sigma w_i f_i(x) + \theta_o\}]$. We now obtain the necessary conditions for the forms of each $f_i(x)$ in order to learn the structure of $F(x)$.

Lemma 3.1: In order to learn the structure of the function $F(x)$ with a multi-layer network, it is necessary that each $f_i(k)$ satisfies the following conditions:

(i) For any $x \varepsilon K$: $\quad \mathrm{sgn}\{F'(x)\} = \mathrm{sgn}\{\Sigma w_i f'_i(x)\}$ (3-1)

(ii) For any extremum $x_s \varepsilon G$ of $F(x)$: $\Sigma w_i f'_i(x_s)=0$ (3-2)

Proof: If $f(x)$,the output function of the multi-layer network, approximates $F(x)$, its derivative should be close to that of $F(x)$. The derivative of $f(x)$ is given as $f'(x)= f(x)\{1-f(x)\}\Sigma w_i f'_i(x)$.Since for any $x\varepsilon K, 0<f(x)\{1-f(x)\}\leq 1/4$,we have $\mathrm{sgn}\{F'(x)\}=\mathrm{sgn}\{f'(x)\}= \mathrm{sgn}\{\Sigma w_i f'_i(x)\}$. At the extremum x_s of $F(x)$, we should have $\Sigma w_i f'_i(x_s)=0$.

The derivatives of $f_i(x)$ can be assumed to be linearly independent functions. Therefore,a necessary and sufficient condition for (3-2) with $S>k$ (i.e., the number of the hidden units is less than that of the extremum) is for any $x_s \varepsilon K$ satisfying $F'(x_s)=0$

$$f'_i(x_s)=0 \qquad (3\text{-}3)$$

These results imply that it is necessary for the network to acquire the proper structures of the hidden units satisfying the conditions in (3-1) and (3-3). That is, each output function of the hidden units should be nonmonotonic and the value of its derivative evaluated at each extremum of the function to be learned should be zero. The limitation of a first-order network architecture stems from the linear input to each hidden unit resulting in a monotonic output function of each hidden unit.

4 LEARNING WITH AN ARCHITECTURE WITH HIGH-ORDER FUNCTIONAL NETWORKS

In order to avoid the limitation of the monotonic transformation of a first-order network, we should implement some nonmonotonic functions as the inputs to the network. We consider a three-layer network with some nonmonotonic functional units $\{g_1(x),g_2(x),\ldots,g_n(x)\}$ as input units as shown in Fig.2, termed as a high-order functional network.The advantage of a high-order functional network will be only realized only if how such nonmonotonic functional units should be selected that may fit to a given task domain. We choose a set of Chebychev polynomials. The i-th order Chebychev polynomial $g_i(x)$ defined on $0\leq x \leq 1$ is given as

$$g_i(x) = [\ \cos\{\ i\ \arccos(2x-1)\} + 1]/2. \qquad (4\text{-}1)$$

Such Chebychev polynomials are given as $g_1(x)=x, g_2(x)=(2x-1)^2, g_3(x)=x(4x-3)^2$.

In order to evaluate the learning capability of a high-order functional network, we constructed the three-layer network with Chebychev polynomials input units up to the fifth order and four hidden units for (2-1) and up to seventh order Chebychev polynomials input units and four hidden units for (2-2). We trained each network with the arbitrary chosen training examples (120 training examples for (2-1) and 50 training examples for (2-2).) The training cycle for each case was 2,000. The network outputs after training when presented any value x in [0,1] with the interval of 1/500 are shown in Fig.3. This result demonstrates the representation and generalization capability of the high-order functional network. The high-order network learns and infers highly nonlinear structures of the continuous mappings with a small set of the training examples. It is a relatively long history of supervised learning of implementing high-order terms as input units [2][4].The two-layer network, termed as a high-order flat network, as shown in Fig.1(b) can eliminate the need of hidden units and provide extremely rapid learning. We trained a three layer first-order network with 15 hidden units and a high-order flat network with 7 high-order input units for Case 2 in (2-2).(The training cycle was 10,000 for each case.) The network outputs after training are shown in Fig.4(a)(b). These results show the networks with the traditional architectures that do not satisfy the conditions of Lemma 3.1 can no longer generalize the structure of a complex continuous mapping. Especially, the traditional approach of high-order flat networks puts less emphasis on the utility of the internal representation of the networks.

5 ANLYSES OF INTERNAL REPRESENTATION CAPABILITY

In this section, we show a high-order functional network provides excellent generalization by specializing the selection of the structures of hidden units. If the output function of each hidden unit is in forms of generalized step functions defined by

$$f_i(x) = \phi_{sj}(x) \quad x \,\varepsilon\, N(x_{sj}) \qquad (5\text{-}1)$$
$$= \{0, 1\} \quad \text{Otherwise}$$

where $\phi_{sj}(x)$ is defined over the neighborhoods $N(x_{sj})$ of some extremum x_{sj} of the learning function $F(x)$ and satisfies $\phi'_{sj}(x_{sj})=0$. Then those functions satisfy the conditions of Lemma 3.1. The output function of each hidden unit of the high-order functional networks that were trained with the training examples of (2-1) and (2-2) are shown in Fig.6(a)(b). The high-order functional network realizes the internal network representation by acquiring the proper structures in the forms of (5-1). The roles of the hidden units and nonmonotonic functional input units are characterized by capturing the features of the number and the position of the extremum of the learning function. The role of the nonmonotonic input units is also characterized to capture the curvatures of the task function. Let d be the minimum absolute value of the radius of the curvature at the extremum of the learning function F(x),i.e., $1/d^*=\text{Max } F''(x_S)$. At the extremum x_s of $F(x)$, we have $F''(x_s)=f''(x_s)=f(x_s)\{1-f(x_s)\}\Sigma w_i g''_i(x_s)$. Therefore, the curvature of the network function $f(x)$ at its extremum is essentially determined by the curvature of $g_n(x)$,the nonmonotonic input unit with the highest order. The absolute value of the curvature of Chebychev polynomial with the n-th order at its extremum is given by $g''_n(x)=n^2/2x(1-x)$, and the required order of the nonmonotonic functional input unit n^* can be roughly estimated by

$$n^* = (d^*/2)^{1/2} \qquad (5\text{-}2)$$

Each hidden unit receiving the functional inputs up to the n-th order from the input units can represent roughly (n-1)/2 extremums of the learning function. Therefore the number of the required hidden units k^* is also estimated as

$$k^* = 2S/(n^*-1) \qquad (5\text{-}3)$$

6 CONCLUSION

We showed the first-order multi-layer networks and the high-order flat networks without hidden units that are commonly used in the previous researches are inadequate to generalize and to learn the nonlinear structures underlying the continuous mappings. We proposed the new network architecture,high-order functional networks,with some nonmonotonic functionals as input units. It was shown the three-layer networks with some Chebychev polynomials as input units can learn and generalize complex nonlinear structures of nonlinear continuous mappings. The internal representation capability of a high-order functional network was analyzed. It was shown the combination of some nonmonotonic input units and some monotonic hidden units provides the excellent generalization by specializing the selection of the internal representation to match the specific task domain.

REFERENCES

[1] Ahmad,S.,and Tesauro,G.,"Scaling and generalization in neural network: A case study",Proc.of the 1988 Connectionist Models,pp3-10, Morgann Kaufman (1988)
[2] Giles,C.L.,and Maxwell.T,"Learning,invariance and generalization in high-order neural networks", Applied Optics,Vol.26, pp4972-4978,(1987)

[3] Irie,B.,and Miyake,S.,"Capability of three-layered perceptron",IEEE International Conference on Neural networks, Vol.1, pp509-513,(1988)
[4] Klassen,T.,Pao,Y.H.,and Chen,V,"Characteristics of the functional link nets",IEEE Int. Conf. on Neural Networks,Vol.1,pp509-513,(1988)
[5] Lippmann,R.P.,Beckman,P.," Adaptive neural nets preprocessing for signal detection in non-Gaussian" in Advances in Neural Information Processing Systems 1,(ed.Tourretzky), Morgan Kaufmann, pp124-132 (1989)
[6] Namatame,A.,and Kimata,Y.,"Improving generality of a backpropagation learning system" International Journal of Neural Networks (1989) (to appear)

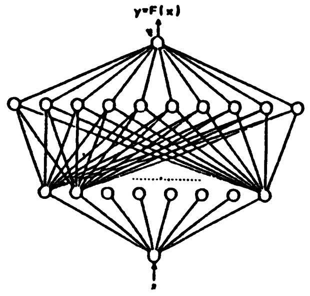


Fig.1(a) An architecture of first-order networks

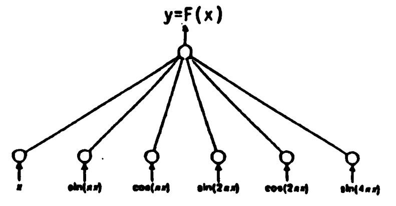


Fig.1(b) An architecture of high-order flat networks

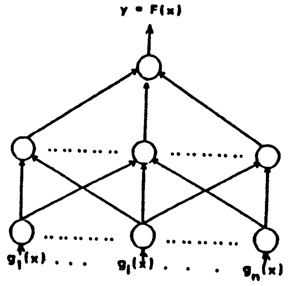


Fig.2 An architecture of high-order functional networks

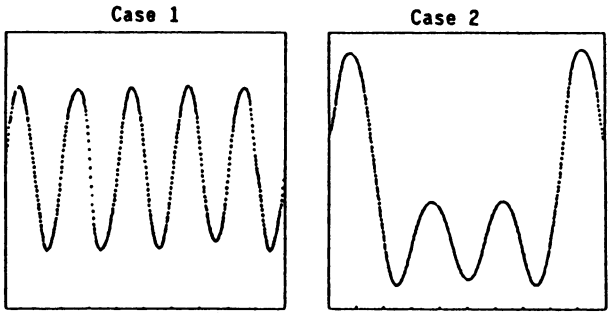


Fig.3 Learning achieved with high-order functional networks

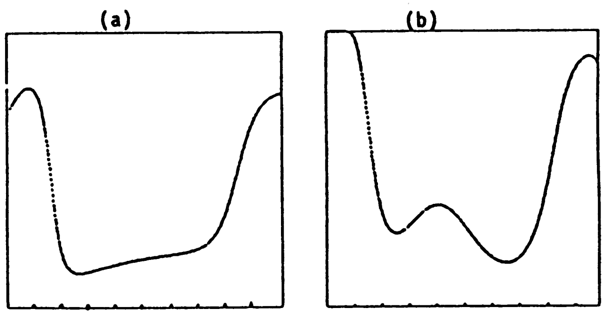


Fig.4 Learning achieved with first-order networks(a) and high-order flat networks (b)

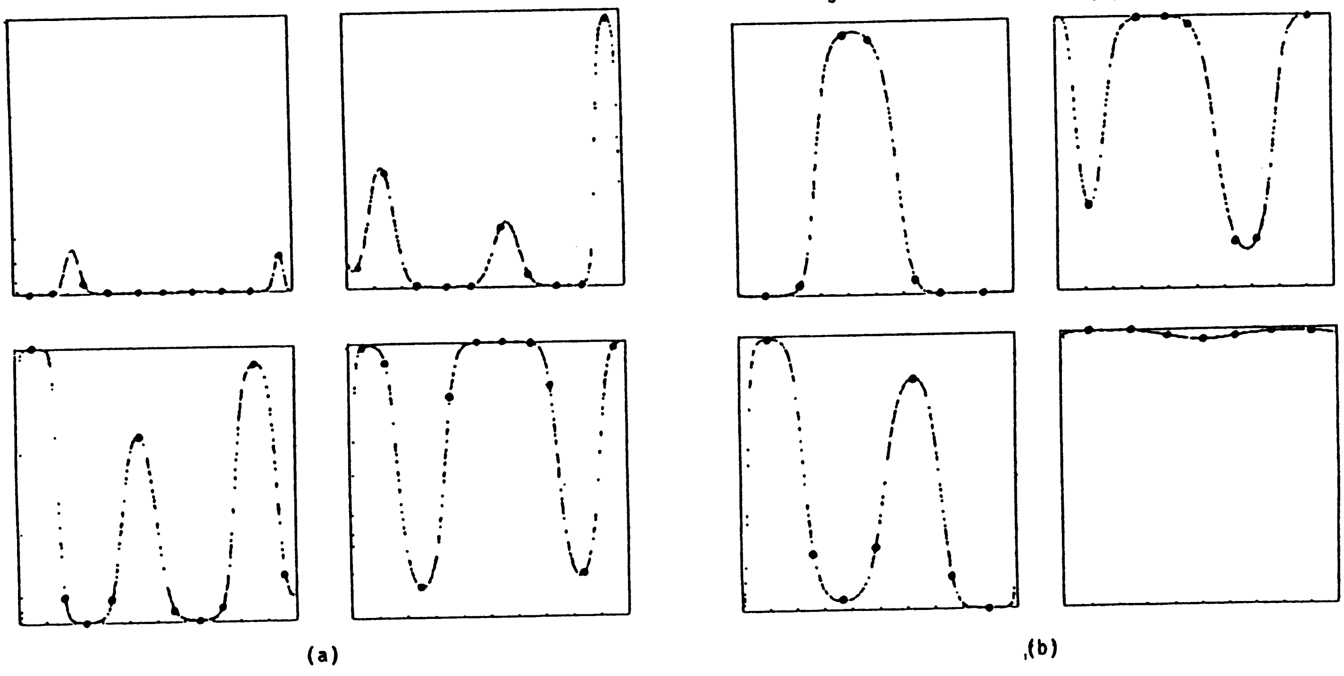


Fig.5 The depction of the output of each hidden unit after learning: (a) Case 1 (b)Case 2 (The dots denote the extremums of the function to be learned)

NN/I: A Neural Network Which Divides and Learns Environments

Yoshikazu NISHIKAWA, Hajime KITA, Akinori KAWAMURA
Dept. of Electrical Engineering, Kyoto University
Yoshida-Honmach, Sakyo, Kyoto 606, Japan

Abstract: The error back-propagation rule, though it is quite popular nowadays, has some drawbacks in practical applications. In order to get rid of them, this paper proposes a new sort of neural network called "NN/I". NN/I consists of two different types of network, i.e., a two-layer control network of Kohonen type and a three-layer main network which contains several subnetworks of back-propagation type. The control network learns to categorize the input patterns roughly, and acquires an ability to select one of the subnetworks in response to an input pattern. Then the selected subnetwork works to recognize and classify the input pattern in detail. Ability and features of NN/I are evaluated through several computer simulations.

1. INTRODUCTION

The error back-propagation (BP) algorithm [1] is one of the powerful learning algorithms for layered neural networks, and many applications have been reported [2,3]. While the BP algorithm shows excellent performance in dealing with small problems [4], it faces some difficulties in applications to the large problems in the real world:

1) *Speed of Learning*: In the BP algorithm, it often takes much computation time to learn a task, and sometimes it fails to find the global optimal connections. When a complex task is tried by use of a large network, this factor restricts the actual computation.
2) *Robustness of Memory:* Suppose that a set of training tasks is to be learned by a BP network, and those tasks are learned one by one in some order. The BP algorithm works to modify the network connections for achieving only the present task without regard to their usefulness in the previous tasks. In other words, the network is not robust in the sense that it forgets the old tasks, if presentation of the training data to the networks is not made appropriately.
3) *Information Representation on Hidden Layers:* In the BP algorithm, information representations on the hidden layers are formed only to achieve a given task. They sometimes become too dispersive, and make analysis of the network behavior difficult.

In the present paper, a neural network NN/I and its learning algorithm are proposed to resolve these difficulties to some extent. In the succeeding section, the structure of NN/I and its recalling/learning processes are described. In Section 3, some computer simulations are presented to show the performance of NN/I.

2. A NEURAL NETWORK, NN/I

The basic strategy of the proposed network NN/I is to manage hard tasks by 'division' of environments. Namely, a large task of recognition is divided into several subtasks, and each subtask is learned and recalled by a relatively small subnetwork. It makes learning easy and quick, and at the same time it makes memory of the network robust. Ability for both division of the task and assignment of the subtasks to the subnetworks is attained adaptively by means of a self-organizing mechanism of the network.

2.1. Structure of NN/I

NN/I consists of a three-layer main network of the BP type and a two-layer control network of Kohonen type. The main network has n_I input units, and n_O output units, and its hidden layer is divided into n_S groups each containing n_H hidden units. A partial network consisting of the input layer, the output layer and one group of the hidden units is called a 'subnetwork'.

The control network shares the input layer with the main network, and its output layer consists of n_S neurons each corresponding to a subnetwork. The output unit of the control network is called the 'control unit'. Figure 1 shows the structure of NN/I.

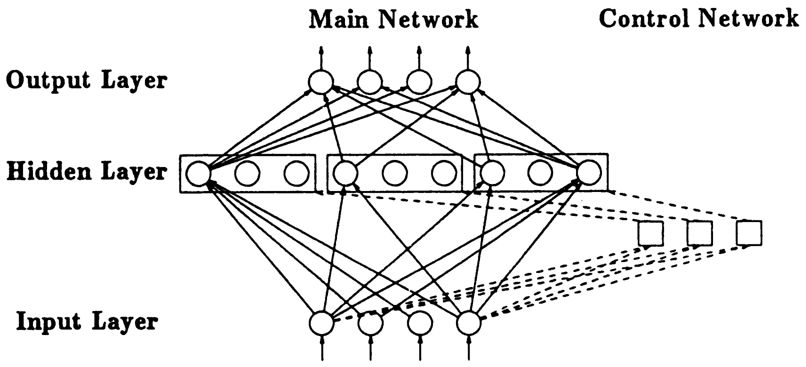


Fig. 1 Structure of NN/I.

2.2. Recalling Process

When an input pattern (n_I vector) $\boldsymbol{p}$ is presented to the input layer of NN/I, only the c-th control unit having the most similar connection weights (n_I vector) $\boldsymbol{w}_{C,c}$ to $\boldsymbol{p}$ becomes active, i.e.,

$$y_{C,c} = 1 \quad \text{for } c \text{ such that } ||\boldsymbol{w}_{C,c} - \boldsymbol{p}|| = \min_l ||\boldsymbol{w}_{C,l} - \boldsymbol{p}||$$

$$y_{C,l} = 0 \quad \text{for } l \neq c,\ 1 \leq l \leq n_s \tag{1}$$

where $y_{C,l}$ denotes the output of the l-th control unit.

The activated control unit permits the corresponding subnetwork to work, and then only the selected subnetwork works to generate the outputs as follows:

On the hidden layer

$$y^c_{H,j} = f((\boldsymbol{w}^c_{H,j})^T \boldsymbol{p} + \theta^c_{H,k}), \quad 1 \leq j \leq n_H \tag{2}$$

On the output layer

$$y_{O,k} = f(\boldsymbol{w}^{cT}_{O,k} \boldsymbol{y}_H + \theta^c_{O,k}), \quad 1 \leq k \leq n_O \tag{3}$$

where $y^i_{H,j}$, $w^i_{H,j}$ and $\theta^i_{H,j}$ denote the output, connection weights and bias of the j-th hidden unit belonging to the i-th subnetwork, respectively, and
$y_H = (y^1_{H,1}, \cdots, y^1_{H,n_H}, \ldots, y^{n_s}_{H,n_H})^T$. Note that $y^i_{H,j} = 0$ for all $i \neq c$, $1 \leq i \leq n_s$. Further, $y_{O,k}$ and $w_{O,k}$ denote the output and connection weights of the k-th output unit, respectively, and $\theta^c_{O,k}$ denotes the bias of the k-th output unit when the c-th subnetwork is activated. f denotes the sigmoid function $1/(1 + \exp(x))$.

2.3. Learning Process

In NN/I, the control network learns a way for dividing the input patterns by means of Kohonen's algorithm [5], that is, the connection weights are modified as follows:

$$\begin{aligned} w_{C,l}(t+1) &= w_{C,l} + \beta(t)(p(t) - w_{C,l}(t)), \quad \text{for } l \in N_c(t) \\ w_{C,l}(t+1) &= w_{C,l} \quad \text{for } l \notin N_c(t) \end{aligned} \tag{4}$$

where $t = 1, 2, \cdots$ denotes discrete time. $\beta(t)$ is a positive parameter (a learning rate), and $N_c(t)$ denotes a neighborhood of the c-th control unit. Both $\beta(t)$ and $N_c(t)$ may be changed with the time t.

The main network, on the other hand, learns the input/output relations by means of the error back-propagation algorithm. Modification of the connections is made only when the corresponding subnetwork is selected (activated) by the control unit. It should be noted that each subnetworks must memorize its error signals at its latest activation in order to introduce a momentum term for accelation of the learning.

3. COMPUTER SIMULATION

First, computer simulations to test the learning speed are carried out. NN/Is having 10, 25, 50, 150 and 250 hidden units are investigated, together with conventional three-layer BP networks having the same numbers of hidden units for comparison. All the networks are equipped with 10 input units and 5 output units. Four tasks containing 5, 10, 30 and 50 patterns, respectively, are used for the test. Each task is to categorize different input patterns selected randomly into five output patterns.

Table 1 shows the result of simulaions. As shown in the table, NN/I is superior to the conventional BP networks in completing the tasks with shorter time especially when the task, and consequently the network are large.

Second, simulations to test robustness of the memory in the NN/I are carried out in comparison with that in the conventional BP network. At the outset the network learns a task, Task A, and after completing this the network learns another task, Task B. After learning Task B, how are the responses of the network to the input patterns of Task A? In case of the conventional BP network, the responses are quite poor because the memory of Task A is destroyed almost completely by learning Task B, even if the network is equipped with a plenty of hidden units. In contrast, in case of NN/I, the responses are in good shape because Task B is assigned mainly to subnetworks different from those used for learning of Task A. Thus, robustness of the memory in NN/I

Table 1 Number of Repetition Required for Completion of Learing

Network		Number of Patterns			
Size	Type	5	10	30	50
10	BP	192	120	209	*
5*2	NN/I	174	121	206	*
25	BP	85	77	*	*
5*5	NN/I	112	148	256	276
50	BP	85	77	*	*
5*10	NN/I	110	174	230	237
250	BP	*	*	*	*
5*50	NN/I	37	39	141	141

"*" indicates that learning is not completed within 5000 iterations.
The size of NN/I is indicated by $n_H * n_S$. It is noted that CPU time required for one repetition is much shorter in NN/I than in BP.

is proved experimentally.

4. CONCLUDING REMARKS AND EXTENSIONS

This paper proposes a novel neural network, the NN/I, which improves remarkably some abilities of the error back-propagation (BP) algorithm for layered networks. NN/I divides the environment, i.e., the input patterns into several categories by the control network of Kohonen type. According to the category of the input pattern, a subnetwork of BP type is selected, and it generates the desired outputs. The advantage of NN/I over the conventional BP network in its learning speed and robustness of the memory is demonstrated through computer simulations.

From the view point of internal information representation, NN/I has an interesting nature. The control network yields a localized representation on one hand, and the main network gives dispersive representations on the other hand. Then by changing the size of a group in hidden layer of the main network, the information representation can be either more localized or more dispersive. Choice of the group size and other related problems such as the generalized internal representation of the recognized patterns are open to further investigation. The authors are investigating such problems by taking the recognition of hand-written Japanese characters as an example task.

REFERENCES

[1] D.E. Rumelhart et al.: "Learning Internal Representation by Error Propagation" in *Parallel Distributed Processing*, Vol.1, MIT Press (1986).

[2] T.J. Sejnowski, et al.: "NETtalk: a parallel network that learns to read aloud", *The Johns Hopkins Univ. Elec. Eng. and Comp. Sci. Tech. Report*, JHU/EECS-86/01, pp.32-41 (1986)

[3] P.R. Gorman et al.: "Analysis of Hidden Units in a layered Network Trained to Classify Sonar Targets", *Neural Networks*, Vol.1. pp.75-89 (1988).

[4] T. Kohonen et al.: "Statistical Pattern Recognition with Neural Networks: Benchmarking Studies," *Proc. ICNN 1988* vol.I, pp.61-68 (1988).

[5] T. Kohonen: "Self-Organization and Associative Memory" 2nd Edition, Springer (1987).

Selective Presentation of Learning Samples for Efficient Learning in Multi-Layer Perceptron

Noboru Ohnishi+ , Atsuya Okamoto++ and Noboru Sugie+

+ Department of Electrical Engineering , Faculty of Engineering
Nagoya University , Furo-cho , Chikusa-ku , Nagoya 464-01 JAPAN
++ Nippondenso Co.,Ltd.

ABSTRACT

Four methods of presenting learning samples are proposed to increase efficiency of learning in multilayer perceptrons. The methods are to selectively present samples instead of randomly presenting samples: typical and confusing samples are selected and presented in systematic orders. The methods were simulated to examine their effectivenesses in a simple three layer perceptron with two inputs and two outputs. The methods ,except the method of presenting typical samples alone, turned out to be superior to the conventional method of random presentation. Among three methods, the following two methods were best: presenting typical samples in the first half period of learning and confusing ones in the second half period of learning, and presenting in turns both typical and confusing samples.

1.INTRODUCTION

Because of the error back-propagation learning algorithm, multilayer perceptrons have been widely used in several fields such as robot manipulator control, speech processing , character recognition, and so on. But a great number of iterative learning is needed. Therefore several revisions of the back-propagation algorithm have been studied[1],[2].

In the learning of perceptrons, random samples are generally used. In this case, a teacher informs to the perceptron of whether the decision is correct or not, but doesn't concern about how to present learning samples. If a teacher presents suitable samples for learning in a systematic order, the convergence of learning will be accelerated.

So we have devised methods of selectively presenting learning samples such as typical samples of each class and confusing samples among classes. The following four methods are studied: 1)presenting typical and confusing samples in the first half and the second half period of learning, respectively, 2)presenting in turns both typical and confusing samples, 3)presenting typical samples alone, and 4)presenting confusing samples alone.

This paper describes the details of proposed methods of selective sample presentation and the results of simulation using a simple three layer perceptron with two inputs and two outputs.

2.SELECTIVE LEARNING SAMPLE PRESENTATION

A good teacher presents good samples to a student to learn discrimination of various patterns. For example, he will present typical samples of each class at first and then confusing or near miss samples among classes. Thus if suitable learning samples are selected and the order of presentation of them is proper, convergence of learning will be accelerated.

In order to examine the effect of sample presentation on learning, the following four methods are experimentally investigated.

Method 1: presenting typical samples in the first half period of learning and confusing ones in the second half period of learning.

Method 2: presenting in turns both typical and confusing samples such as typical one, confusing one, typical one and so on.

Method 3: presenting typical samples alone.

Method 4: presenting confusing samples alone.

3.EXPERIMENT

To compare four methods mentioned above with the conventional method of random samples presentation (this is later called as Method 0), simulation was carried out.

3.1 Conditions

The task used in this experiment is to learn the discrimination of two classes in two dimensional space. Class A is inside area of an unit circle centered at origin, and class B outside of it but bounded by a square (10 X 10).

A three layer perceptron with two inputs and two outputs was used. A learning set is a pair of samples of each class:(a,b), $a \in A$, $b \in B$. It is generally true that the more the number of learning sets is, the better the recognition rate becomes. But many learning steps are needed in the case of using too many learning sets. On the basis of preliminary experimental results , the number of learning sets was set to be 40 sets:80 samples were used in learning.

In each step of learning, one set is selected from 40 learning sets and is inputted to perceptron. Weights are modified by the back-propagation algorithm.

Confusing samples are selected around the unit circle which is the boundary between class A and B : $0.5 < r < 0.7$ for class A and $1.5 < r < 2.0$ for class B (r denotes a radius). On the other hand, typical ones are selected the center of each class, far from the boundary : $0 < r < 0.3$ for class A and $3.7 < r < 4.0$ for class B.

5000 sets of data different from learning samples were selected for evaluation of each method to examine the recognition rate, and the ratio of the number of correct decisions to the number of total ones. At each step of learning, the evaluation data wėre inputted and the recognition rate was calculated.

The ability of perceptron depends largely on the number of hidden units. As the results of preliminary experiment, it was found that the recognition rate saturates on and after eight hidden units. So the number of hidden units was fixed to be eight.

3.2 Results

Four methods mentioned in section 2 and random presentation of samples (Method 0) were simulated. Fig.1 shows the transition of recognition rate with increase of the number of learning.

Though Method 4 is worse than Method 0, other three methods are superior to Method 0 with regard to the higher recognition rate and faster saturation. Next comparing among three methods (i.e. method 1, 2 and 3), the followings are said. There are no difference between methods 1 and 2. The Method 2 is better than Method 3 in the learning number of less than 400, but they show nearly equal recognition rate in the learning of more than 400. Method 1 is superior to Method 3 in the learning number of less than 200. Consequently,the order of good learning is that Method 2 = Method 1 > Method 3 > Method 0 > Method 4.

3.3 Discussions

The reasons why the presentation of selected samples is better than the one of random samples are as follows. A discrimination function will be systematically formed because there are no drastic changes in the function. Next a selection of typical and confusing samples might correspond to a preprocessing such as averaging. But mathematical analysis should be done in future ,because these reasons are rather intuitive.

4.Concluding Remarks

We proposed four methods in which typical and confusing learning samples are selectively presented to the perceptron in systematic orders. As the results of the simulation, proposed methods, except Method 4, were better than the conventional method of random presentation. Though there were no clear difference among other three methods, the following two methods were better: method of presenting typical and confusing samples in the first and second half period of learning, respectively, and the method of presenting in turns both both typical and confusing samples.

REFERENCES

[1]T.P.Vogl et al.,"Accelerating the Convergence of the Back-Propagation Method", Biol. Cybern., 59,pp.257-263(1988).

[2]D.B.Parker,"Optimal algorithms for Adaptive Networks", Proc.ICNN IEEE, Vol.2, pp.593-600(1987).

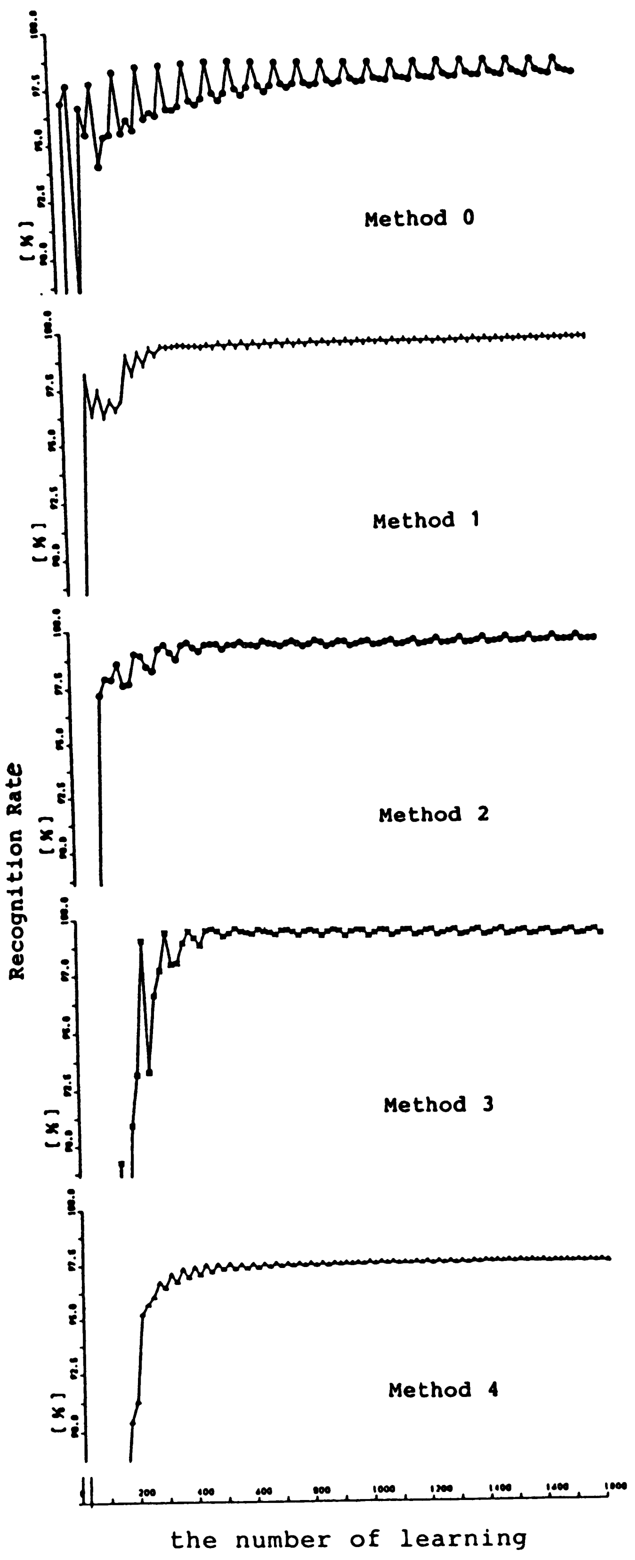

Method 0
Method 1
Method 2
Method 3
Method 4
Recognition Rate
[%]
100.0
97.5
95.0
92.5
90.0
200
400
600
800
1000
1200
1400
1600
1800
the number of learning

Fig. 1

Fast Neural Nets with Gram-Schmidt Orthogonalization

Sophocles J. Orfanidis

Department of Electrical and Computer Engineering
Rutgers University, Piscataway, NJ 08855

Abstract

A new type of feedforward multilayer neural net is proposed that exhibits fast convergence properties. It is defined by inserting a fast adaptive Gram-Schmidt preprocessor at each layer, followed by a conventional linear combiner-sigmoid part which is adapted by a fast version of the backpropagation rule. The resulting network structure is the multilayer generalization of the gradient adaptive lattice filter and the Gram-Schmidt adaptive array.

1. Introduction

A feedforward multilayer neural net adapted by the backpropagation rule [1–3] can be thought of as the multilayer generalization of the adaptive linear combiner adapted by the Widrow-Hoff LMS algorithm [4]. The backpropagation rule inherits the computational simplicity of the LMS algorithm. But, like the latter, it often exhibits slow speed of convergence.

The convergence properties of the LMS algorithm are well-known [4]. Its learning speed depends on the correlations that exist among the components of the input vectors—the stronger the correlations, the slower the speed. This can be understood intuitively by noting that, if the inputs are strongly correlated, the combiner has to linearly combine a lot of redundant information, and thus, will be slow in learning the statistics of the input data. On the other hand, if the input vector is decorrelated by a preprocessor *prior* to the linear combiner, the combiner will linearly combine only the *nonredundant* part of the same information, thus, adapting faster to the input. Such preprocessor realizations of the adaptive linear combiner lead naturally to the fast Gram-Schmidt preprocessors of adaptive antenna systems and to the adaptive lattice filters of time-series problems [4–7].

In this paper, we consider the generalization of such preprocessor structures to multilayer neural nets and discuss their convergence properties. The proposed network structure is defined by inserting, at each layer of the net, an adaptive Gram-Schmidt preprocessor followed by the conventional linear combiner and sigmoid parts. The weights of the preprocessors are adapted *locally* at each layer, but the weights of the linear combiners must be adapted by the backpropagation rule. We use fast adaptation schemes for the weights that are, in some sense, implementations of Newton-type methods for minimizing the performance index of the network. Newton methods for neural nets have been considered previously [8–11]. These methods do not change the structure of the network—only the way the weights are adapted. They operate on the correlated signals at each layer, whereas the proposed methods operate on the decorrelated ones.

2. Gram-Schmidt Preprocessors

In this section, we summarize the properties of Gram-Schmidt preprocessors for adaptive linear combiners. Our discussion is based on [7]. The Gram-Schmidt orthogonalization procedure for an $(M+1)$-dimensional random vector $\mathbf{x} = [x_0, x_1, \cdots, x_M]^T$ generates a new vector $\mathbf{z} = [z_0, z_1, \cdots, z_M]^T$ with mutually *uncorrelated* components, that is, $E[z_i z_j] = 0$ for $i \neq j$. It is defined by starting at $z_0 = x_0$ and proceeding recursively for $i = 1, 2, \cdots, M$

$$z_i = x_i - \sum_{j=0}^{i-1} b_{ij} z_j \tag{2.1}$$

where the coefficients b_{ij} are determined by the requirement that z_i be decorrelated from all the previous zs, $\{z_0, z_1, \cdots, z_{i-1}\}$. These coefficients define a *unit lower triangular* matrix B such that

$$\mathbf{x} = B\mathbf{z} \tag{2.2}$$

known as the *innovations* representation of $\mathbf{x}$. For example, if $M = 3$,

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ b_{10} & 1 & 0 & 0 \\ b_{20} & b_{21} & 1 & 0 \\ b_{30} & b_{31} & b_{32} & 1 \end{bmatrix} \begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} = B\mathbf{z}$$

An equivalent computation of the optimal weights b_{ij} is based on the *sequence* of minimization problems $\mathcal{E}_i = E[z_i^2] = \min$, $i = 1, 2, \cdots, M$. A fast adaptive solution of these, that corresponds to Newton's method with respect to the decorrelated basis, is obtained by the delta rule [7]

$$\Delta b_{ij} = \frac{\beta}{E_j} z_i z_j \tag{2.3}$$

where β is usually set to 1 and E_j is a time-average estimate of $\mathcal{E}_j = E[z_j^2]$, updated from one iteration to the next by

$$E_j = \lambda E_j + z_j^2 \tag{2.4}$$

where λ is a "forgetting" factor with a typical value of 0.9.

Next, we consider the Gram-Schmidt formulation of the adaptive linear combiner. It generates an optimum estimate of a desired output vector $\mathbf{d}$ by the linear combination $\mathbf{y} = W\mathbf{x}$, by minimizing the mean square error $\mathcal{E} = E[\mathbf{e}^T\mathbf{e}] = \min$, where $\mathbf{e} = \mathbf{d} - \mathbf{y}$ is the estimation error. The output $\mathbf{y}$ may also be expressed in the decorrelated basis $\mathbf{z}$ by $\mathbf{y} = W\mathbf{x} = G\mathbf{z}$, where G is the combiner's weight matrix in the new basis, defined by $WB = G$, which implies $W = GB^{-1}$. The conventional LMS algorithm is obtained by considering the performance index $\mathcal{E}$ to be a function of W. In this case, the matrix elements of W are adapted by $\Delta w_{ij} = \mu e_i x_j$, where μ is the learning rate. Similarly, viewing the performance index as a function of G, we obtain a fast adaptive version, analogous to (2.3)

$$\Delta g_{ij} = \frac{\mu\beta}{E_j} e_i z_j \tag{2.5}$$

with E_j adapted by (2.4). Like (2.3), it is equivalent to applying Newton's method with respect to the decorrelated $\mathbf{z}$-basis. Conceptually, the adaptation of B has nothing to do with the adaptation of G, each being the solution to a different optimization problem. However, in practice, B and G are *simultaneously* adapted using Eqs. (2.3) and (2.5). In (2.5), we used the scale factor $\mu\beta$ instead of β to allow us greater flexibility when adapting both b_{ij} and g_{ij}.

3. Gram-Schmidt Neural Nets

In this section, we incorporate the Gram-Schmidt preprocessor structures into multilayer neural nets and discuss their adaptation. Consider a conventional multilayer net with N layers and let $\mathbf{u}^n, \mathbf{x}^n$ denote the input and output vectors at the nth layer and W^n the weight matrix connecting the nth and $(n+1)$st layers. The overall input and output vectors are $\mathbf{x}^0, \mathbf{x}^N$. The operation of the network is described by the forward equations: For $n = 0, 1, \cdots, N-1$

$$\begin{aligned} \mathbf{u}^{n+1} &= W^n \mathbf{x}^n && (3.1a) \\ \mathbf{x}^{n+1} &= \mathbf{f}(\mathbf{u}^{n+1}) && (3.1b) \end{aligned}$$

where we denote $\mathbf{f}(\mathbf{u}) = [f(u_0), f(u_1), \cdots]^T$ if $\mathbf{u} = [u_0, u_1, \cdots]^T$. The sigmoidal function is defined by $f(u) = 1/(1+e^{-u})$. The performance index of network is

$$\mathcal{E} = \frac{1}{2} \sum_{\text{patterns}} (\mathbf{d} - \mathbf{x}^N)^T (\mathbf{d} - \mathbf{x}^N) \tag{3.2}$$

For each presentation of a desired input/output pattern $\{\mathbf{x}^0, \mathbf{d}\}$, the backpropagation rule [1–3] computes the gradients $\mathbf{e}^n = -\partial\mathcal{E}/\partial\mathbf{u}^n$ by starting at the output layer

$$\mathbf{e}^N = D^N(\mathbf{d} - \mathbf{x}^N) \tag{3.3}$$

and proceeding backwards to the hidden layers, for $n = N-1, N-2, \cdots, 1$

$$\mathbf{e}^n = D^n W^{nT} \mathbf{e}^{n+1} \tag{3.4}$$

where $D^n = \text{diag}\{\mathbf{f}'(\mathbf{u}^n)\}$ is the diagonal matrix of derivatives of the sigmoidal function, and $f' = f(1-f)$. The weights W^n are adapted in the conventional way

$$\Delta w_{ij}^n = \mu e_i^{n+1} x_j^n \tag{3.5}$$

The proposed Gram-Schmidt network structure is defined by inserting an adaptive Gram-Schmidt preprocessor at each layer of the network. Let $\mathbf{z}^n$ be the decorrelated outputs at the nth layer and B^n the corresponding Gram-Schmidt matrix, such that by Eq. (2.2), $\mathbf{x}^n = B^n\mathbf{z}^n$, and let G^n be the linear combiner matrix. It is related to W^n by $W^nB^n = G^n$, which implies that $W^n\mathbf{x}^n = G^n\mathbf{z}^n$. The forward equations (3.1) are replaced now by

$$\begin{aligned} B^n\mathbf{z}^n &= \mathbf{x}^n && (3.6\text{a}) \\ \mathbf{u}^{n+1} &= G^n\mathbf{z}^n && (3.6\text{b}) \\ \mathbf{x}^{n+1} &= \mathbf{f}(\mathbf{u}^{n+1}) && (3.6\text{c}) \end{aligned}$$

where (3.6a) is solved for $\mathbf{z}^n$ by forward substitution, as in (2.1). Bias terms in Eq. (3.6b) can be incorporated by extending the vector $\mathbf{z}^n$ by an additional unit which is always on. Inserting $W^n = G^n(B^n)^{-1}$ in the backpropagation equation (3.4), we obtain $\mathbf{e}^n = D^n(B^{nT})^{-1}G^{nT}\mathbf{e}^{n+1}$. To facilitate this computation, define the intermediate vector $\mathbf{f}^n$ such that $B^{nT}\mathbf{f}^n = G^{nT}\mathbf{e}^{n+1}$. Then, Eq. (3.4) can be replaced by the pair

$$\begin{aligned} B^{nT}\mathbf{f}^n &= G^{nT}\mathbf{e}^{n+1} && (3.7\text{a}) \\ \mathbf{e}^n &= D^n\mathbf{f}^n && (3.7\text{b}) \end{aligned}$$

where, noting that B^{nT} is an upper triangular matrix, Eq. (3.7a) may be solved efficiently for $\mathbf{f}^n$ using backward substitution. Using (2.3), the adaptation equations for the weights B^n are given by

$$\Delta b_{ij}^n = \frac{\beta}{E_j^n} z_i^n z_j^n \tag{3.8}$$

with E_j^n updated from one iteration to the next by

$$E_j^n = \lambda E_j^n + (z_j^n)^2 \tag{3.9}$$

Similarly, the adaptation of the weights G^n, based on (2.5), is given by

$$\Delta g_{ij}^n = \frac{\mu\beta}{E_j^n} e_i^{n+1} z_j^n \tag{3.10}$$

The complete algorithm consists of the forward equations (3.6), the backward equations (3.3) and (3.7), and the adaptation equations (3.8), (3.9), and (3.10).

4. Simulation Results

In this section, we present some simulations illustrating the performance of the proposed network structures. Consider two network examples, the first is a 3:3:2 network consisting of three input units, two output units, and one hidden layer with three units, and the second is a 3:6:2 network that has six hidden units. We choose a set of eight input/output training patterns given by

0	0	0	0	1
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	0

The first of the two outputs is simply the XOR sum of the three inputs and the second output is the complement of the first. We computed the average convergence times for the above examples based on 200 simulations with random initializations. Convergence time was defined as the number of iterations for the performance index (3.2) to drop below a certain threshold value—here, $\mathcal{E}_{\max} = 0.01$. One iteration represented one epoch, that is, the presentation of all eight patterns in sequence. The following values of the parameters were used: $\mu = 0.25$, $\beta = 1$, $\lambda = 0.85$. We used both epoch updating and pattern updating with a value of the momentum parameter $\alpha = 0.85$. The average convergence times are shown in the table below. The speed advantage of the Gram-Schmidt method is evident.

	3:3:2		3:6:2	
	epoch	pattern	epoch	pattern
conventional	2561	1038	1660	708
Gram-Schmidt	923	247	349	100

References

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation" in D. E. Rumelhart and J. L. McClelland, Eds., *Parallel Distributed Processing*, vol. 1, Cambridge, MA, MIT Press, 1986.

[2] P. J. Werbos, "Backpropagation: Past and Future", *Proc. IEEE Int. Conf. Neural Networks*, San Diego, July 1988, p. I-343, and earlier references therein.

[3] D. B. Parker, "Optimal Algorithms for Adaptive Networks: Second Order Back Propagation, Second Order Direct Propagation, Second Order Hebbian Learning", *Proc. IEEE First Int. Conf. Neural Networks*, San Diego, June 1987, p. II-593, and earlier references therein.

[4] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*, Englewood-Cliffs, NJ, Prentice-Hall, 1985.

[5] R. A. Monzingo and T. W. Miller, *Introduction to Adaptive Arrays*, New York, Wiley, 1980.

[6] R. T. Compton, *Adaptive Antennas*, Englewood-Cliffs, NJ, Prentice-Hall, 1988.

[7] S. J. Orfanidis, *Optimum Signal Processing*, 2nd edition, New York, McGraw-Hill, 1988.

[8] R. S. Sutton, "Two Problems with Back Propagation and Other Steepest-Descent Learning Procedures for Networks", *Proc. 8th Ann. Conf. Cognitive Sci. Soc.*, 1986, p. 823.

[9] R. L. Watrous, "Learning Algorithms for Connectionist Networks: Applied Gradient Methods of Non-linear Optimization", *Proc. IEEE First Int. Conf. Neural Networks*, San Diego, June 1987, p. II-619.

[10] S. Kollias and D. Anastassiou, "Adaptive Training of Multilayer Neural Networks Using a Least Squares Estimation Technique", *Proc. IEEE Int. Conf. Neural Networks*, San Diego, July 1988, p. I-383.

[11] R. A. Jacobs, "Increased Rates of Convergence Through Learning Rate Adaptation", *Neural Networks*, 1, 295 (1988).

FAST TRAINING OF MULTILAYER PERCEPTRONS USING MULTILINEAR PARAMETRIZATION

Francesco Palmieri and Samir A. Shah
Department of Electrical and Systems Engineering
The University of Connecticut
Storrs, CT 06269-3157

Abstract

The training of multilayer perceptrons by the Backpropogation algorithm is usually plagued by poor convergence and misadjustment.

In this paper, we introduce a faster version of the Backpropogation algorithm based on the technique of *Multilinear Parametrization* using *Recursive Least Squares*. We show that the added complexity associated with the Enhanced algorithm can be easily incorporated in the network architecture; consequently, the update of the connection weights is local. The enhanced algorithm performed consistently better when compared to the Backpropogation algorithm in a set of simulations involving two benchmark problems.

1 Introduction

A *multilayer perceptron* (MLP) is a combination of linear blocks and memoryless, sigmoidal nonlinearities. Despite the simple structure, a multilayer perceptron is endowed with almost unlimited functional capability. Recently, Cybenko and others [1] [2] reported that continuous valued multilayer perceptrons with two hidden layers and a fixed sigmoidal nonlinearity can approximate any continuous function from $\mathbf{R}^m$ to $\mathbf{R}^n$ (with arbitrary accuracy) on a compact set given sufficient units in the hidden layers.

In a typical application, paired input-output vectors from the training set are presented to the network. The MLP network "learns" the input-output mapping using the *Backpropogation algorithm* (BP) [3] [12].

Each iteration of the backpropogation algorithm is accomplished in three steps [12]. In the first step, the input vector is propogated through the MLP to produce a network output. This output is compared with the desired output to generate an error. Subsequently, this error is backpropogated through the network to define errors at the output of the intermediate linear blocks. Finally, the backpropogated error is utilized to update the connection weights in the linear blocks. In the conventional Backpropogation algorithm the weights are updated via an LMS-like algorithm [5].

Backpropogation algorithm works well for small nets. It is computationally inexpensive and local. However, it is plagued by slow convergence, misadjustment, and the rate of convergence is critically dependent on the learning parameter.

There are numerous schemes to speedup the convergence of BP algorithm. We could group them into two categories: algorithms based on heuristics such as those report by Jacobs and Falhman [4]; and algorithms stemming from optimization techniques like conjugate gradient methods and approximations to the Newton method [8]. Recently, Singhal and Wu [9] reported a method to train the network using the extended Kalman filter. Although this method is computationally expensive and not suited for real implementation, it is probably the fastest in terms of convergence.

In this paper, we introduce a fast version of the Backpropogation algorithm based on the idea of *Multilinear Parametrization* using *Recursive Least Squares* (RLS). We show that the added storage requirement and computation associated with RLS can be easily incorporated into the network architecture, consequently the algorithm is still local.

The RLS technique has also been idependently suggested by Azimi-Sadjadi and Citrin [10].

The following section presents a matrix formulation of the Backpropogation algorithm. The new algorithm is introduced in Section 3. Finally, a set of simulation results is presented in Section 4.

2 Conventional Backpropogation Algorithm

Consider the multilayer perceptron (three layers for simplicity) depicted in Figure 1. The network consists of a cascade of three multi-input multi-output linear filters coupled by memoryless, sigmoidal nonlinearities.

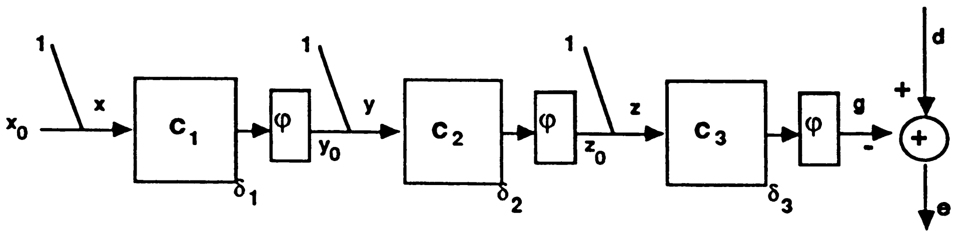


Figure 1: A multilayer perceptron

The three linear filters are characterized by connection matrices $\mathbf{C}_1$, $\mathbf{C}_2$ and $\mathbf{C}_3$ of dimensions $N_2 \times (N_1+1)$, $N_3 \times (N_2+1)$ and $N_4 \times (N_3+1)$ respectively.

The nonlinear transformation , $\varphi(x)$, is the logistic function $1/(1+e^{-x})$ and is applied element by element to the output of the three linear blocks. The biasing of the nonlinearities is accomplished by augumenting the vectors $\mathbf{x}_0$, $\mathbf{y}_0$ and $\mathbf{z}_0$ as follows :

$$\mathbf{x} = [1\ \mathbf{x}_0{}^T]^T \quad \mathbf{y} = [1\ \mathbf{y}_0{}^T]^T \quad \mathbf{z} = [1\ \mathbf{z}_0{}^T]^T$$

Given a set of input output pairs $\{(\mathbf{x}_0(n), \mathbf{d}(n))\ \ i = 1, \cdots, L\}$, the BP algorithm updates the weight matrices in the steepest descent direction of the instantaneous squared error at the output. The algorithm can be rewritten in the matrix form as follows,

1. Conventional Backpropogation algorithm

```
0. initialize the matrices $\mathbf{C}_1$, $\mathbf{C}_2$ and $\mathbf{C}_3$ to random weights.
for every input output pair $\{\mathbf{x}_0(n), \mathbf{d}(n)\}$
1. propogate the input forward   $\mathbf{y}_0 = \varphi(\mathbf{C}_1\mathbf{x})$,  $\mathbf{z}_0 = \varphi(\mathbf{C}_2\mathbf{y})$,  $\mathbf{g} = \varphi(\mathbf{C}_3\mathbf{z})$
2. $\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{g}(n)$   /* —error vector— */
/* backpropogate the error */
3. $\delta_3(n) = \mathbf{g}(n) \bullet (1 - \mathbf{g}(n)) \bullet \mathbf{e}(n)$
4. $\mathbf{C}_3(n+1) = \mathbf{C}_3(n) + \mu_3\delta_3(n)\mathbf{z}^T(n)$
5. $\delta_2(n) = \mathbf{z}_0(n) \bullet (1 - \mathbf{z}_0(n)) \bullet \mathbf{C}_{30}^T(n)\delta_3(n)$
6. $\mathbf{C}_2(n+1) = \mathbf{C}_2(n) + \mu_2\delta_2(n)\mathbf{y}^T(n)$
7. $\delta_1(n) = \mathbf{y}_0(n) \bullet (1 - \mathbf{y}_0(n)) \bullet \mathbf{C}_{20}^T(n)\delta_2(n)$
8. $\mathbf{C}_1(n+1) = \mathbf{C}_1(n) + \mu_1\delta_1(n)\mathbf{x}^T(n)$
```

where μ_1, μ_2 and μ_3 are the learning parameters; δ_1, δ_2 and δ_3 are the error vectors of sizes $\mathbf{N}_2$, $\mathbf{N}_3$ and $\mathbf{N}_4$ respectively; $\bullet$ represent an element by element multiplication, and the matrices $\mathbf{C}_{i0}$ (for $i = 1, 2, 3$) are the matrices C_i with the first column removed

3 Enhanced Backpropogation Algorithm

The problem of computing the weights in a MLP can be viewed as a global nonlinear identification problem. Due to the cascaded structure of the network, it can be partitioned into a set of linear least squares problems. Each subproblem corresponds to identifiying one linear block in the cascaded structure. Using the technique of *multilinear parametrization* (an extension of bilinear parametrization [6]), we can solve the global problem by iteratively solving each subproblem independently, while keeping the other blocks fixed. One of the most efficient ways of solving each linear subproblem is to employ the technique of recursive least squares [5]. The RLS algorithm is summarized in the following paragraph.

Consider a linear transformation $\eta = \mathbf{C}\zeta$, where $\mathbf{C}$ is an $(N \times M)$ matrix. The adaptive steps for $\mathbf{C}$ to minimize the sum of squared error between the actual output, η and the desired output, θ are:

2. Recursive Least Squares

a. $\zeta_1(n) = \lambda^{-1}\mathbf{P}(n)\zeta(n)$

b. $\mathbf{k}(n) = \zeta_1(n)[1 + \zeta(n)^T\zeta_1(n)]^{-1}$

c. $\mathbf{C}(n+1) = \mathbf{C}(n) + \delta(n)\mathbf{k}(n)^T$

d. $\mathbf{P}(n+1) = \lambda^{-1}\mathbf{P}(n) - \mathbf{k}(n)\zeta_1(n)^T$

$\zeta_1(n)$ is the linearly transformed vector $\zeta(n)$; $\mathbf{k}(n)$ is the Kalman gain; $\delta(n) = \theta(n) - \eta(n)$; P is the current inverse of the correlation matrix of ζ. $0 < \lambda \leq 1$ is the forgetting factor.

P(n) is initialized to a diagonal matrix with large diagonal terms [5].

For each input-output pair, the most straightforward adaptation strategy involves updating all the linear blocks by an RLS iteration, one at a time, beginning with the output block and proceeding towards the input. Thus, the Enhanced algorithm can be obtained by substituting each of the steps 4, 6 and 8 in the conventional Backpropogation algorithm by the above set of equations. The errors δ_1, δ_2 and δ_3 of algorithm-1 correspond the error δ in the RLS steps, the inputs x, y and z correspond to ζ; $\mathbf{C}_1$, $\mathbf{C}_2$ and $\mathbf{C}_3$ correspond to C. Additional storage is required for the three matrices $\mathbf{P}_1$, $\mathbf{P}_2$ and $\mathbf{P}_3$. These matrices will account for the correlation at the input of the three linear blocks.

The errors δ_1, δ_2 and δ_3 have to be backpropogated from the output through the nonlinearities and the linear blocks. Note that the backpropogation of the errors through the nonlinearities could be simplified by excluding the multiplication of the error by the derivative of the nonlinearity, as in steps 3, 5 and 7. Since the nonlinearity is monotonically nondecreasing, a change in the error at the input of the nonlinearity corresponds to a change in the error at the output in the same direction. Therefore the error could be carried through the nonlinearity without modification.

$$\delta_3 = \mathbf{e} \quad \delta_2 = \mathbf{C}_{30}^T\delta_3 \quad \delta_1 = \mathbf{C}_{20}^T\delta_2$$

Note that we found this simplification usually leads to faster convergence.

The RLS based algorithm provides very fast convergence at the expense of greater computational complexity. In fact additional matrices $\mathbf{P}_1$, $\mathbf{P}_2$ and $\mathbf{P}_3$ have to be stored at the input of each linear block for the computation fo the Kalman gains. This increased complexity could be incorporated into the network structure in the form of lateral connections at the input of each linear block. The update of the forward and the lateral connections (steps *c* and *d*) is Hebbian in both cases because it is proportional to a product term. At initialization the $\mathbf{P}_i$ $(i = 1, 2, 3)$ are diagonal, implying no lateral connections, but as the algorithm progresses the lateral connections grow.

The enhanced backpropogation algorithm (EBP) usually exhibits very fast convergence but presents some drawbacks. It is typical of multilinear Parametrization techniques to occasionally end up in a local minima. A way of alleviating this problem is to randomize the order in which the input-output patterns are presented to the network. Also, if the network contains enough redundancy, the locally minimum solution may be acceptable. The forgetting factor, λ [5] can be used to correct the so called ***data saturation*** problem[9] typically associated with any Kalman algorithm.

4 Simulation Results

We investigated the convergence properties of the enhanced backpropogation algorithm on two benchmark problems - the binary XOR and the 4-2-4 encoder [4] [7].

In both problems the training set consisted of four training patterns. The MLP was presented with patterns sequentially and the network was updated after every presentation. A sweep through the entire training set is referred to as an *epoch*. Several epochs are required to accomplish learning. To avoid getting trapped into a local minima, the ordering of patterns within a training set is randomly shuffled after every epoch. The target values of 0.1 and 0.9 are used instead of 0 and 1. The network output is thresholded at 0.5, and when the desired output matches the network output over two consecutive epochs, training is said to be complete.

XOR : The network consisted of two inputs, two hidden units and one output unit (2-2-1 architecture). The learning of an XOR problem by a 2-2-1 network is usually difficult since this a minimal configuration. Twenty-five simulations were performed with random initial connection weights uniformly distributed between -0.25 and +0.25. The results of the simulation are presented in Table 1. A simulation was considered

	no. of runs	no. of successes	max (epochs)	min (epochs)	avg. (epochs)	median (epochs)	SD
EBP	25	19	533	12	154	75	165
BP	25	11	591	232	350	297	112

Table 1: XOR simulation

	no. of runs	no. of successes	max (epochs)	min (epochs)	avg. (epochs)	median (epochs)	SD
EBP	25	25	61	7	24	19	16
BP	25	22	95	17	52	52	18

Table 2: 4-2-4 encoder simulation

successful if it converged within 600 epochs. The statistics presented in Table 1 exclude unsuccessful simulations. The learning rate (μ) for the conventional backpropogation was set at 0.7 and no momentum was used.

4-2-4 encoder : The network consisted of 4 input units, 2 hidden units, and 4 output units (4-2-4 architecture). The network was presented with four distinct input patterns, each of which had only one bit turned on (set at 1). The task of the network was to duplicate the input pattern at the MLP output. To accomplish this task, the network had to develop a unique encoding scheme at the hidden layer [7]. The simulation results are presented in Table 2. If a simulation converged within 100 epochs, it was considered successful. The learning parameter for conventional backpropogation was set at 0.3.

5 Conclusion

In this paper, we presented an enhanced version of the Backpropogation algorithm. The simulation results clearly indicate the superiority of the enhanced backpropogation algorithm. Since this algorithm does not involve any tunable parameters, guesswork is eliminated and good performance can be expected for various problems. The algorithm is computationally complex but local.

Simulation results on continuous input-output mapping have shown encoraging results in terms of convergence rate and *generalization*. These results will be reported elsewhere [11].

References

[1] G. Cybenko, "Continuous Valued Neural Networks with Two Layers are Sufficient," Tech. Rep. Tuft University, Medford, MA, March 1988.

[2] Ken-ichi Funahashi, "On the Approximate Realization of Continuous Mappings by Neural Networks," *Neural Networks*, Vol. 2, No. 3, pp. 183-192, 1989.

[3] D. E. Rumelhart, G. E. Hinton and P. J. Williams, "Learning Internal Representation by Error Propagation," in *Parallel Distributed Processing*, Vol. 1, Ed. D. E. Rumelhart and J. L. McClelland, MIT Press 1986.

[4] R. A. Jacobs, "Increased Rates of Convergence Through Learning Rate Adaptation," *Neural Networks*, Vol. 1, N. 4, pp. 295-307, 1988.

[5] S. Haykin, *Adaptive Filter Theory*, Prentice Hall, 1986.

[6] L. Ljung, *System Identification, Theory for the User*, Prentice Hall, 1987.

[7] Scott Falhman, "Faster-learning Variations on Backpropogation : An empirical study," *Proceedings of the 1988 Connectionist Models*, Summer school 1988, Carnegie Mellon University, pp. 38-50, 1988.

[8] Sue Becker, Yann le Cunn, "Improving the convergence of Backpropogation Learning using Second Order Methods," *Proceedings of the 1988 Connectionist Models*, Summer school 1988, Carnegie Mellon University, pp. 29-37, 1988.

[9] S. Singhal, L. Wu, "Training Feed-Forward Networks with the Extended Kalman Filter," *Proceedings of 1989 International Conference on ASSP*, pp. 1187-1190, Glasgow, Scotland, 1989.

[10] M.R. Azimi-Sadjadi, S. Citrin, "Fast learning of Multi-Layer Perceptron using Recursive Least Squares Technique," abstract in the Proceedings of the *International Joint Conference on Neural Networks*, Washington D.C., June 1989.

[11] F. Palmieri, S. Shah, "A New Algorithm to Train Multilayer Perceptrons," to be presented at the *1989 IEEE International Conference on Systems, Man and Cybernetics*, Boston, MA, 14-17 Nov., 1989.

[12] P.J. Werbos, "Generalization of Backpropogation with application to recurrent gas market model," *Neural Networks*, Vol. 1, N. 4, pp. 339-356, 1988.

Learning by Local Variations

Ajay Patrikar and John Provence
Department of Electical Engineering
Southern Methodist University
Dallas, TX 75275

Abstract

A new learning algorithm for multilayer feedforward neural networks is developed. The algorithm trains the neural network by applying local variations to its parameters (i.e., weights and thresholds) and referred to as Learning by Local Variations (LLV) algorithm. The initial values of weights and thresholds are generated randomly and then systematically perturbed such that the cumulative mean square error decreases monotonically with each iteration. The performance of this algorithm was compared with the back-propagation learning algorithm. It is observed that LLV algorithm outperforms back-propagation in terms of computational requirement, simplicity and accuracy. Extensive simulation studies indicate that the LLV algorithm has some desirable features such as fast convergence rate and reduced dependence of rate of convergence on the initial conditions.

I. Introduction

With the development of back-propagation (BP) algorithm [1] the multilayer feedforward networks have been used successfully to solve number of pattern classification problems. However, when the BP algorithm is employed to train neural networks to solve complex problems, the computational time requirements are often very high. Also, the BP algorithm suffers from some limitations such as dependence of rate of convergence on initial values of weights and the possibility of converging to local minima. Several attempts have been made to modify the BP algorithm to achieve faster convergence. The second order BP methods [2]-[3] improve the rate of convergence but with added computational complexity.

Our aim in this paper is to present a new learning algorithm based on local variations of weights and thresholds. Our approach differs significantly from that of error back-propagation. We make systematic changes in the weights, one at a time, and see if any improvement is obtained at the output. If there is any reduction in cumulative mean square error we make the changes permanent. Otherwise we discard the changes. The cumulative mean square error decreases monotonically with each successful local variation.

This technique is termed as "Learning by Local Variations" or LLV algorithm. The LLV algorithm is very simple to apply and it has very fast convergence rate as compared to the BP algorithm. Extensive simulation studies indicate that its convergence rate is less dependent on the initial conditions and the possibility of its converging to local minima is low.

Section II of this paper describes the learning problem. Section III presents the LLV algorithm. The discussion regarding computational complexities of LLV and BP algorithms and the simulation results are presented in section IV. Finally section V is the conclusion.

II. Problem Formulation

A multilayer feedforward neural network with one hidden layer is shown in Fig. 1. When an input pattern $X_p=[x_{0p},x_{1p},\ldots,x_{(N_0-1)p}]$ is presented at the input, it propagates forward in the network and maps into an output pattern $Y=[y_0,y_1,\ldots,y_{(M-1)}]$. The layer l contains N_l neurons. x_j^l denotes the output of the j th neuron belonging to the l th layer and is computed as

$$x_j^l=f(\sum_{n=0}^{N_l-1} w_{nj}^{l-1}x_n^{l-1}-\theta_j^l) \tag{1}$$

where w_{nj}^{l-1} is the weight of connection between the n th neuron in the (l-1) th layer to the j th neuron in the l th layer and θ_j^l is the offset value of the j th neuron in the l th layer. The function f is characterized by sigmoid nonlinearity.

In supervised training of neural networks we are given a set of P input patterns X_p , p = 1, .. , P which need to be mapped into the corresponding set of output patterns. Let $D_p=[d_{0p},d_{1p},\ldots,d_{(M-1)p}]$ be the desired output pattern corresponding to the input pattern X_p; and Y_p be the actual output obtained during training phase. The mean square error for pattern p is given by

$$E_p=\frac{1}{M}\sum_{n=0}^{M-1}(d_{np}-y_{np})^2 \tag{2}$$

and the cumulative mean square error is given by

$$E=\frac{1}{P}\sum_{p=1}^{P}E_p \tag{3}$$

The learning problem is to find the set of parameters w_{nj}^l and θ_j^l, such that the cumulative mean square error given by eq.(3) is minimized.

III Learning by Local Variations

A set of initial weights and thresholds are generated randomly. All the input patterns X_p,p = 1,..,P are presented at the input of neural network and the corresponding output patterns Y_p are evaluated by forward propagation. The cumulative mean square E is calculated using eqs.(2) and (3).

In each iteration of LLV, local variations are performed on each of the weights until all the weights in the network are covered. The treatment of thresholds is the same as weights. To perform local variations on the weight w_{nj}^l , first positive local variation is attempted as follows.

$$w_{nj}^{l'}=w_{nj}^l(t)+\delta w^l \tag{4}$$

where $w_{nj}^l(t)$ is the current value of weight w_{nj}^l and δw^l is value of the incremental change to be attempted in the weights of layer l. The value of δw^l is chosen through trial and error at the beginning of the algorithm. All the patterns are now applied to the neural network with this new value of weight and the cumulative mean square error E' is calculated. If E' is less than E then the positive local variation is successful and the weight $w_{nj}^l(t+1) = w_{nj}^{l'}$. If positive variation is successful we go to the next weight. Otherwise the positive variation is discarded and the negative local variation is attempted as follows:

$$w_{nj}^{l'}=w_{nj}^l(t)-\delta w^l \tag{5}$$

A similar procedure is repeated for this change and E' is calculated. If E' is less than E we make the change permanent i.e., $w_{nj}^l(t+1) = w_{nj}^{l'}$, otherwise, none of the variations are successful and we go to the next weight. In each iteration we keep count of the successful variations in weights of each layer l. If there are no successful variations in the weights of layer l (i.e., w_{nj}^l), then the value of δw^l is reduced to $\alpha\delta w^l$ $(0<\alpha<1)$ for the next iteration.

Thus each iteration of the algorithm involves local variation of each of the weights and thresholds of the network. The algorithm is terminated when sufficient reduction in the cumulative mean square error is obtained. The LLV algorithm thus performs a search in the local neighborhood of the point represented by current weights in the weight space.

Whenever a local variation is attempted on a given weight, the entire forward propagation for patterns need not be carried out. Instead, only the variables along a path affected by the variation should be reevaluated. For example, when a local variation is being attempted on the weight w_{nj}^l in the network shown in Fig. 1, only the variables along dotted paths are affected and need to be reevaluated. This requires that some extra variables such as activations and outputs of neurons need to stored for each pattern in the pattern set. This scheme brings about significant reduction in computations for every pattern presentation and is feasible in terms of memory storage if the number of patterns is not very large.

IV Simulation Results

In order to compare the performance of two algorithms we need to take into consideration the time complexity as well as the optimization efficiency of the algorithms [2]. We limit our discussion to one hidden layer networks as these are the most widely used. The time complexities of the network with one hidden layer are presented here. One iteration of BP has time complexity of O(n) where $n=N_0N_1+N_1N_2$. One iteration of LLV has complexity of $O(n')$ where $n'=N_0N_1N_2+N_0N_1+N_1N_2$. For detailed comparison of two methods see Table I. However the number of iterations required for convergence by the LLV algorithm is significantly less than that of BP algorithm. In most of the problems attempted during simulation studies we obtained convergence within 15-100 iterations using LLV while BP took several thousand iterations.

The performance of these two methods on some of the problems such as Exclusive-OR, Parity and Mux is presented in Table II. The parameters of both the algorithms were chosen so as to obtain their best performance. The initial values of weights and thresholds were chosen to be random numbers in the interval [-0.5,0.5] and were identical for both the algorithms. Column 1 of the Table II gives information about architecture of the neural network to solve the problem. Thus 2-2-1 for XOR problem means that there are two inputs, two neurons in the hidden layer and one output. Parity is a four bit parity problem with six neurons in the hidden layer. The multiplexer problem consists of a two bit select field, a four bit data field and a one bit output. The algorithms are said to be converged if the cumulative mean square error falls below a certain threshold. In the problems described here the threshold was chosen to be $(0.01)^2$ i.e., the error in each of the output bits is approximately 0.01 for each pattern. In some cases the BP method could not meet this criterion even after a large number of iterations.

Figs. 2 to 4 show cumulative square error versus the number of pattern set presentations for each of the problems. It is evident that the rate of convergence for LLV is much higher than that of BP. Also as the error becomes smaller, BP slows down. But no such effect is observed in case of LLV. These plots, however, do not present a clear picture of the effectivenesss of LLV because the number of computations per pattern set presentation for LLV is much lesser than that of BP as mentioned in section III. Also, in the case of LLV, there is no error back-propagation with each pattern presentation. Evaluation of number of computations required by both the methods on small to moderate sized networks indicates that LLV requires one to two orders of magnitude lesser computations than BP. For larger networks the improvement in performance was observed to be more problem specific and not so significant. We also observed the performance of these methods under different initial conditions. The performance of LLV was less sensitive to the initial conditions than that of BP. Also the frequency of LLV to converge to local minima was significantly less than that of BP.

V Conclusion

A new learning algorithm for multilayer feedforward neural network has been presented. The proposed algorithm is much simpler and computationally efficient compared to the back-propagation algorithm for small to moderate sized networks and often gives more accurate results. Simulation studies indicate that the convergence rate of LLV is less dependent on the initial conditions and the probability that it would converge to local minima is significantly less than that of back-propagation. The back-propagation method often takes an excessively large time to train networks to solve complex problems. The new technique has potential of training neural networks to solve complex problems within a reasonable time period.

References

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation" in D. E. Rumelhart & J. L. McClelland (Eds.), Parallel Distributed Processing, vol. 1: Foundations. MIT Press (1986)

[2] R. L. Watrous, "Learning Algorithms for Connectionist Networks: Applied Gradient Methods of Nonlinear Optimization" in IEEE International Conference on Neural Networks, 1987.

[3] S. Kollias, D. Anastassiou, "Adaptive Training of Multilayer Neural Networks Using a Least Squares Estimation Technique" in IEEE International Conference on Neural Networks, 1988.

	Back-propagation	LLV (worst case)
Mults	$P(5N_1N_2+4N_0N_1+4N_1+4N_2)$	$2P(N_0N_1N_2+N_0N_1+2N_1N_2)$
Sigmoids	$P(N_1+N_2)$	$2P(N_0N_1N_2+N_0N_1+2N_1N_2+N_1+N_2)$

Table I. Approximate number of computations per iteration for one hidden layer network.

	BP	LLV
Xor 2-2-1	700	16
Parity 4-6-1	4700	42
Mux 6-4-1	>15000	96

Table II. Number of iterations for convergence.

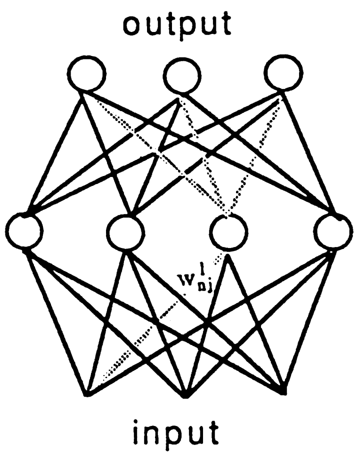


Fig 1. Multilayer feedforward Network.

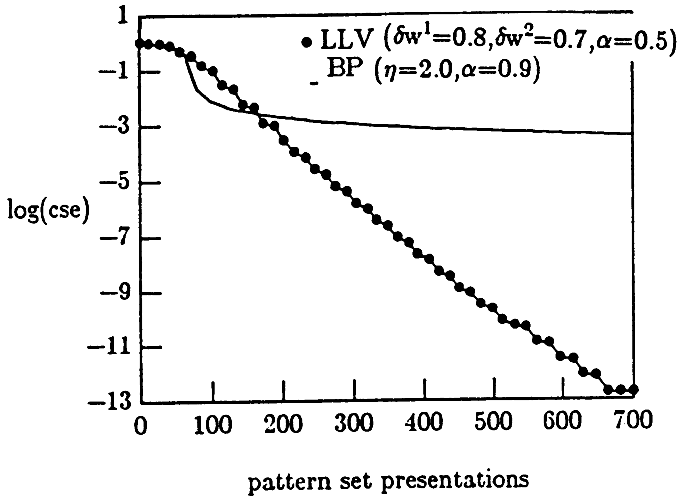


Fig. 2. The XOR problem.

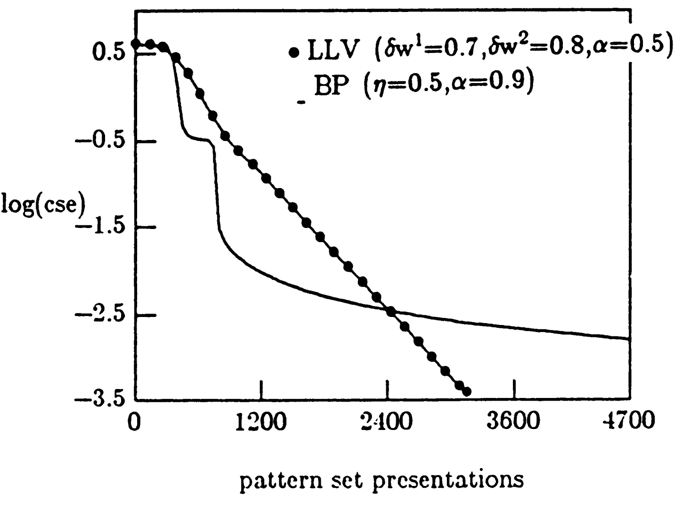
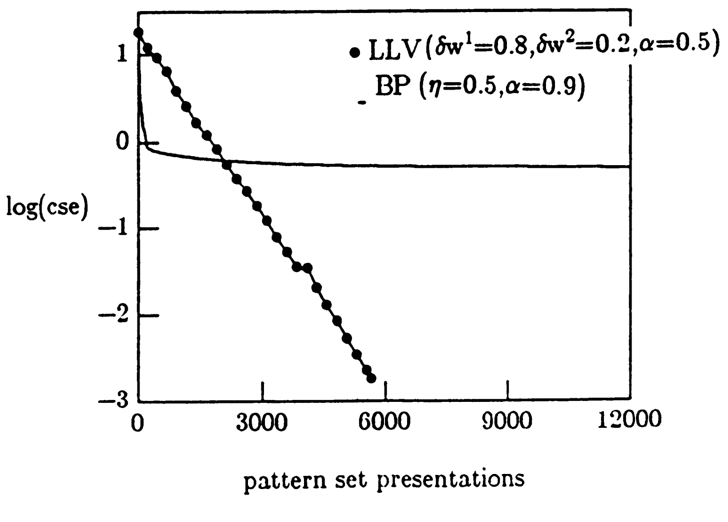


Fig. 3. The Parity problem



4. The Mux problem.

A NEW LEARNING ALGORITHM FOR THE BSB MODEL

Robert Proulx and Jean Begin
Department of Psychology
Universite du Quebec a Montreal
Montreal, Que, H3C 3P8 Canada

During the past ten years, connectionist networks have been applied to a wide variety of problems ranging from low level pattern recognition to verbal learning and decision optimization (Rumelhart & McClelland, 1986). Among the various parallel distributed models proposed, the Brain-State-in-a-Box (BSB) of Anderson Silverstein Ritz and Jones (1977) has the advantage of being able to explain the process of categorization in a natural learning framework. In the model, this is accomplished on the one hand, by a positive feedback loop which continuously returns the current state back to the input, thus forcing any initial ambiguous state to converge toward one of the discrete directions defined by the eigen vectors of the connectivity matrix, and on the other hand, by the imposition of saturation limits that constrain vectors to lie inside a hypercube of which corners, if they are stable, constitute the invariant final states of the system. Thus, the performance of the model is closely linked to conditions under which stable responses (corners) are acquired.

In a recent study, Proulx (1986, 1987) criticized the notion of stability as defined by Anderson *et al*, (1977) and showed that certain selectivity problems emerge when the system learns in the presence of non-orthogonal stimuli. More precisely, the imposed limits to adaptation of connections, which prevent the system from being dominated by it's first eigen vector (Anderson and Mozer, 1981, Golden, 1986), inevitably leads to decreased efficiency of the feedback to drive initial states in the appropriate direction, while simultaneously facilitating the stability of all the corners of the hypercube. This seriously limits the applicability of the BSB model to human categorization.

In this paper, we present a revised version of the BSB model in which the coefficients of the connectivity matrix are updated according to two distinct learning rules instead of only one. The first is the standard hebbian rule used by Anderson & Mozer (1981) and the second is an anti-learning rule that operates with a general parameter slightly less in magnitude and opposite in sign relative to the first rule. Such a modification of the BSB's learning algorithm has for major effect to introduce a negative feedback in addition to the usual positive feedback in the system. Moreover, since the application of the two synaptic modification rules is asynchronous (anti-learning following learning by five iterations), the canonical bases of the two components of the feedback never overlap and the resulting effect on the performance of the model is much more complex and interesting than what is expected by the simple application of two antagonistic processes.

More specifically, as demonstrated by a theoretical analysis of the dynamic properties of the feedback, as well as by the results of several computer simulations based on the operation of a 34 unit network in a recognition task of nine correlated stimuli (letters), the new model performs better than the standard BSB at all levels of analysis. First, inspection of the evolution the eigen structure of the connectivity matrix as a function of the number of learning trials reveals 1) that the eigen vectors of the total matrix develop progressively to represent a well defined set of distinctive features of the objects in the stimulus domain, and 2) that each eigen value sequentially emerges from noise level to converge toward a magnitude that depends on the relative importance of the associate characteristic (eigen vestor). Second, an analysis of the two eigen value spectra obtained for each component of the feedback demonstrates that the ratio between successive eigen values increases, thus indicating that the system becomes more selective as it learns. This conclusion is also supported by the fact 1) that the final corners used by the system are exact representations of the input stimuli though, in the presence of noise, unaltered letters have never been presented, and 2) that only those corners reach the appropriate level of stability during the learning process. Finally, when tested in the presence of varying levels of noise, the performance of the model is nearly ideal, both in lterms of accuracy of categorizations and in speed of recognition.

Such findings strongly suggest that reconciliation of selectivity with generalization in the BSB model depends on the dynamic properties of the interaction of two types of feedback rather than on the effect of positive feedback only.

References

Anderson, J.A. (1983). "Cognitive and psychological computation with neural models". *IEEE Transactions on Systems, Man, and Cybernetics,* 13(5) 799-815.

Anderson, J.A. & Mozer, M.C. (1981). "Categorization and selective neurons". In G.E. Hinton, & J.A. Anderson (Eds.) *Parallel models of Associative Memory* (pp.213-236), Hillsdale, N.J.; Lawrence Erlbaum Associates.

Anderson, J.A., Silverstein, J.W., RRitz, S.A., & Jones, R.S. (1977). "Distinctive features, categorical perception, and probability learning· Some applications of a neural model". *Psychological Review, 84*. 413-451.

Golden, R.M. (1986). "A developmental neural model of visual word perception". *Cognitive Science, 10*(3), 241-276.

Proulx, R. (1986). *Etude des proprietes de selectivite d'un modele parallele de memoire associative applique au probleme de la reconnaissance des caracteres alphabetiques*. Unpublished doctoral dissertation, Universite de Montreal, Montreal.

Proulx, R. (1987, August). *Selectivity problems in the Anderson's BSB Model*. Paper presented at the meeting of the Society for Mathematical Psychology, Berkeley, CA.

Rumelhart, D.E. & McClelland, J.L. (Eds.) (1986). *Parallel distributed processing: Explorations in the microstructure of cognition, Volume I: Foundations*. Cambridge, MA: Bradford Books/ MIT Press.

THE EFFECT OF THE SLOPE OF THE ACTIVATION FUNCTION ON THE BACK PROPAGATION ALGORITHM

Ali Rezgui and Nazif Tepedelenlioglu
Electrical and Computer Engineering Department
Florida Institute of Technology
Melbourne, Florida 32901

***Abstract**-We investigate in this paper the effect of the slope of the activation function(the node nonlinearity) on the performance of the back propagation algorithm in training a multilayer perceptron. The main conclusion of the paper is that the back propagation algorithm can be made more robust by not only making the weights adaptive, but the slopes of the nonlinearities as well.*

INTRODUCTION

The back propagation algorithm[1],[2] is an adaptive procedure which is widely used in training a multilayer perceptron for a number of classification applications in areas such as speech and image recognition.

The network is presented a set of input vectors together with the corresponding desired output patterns. At each iteration step, the algorithm compares the actual output with the desired output and calculates the error. The error is then used to update the weights in the output layer in the desired direction; and then back propagated to the hidden layers to update the remaining weights.

The back propagation algorithm suffers from a major drawback however. The convergence of the algorithm is very sensitive to the initial set of weights. Very often, one may find that the algorithm fails to converge. This phenomenon is usually explained as the algorithm " getting stuck in a local minimum" of the error surface[1]. While this may be true for some cases, there is another phenomenon which occasionally prevents convergence.

Because the unsaturated range of the nonlinearity (dynamic range) is predetermined and kept fixed throughout; in the classical version of the algorithm it is very common that the initial set of weights causes the nonlinearities to saturate and therefore the derivative of the nonlinearities becomes so small that the weights update very slowly, if at all, which causes the algorithm to fail to converge or to converge after a relatively large number of iterations.

This problem makes the back propagation algorithm very unpredictable. It is the experience of the authors that it takes several trials to find an initial set of weights that would be suitable for convergence. Except for a new algorithm that uses Kalman filtering to train the perceptron[4], little attempt, if any, has been made to improve the back propagation algorithm so that the convergence would be less dependent on the initial set of weights.

In the next section, we introduce a modification to the classical back propagation algorithm which updates the slopes of the nonlinearities as well as the weights, hence prevents this saturation phenomenon. The remainder of the paper is devoted to experimental results.

SLOPE OF THE NODE NONLINEARITY

While it is standard procedure to use the so called sigmoid as the node nonlinearity[1], we shall depart from this and use the saturating linear soft limiter (SLSL) given by

$$f(x) = \begin{matrix} 1 & \text{if} \quad x > 1/a \\ ax & \text{if} \quad |x| \leq 1/a \\ -1 & \text{if} \quad x < -1/a \end{matrix} \tag{1}$$

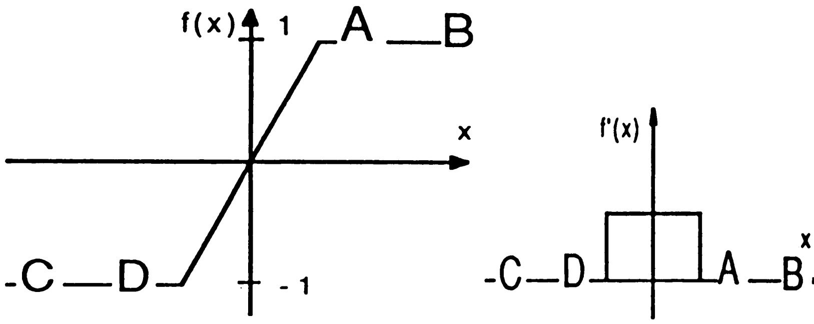
instead , because of its advantages mentioned in [5].

To illustrate the effect of the slope of the nonlinearity, a two layer perceptron with two input nodes, two nodes in the hidden layer and one output layer will be used to solve the XOR-problem[3].

The possible inputs, the desired output responses, and symbols to represent the actual output of the net for the XOR-problem are shown in table 1.

x1	x2	DES. OUTPUT	symbolized output
1	1	-1	A
1	-1	1	B
-1	1	1	C
-1	-1	-1	D

Table 1. XOR-problem



(a) (b)

Fig 1. (a) node nonlinearity, (b) Derivative

Since the initial set of weights is randomly chosen, it is common that the output of the linearcombiner for a given slope setting saturates the nonlinearity as shown in Fig 1.(a), where x is the input to the nonlinearity and the symbols are the actual outputs of the net for the four possible inputs of an XOR-problem for a specific case. It is seen that the slope of the nonlinearity is so steep that all possible inputs to the nonlinearity fall in the saturated range.

Fig 1.(b) shows that when such a case occurs, the derivative of the nonlinearity is very small(instead of setting the derivative to zero in the saturation regions as shown in the figure, we set it to a small ϵ as explained below) and therefore the weights will update very slowly which might cause the algorithm not to converge.

On the other hand, if the slope is made very small so as to have all the inputs to the nonlinearity, however big, fall in the dynamic range, it might be so small that it would take a relatively large number of iterations for the actual outputs to move out from the dynamic range and fall in the correct saturated regions.

Therefore, there must be an optimal value for the slope of the nonlinearity for different initial sets of weights.

Since there is no way to determine the optimal slope, it would be propitious to modify the back propagation algorithm such that the slopes of the nonlinearities used in the net are made adaptive as well.

In order to make the slope of the nonlinearity adaptive, the SLSL given by (1) is viewed as a function of two variables : the input x and the slope a.

Therefore the derivatives of (1) with respect to x and a are given by

$$f'_x(x,a) = \begin{cases} a & \text{if } |x| \le 1/a \\ \epsilon & \text{if } |x| > 1/a \end{cases} \tag{2}$$

and

$$f'_a(x,a) = \begin{cases} x & \text{if } a \le |1/x| \\ \epsilon & \text{if } a > |1/x| \end{cases} \tag{3}$$

respectively, where ϵ is a suitably small positive number, included so as to have the derivative never equal to zero so that a small updating occurs even when the nonlinearity is saturated.

Let the sequence of input vectors and the corresponding desired output vectors be $\mathbf{X}_p$, $p=1,2,\ldots,M$ and $\mathbf{D}_p$, $p=1,2,\ldots,M$, respectively.

The purpose is to determine adaptively the weight vectors $\mathbf{W}_j$ and the slope of the nonlinearity a_j that belong to each neuron (node) so as to minimize the mean squared error

$$E_p = \| \mathbf{Y}_p - \mathbf{D}_p \|^2$$

where $\mathbf{Y}_p$ is the actual net output corresponding to the input $\mathbf{X}_p$.

The new algorithm does this following the steps below:

1. Start with a random set of weights and a random set of slopes.
2. Calculate $\mathbf{Y}_p$ by propagating $\mathbf{X}_p$ through the network.
3. Calculate E_p.
4. Adjust slopes by

$$a_j(t+1) = a_j(t) + \beta \Delta_{aj} + \rho (a_j(t) - a_j(t-1)) \tag{4}$$

where $a_j(t)$ is the slope at time t belonging to node j, β is the step size, and ρ is a momentum term and

$$\Delta_{aj} = f_a' (x_{pj}, a_j) (d_{pj} - y_{pj}) \tag{5}$$

if j is an output node and

$$\Delta_{aj} = f_a' (x_{pj}, a_j) \Sigma_k \Delta_{wk} w_{jk} \tag{6}$$

if j is a hidden node.

This step is the only addition to the classical algorithm and its derivation follows exactly the same steps used in deriving (7) in the literature. It can be seen in (4) that the slopes are updated in a very similar manner to the way the weights are updated in the classical back propagation algorithm.

5. If $a_j(t+1) < a_{min}$ then $a_j(t+1) = a_{min}$

where a_{min} is a small positive number which is used to prevent the slopes from taking very small or negative values.

6. Adjust weights by

$$\mathbf{W}_j(t+1) = \mathbf{W}_j(t) + \mu \Delta_{wj} \mathbf{X}_j + \alpha (\mathbf{W}_j(t) - \mathbf{W}_j(t-1)) \tag{7}$$

where $$\Delta_{wj} = f_x' (x_{pj}, a_j) (d_{pj} - y_{pj}) \tag{8}$$

if node j is an output node and

$$\Delta_{wj} = f_x' (x_{pj}, a_j) \Sigma_k \Delta_{wk} w_{jk} \tag{9}$$

if node j is a hid den node.

7. Repeat by going to step 2.

It must be noted that the new algorithm has only two new terms compared to the classical algorithm,Δ_{aj} and $f_a'(x_{pj},a_j)$. The term $f_a'(x_{pj},a_j)$ is the derivative of the nonlinearity with respect to the slope and has the same computational complexity as the term $f_x'(x_{pj}, a_j)$, and the term Δ_{aj}, as seen in (6), is related to Δ_{wj} which is used to update the weights. Therefore, the new algorithm does not have any more computational burden than the classical one since the Δ_{wj}'s, used to update the weights, are also used to update the slopes.

SIMULATION RESULTS

Both the classical and the new algorithm are used to train a two-layer neural network, as described in the previous section, to solve the XOR-problem. Both algorithms are run with the same sequence of inputs, the same initial set of weights, the same parameters : $\alpha = .5$, $\mu = .7$, and the same initial set of slopes $a_j = 1$ for all the nodes. The new algorithm has the additional parameters β = .15 and ρ = .05.

The input data are presented in sequence together with the desired output to the net as shown in table 1.

Since there are four different pairs of inputs in the XOR-problem, the MSE between the desired output and the actual output of the net is computed at every four iterations.

The results of the simulation are plotted in Fig 2. It is seen that while the new algorithm converges in about 50 iterations, the classical algorithm does not converge even after 300 iterations. It must be emphasized here that this behavior for the classical back propagation algorithm occurs sufficiently often to justify the modification.

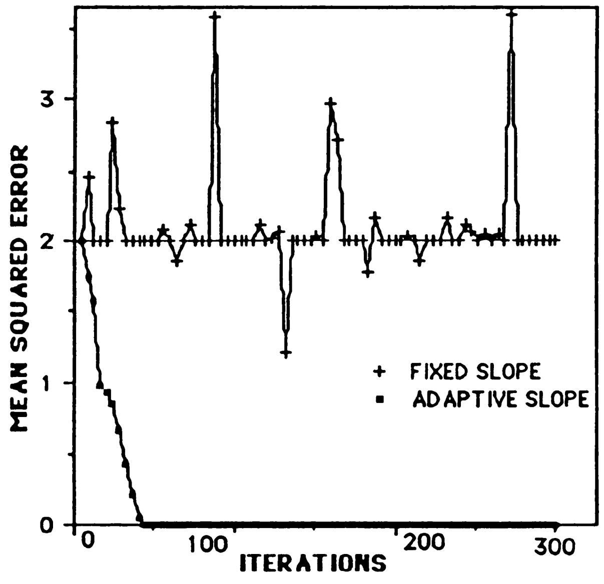


Fig 2. Learning curves for the XOR-problem.

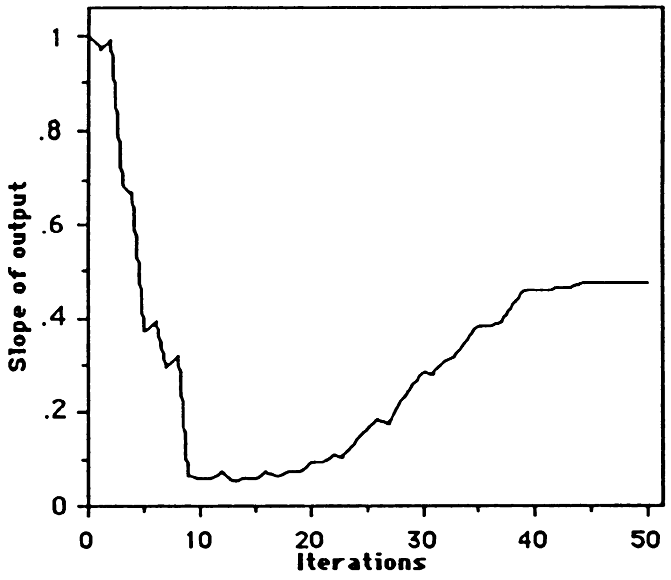


Fig 3. slope of the output nonlinearity.

The value of the slope of the output node nonlinearity is shown in Fig 3. It seen that the slope decreases from its initial value to a very small value, which increases the dynamic range and therefore allows the weights to update rapidly in the initial stages of the training period. As the algorithm begins to converge, however, we see from the figures that the slope begins to increase, thus decreasing the dynamic range and therefore limiting adaptation.

REFERENCES

[1] R. P. Lippman, "An Introduction to Computing with Neural Nets", IEEE ASSP Magazine, Vol. 4, Number 2, April 1987.

[2] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing*, Vol I, Cambridge, MA, MIT Press, 1986.

[3] B. Widrow, R. G. Winter and R. A. Baxter, "Layered Neural Nets for Pattern Recognition", IEEE Trans. Acous., Speech, Signal Processing, Vol. 36, Number 7, July 1988.

[4] R. Scalero and N. Tepedelenlioglu, " A Fast New Algorithm for Training Feed-Forward Neural Networks " submitted to the IEEE Trans. Acous., Speech, Signal Processing, May 1989.

[5] N. Tepedelenlioglu and Ali Rezgui, " The Effect of the Activation Function on the Performance of the Back Propagation Algorithm ", IEEE International Conference on Systems Engineering, Dayton Ohio, August 24-26, 1989.

LEARNING WITH THE OPTIMUM PATH PARADIGM

Samir I. Sayegh
Physics Department
Purdue University
Ft Wayne, IN 46805-1499
sayegh@ipfwcvax.bitnet

THE FEED-FORWARD PHASE

The proposed optimum path paradigm (OPP) works as follows. The input is specified by a string of $\underline{w}$ symbols taken from an alphabet A = $\{a, b, c, \ldots\}$, containing $\underline{g}$ letters. The output is a string of $\underline{w}$ symbols in alignment with the $\underline{w}$ input symbols. The output symbol corresponding to a given input symbol can be taken from a collection of allowable output symbols for each input symbol. For example, the input symbol a would admit output symbols a1, a2, ..., aq. The direct problem is to find the correct string of output symbols for each admissible input string. The inverse problem, i.e., learning, consists in finding the association laws or dynamics, by observing input/output pairs, thus allowing oneself to predict future outputs.

The input string is written horizontally and below each input symbol one writes a layer of all possible output symbols corresponding to the input symbol under consideration. Specifying the output string consists in making a choice of an output symbol per layer. If one thinks of joining each output symbol to the next chosen one as a means of specifying the correct output string, one then gets a path connecting the first layer to the last layer, and containing one node per layer.

All symbols in two successive layers are assumed to be connected with a weight assignment for each such connection. This is the same connectivity as for the backpropagation algorithm BP [5,8]. The traversal of the network is specified by finding the path of least cost from first to last layer. It can be implemented with the Viterbi algorithm [7] which is itself a version of Dijkstra's algorithm [1]. The topology is identical to that of BP and has the same degree of parallelism. All nodes of a given layer compute simultaneously. Each node performs a function of q arguments, one argument corresponding to each node of the previous layer. Each such argument is a sum (instead of a product as for BP) of a value stored at the previous node and that of a weight on the connection between the two nodes. The function used here is a min function (i.e. take the smallest argument) instead of the sum followed by a sigmoid used in BP.

The specification of output described above, presumes that there is a $\underline{gq}$ x $\underline{gq}$ matrix W of weights, specifying all possible transitions between all possible pairs of output symbols. Learning deals with the problem of finding such a matrix.

THE LEARNING PROCEDURE

Learning consists in observing a number of input/output pairs and in reconstructing from them the input/output mapping. An input/output pair is equivalent to the specification of a net and an optimum path on that net. Given a collection of such nets, together with the corresponding optimum paths, one would like to reconstruct the matrix W of weights of transition from one output symbol ci to another cj.

Notice that the specification of an optimum path is equivalent to the specification of a number of inequalities, stating that the sum of the weights of the edges forming the optimum path is larger than the sum of any combination of weights corresponding to an alternate path. Knowledge of a collection of input/output pairs is therefore equivalent to the knowledge of a large number of inequalities among the entries of the matrix W. Attempting to achieve learning, i.e., determining the entries of W or one such equivalent set of entries, is tantamount to the "solution" of a large number of inequalities among the matrix elements. This is achieved by introducing a heuristics that relates the frequency of occurrence of a weight as part of a large side of an inequality to its actual value. This is quite intuitive since the larger a weight the more likely it is to occur on the larger side of inequalities involving different weights. Simulations bear this fact [6].

When translated back to the network formulation, the heuristics dictates strengthening each edge or connection that occurs on a best path. The matrix W' is therefore constructed by successive incrementation of the entries corresponding to transitions occurring on best paths. This is a form of Hebbian learning [2] in the context of a different feed-forward paradigm for supervised learning.

EXAMPLE

The task to be learned here will be assumed to be implementable through minimization and that a transition matrix W actually exists to represent the given mapping through minimization as discussed. Learning will then consist in recovering an equivalent matrix W', through observation of input/output pairs generated through the use of W. One then uses W' to generate outputs and compares them to the one generated with W. If they agree, learning has been successful.

It is not necessary to have a specific model for the objects that W represents. If such a model is considered useful at this stage, one could use that of a robot arm. In this particular model we assume that we have three positions p1, p2 and p3, that it is desirable to reach. Each such position is realizable through two different configurations of the joint variables of the robot arm. We thus have 6 different possible configurations and the costs of transition from any one configuration to another is entered in the 6 x 6 matrix, W.

Through the use of W, one plans the optimum path for any prescribed trajectory. By considering the actual trajectories followed and applying the heuristics described above, one then generates a new matrix W' of inferred costs of transition. If learning is successful, the use of W' for path planning will result in the same paths as the use of W.

The above is illustrated by the following example:

The matrix W is given by:

$$W = \begin{pmatrix} 0 & 1 & 2 & 1 & 2 & 3 \\ 1 & 0 & 1 & 3 & 1 & 2 \\ 2 & 1 & 0 & 1 & 2 & 2 \\ 1 & 3 & 1 & 0 & 3 & 2 \\ 2 & 1 & 2 & 3 & 0 & 1 \\ 3 & 2 & 2 & 2 & 1 & 0 \end{pmatrix}$$

We then consider 3-position sequences. For each such sequence we generate the optimum path. By applying OPP learning we obtain the following matrix W':

$$W' = \begin{pmatrix} 2 & 0 & 0 & 5/2 & 0 & 0 \\ 0 & 4 & 7/2 & 0 & 6 & 0 \\ 0 & 7/2 & 3 & 0 & 17/6 & 11/6 \\ 5/2 & 0 & 0 & 3 & 0 & 4/3 \\ 0 & 6 & 17/6 & 0 & 4 & 0 \\ 0 & 0 & 11/6 & 4/3 & 0 & 2 \end{pmatrix}$$

It can be verified that maximum paths using W' generates the same paths as W has. Learning was thus successful.

DISCUSSION

The present paradigm is promising for learning with connectionist networks. One reason is the speed of both feed-forward and learning phases. The fast nature of the learning phase, which processes each weight no more than once, makes it a good candidate for practical applications [4,5].

Despite the wide difference in concept between OPP and BP, there are fundamental similarities between them. A practical similarity is related to the identical topology in the feed-forward phase. A fixed architecture will run both paradigms. A theoretical similarity arises from the fact that BP performs gradient descent in the space of weights during the learning phase, thus searching for the "state of minimum energy," in a way similar to actual physical systems. In OPP, during the feed forward phase, the system looks for the path of "minimum total weight" or "least action," which again is what physical systems do. The success of both paradigms might be rooted in the simple fact that they emulate the long tested ways Nature uses to evolve.

REFERENCES

[1] Dijkstra, E. W. 1959. "A Note on Two Problems in Connection with Graphs." Numerische Math. 1, 269-271.

[2] Hebb, D. 1949. Organization of Behavior. Wiley, New York.

[3] Rumelhart, D.E. and McLelland, J.L. 1987. Parallel Distributed Processing. Cambridge, MA: MIT Press.

[4] Sayegh, S.I. 1989 "Fingering for String Instruments with the Optimum Path Paradigm" Computer Music Journal Special Issue on Connectionism (September 1989).

[5] Sayegh, S.I. and Manzor-Coats, L. 1988. "Neural Networks as an Alternative to Rule-Based Systems for Learning Spanish Phonetics," Proceedings of the International Symposium on Artificial Intelligence, Monterrey, Mexico.

[6] Sayegh, S.I. and Tenorio, M.F. 1988. "Inverse Viterbi Algorithm as Learning Procedure and Application to Optimization in the String Instrument Fingering Problem," Proceedings of the IEEE International Conference on Neural Networks, San Diego, CA.

[7] Viterbi, A.J. 1967. "Error bounds for convolutional Codes and an Asymptotically Optimum Decoding Algorithm." IEEE Transactions on Information Theory.

[8] Werbos, P. 1974. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Ph.D. thesis, Harvard University Committee on Applied Mathematics.

A FAST TRAINING ALGORITHM FOR NEURAL NETWORKS

Robert S. Scalero is with Grumman Melbourne Systems, P.O. Box 9650, Melbourne, FL 32904.

Nazif Tepedelenlioglu is with the Electrical and Computer Engineering Department of the Florida Institute of Technology, Melbourne FL 32901.

ABSTRACT

A fast new algorithm is presented for training multi-layer perceptrons as an alternative to the back-propagation algorithm. This new algorithm reduces the required training time considerably and overcomes many of the shortcomings presented by the conventional back-propagation algorithm.

The algorithm uses a modified form of the back-propagation algorithm to generate error signals with respect to the summation outputs (inputs to the nonlinearities). These error signals are used to estimate values at the summation outputs that will improve the total network error. The determination of weights using these summation output estimates is thereby reduced to a linear problem which may be solved by using a Kalman filter at each layer.

The new algorithm, as shown in this paper, shortens the training time by several orders of magnitude for the pattern recognition type problem considered. In some cases improvement ratios of the new algorithm over the back-propagation algorithm run higher than 10,000. The new algorithm is also less effected by the choice of initial weights and set up parameters.

INTRODUCTION

The purpose of this paper is to present a novel algorithm for rapid training of feed-forward multilayer perceptrons. It is not our intent to derive the algorithm here, but to state the algorithm and present experimental results in pattern recognition. For the full derivation of the algorithm, and additional experimental results, the reader is refered to [1], [2].

The back-propagation algorithm [3], [4], which is a generalization of the LMS algorithm that minimizes the mean squared error between the desired and the actual outputs of the network with respect to the weights, has become the standard way of training multilayer perceptrons. The algorithm, however, suffers from a number of shortcomings. Aside from the large amount of time required for the algorithm to converge, it has the added disadvantage of being very sensitive to initial weights and set up parameters.

It is the presence of the nonlinear functions in the network that prevents us from directly applying the standard least squares adaptive filtering techniques which are known to have rapid convergence properties. If the nonlinear neural network problem can be partitioned into linear and nonlinear parts, a wide range of techniques would be at our disposal for solving this problem. In contrast to the standard back-propagation algorithm, the new algorithm uses a modified form of the back-propagation algorithm to minimize the mean squared error with respect to the summation outputs (inputs to the nonlinearities). Error signals are generated and used to estimate values at the summation outputs that will improve the total network error. By specifying improved summation outputs this way, the determination of the weights is thereby reduced to a linear problem. The solution of the weight vector at each node, is the solution of the deterministic normal equation [4] whose input variables are the estimated summation outputs, along with the input vector to the respective node. Solving the normal equation is implemented recursively by using a Kalman filter [5] at each layer.

THE ALGORITHM

1. Initialize
   Equate the node offset $x_{j-1,0}$ of every node to some non-zero constant for layers j = 1 through L, randomize all weights in the network, and initialize the inverse matrix $\mathbf{R}^{-1}$.

2. Select training pattern
   Randomly select an input/output pair to present to the network. The input vector is $\mathbf{X}_0$ and desired network output vector is $\mathbf{O}$.

3. Run selected pattern through the network
   For each layer j from 1 through L, calculate the summation output

   $$y_{jk} = \sum_{i=0}^{N} x_{j-1,i} \; w_{jki} \quad \text{and the function output} \quad x_{jk} = f(y_{jk}) = \frac{1 - \exp(-a\, y_{jk})}{1 + \exp(-a\, y_{jk})}$$

   for every node k. N is the number of inputs (not including the offset) to a node, and constant a is the sigmoid slope.

4. Invoke Kalman filter equations
   For each layer j from 1 through L, calculate $\mathbf{A}_j = \mathbf{R}^{-1}_j \mathbf{X}_{j-1}$, calculate the Kalman gain $\mathbf{K}_j = \mathbf{A}_j \; [b_j + \mathbf{X}^T_{j-1} \mathbf{A}_j]^{-1}$, and update the inverse matrix $\mathbf{R}^{-1}_j = [\mathbf{R}^{-1}_j - \mathbf{K}_j \; \mathbf{A}_j] / b_j$, where b_j is the the forgetting factor for the jth layer.

5. Back-propagate error signals
   Compute the derivative of $f(y_{jk})$ using $f'(y_{jk}) = \frac{2\, a\, [\exp(-a\, y_{jk})]}{[1 + \exp(-a\, y_{jk})]^2}$.
   Calculate error signals in the output layer, for every node k, by evaluating
   $e_{Lk} = f'(y_{Lk}) \; (o_k - x_{Lk})$.
   For the hidden layers, starting at layer j = L–1 and decrementing through j = 1, find error signals by solving

   $$e_{jk} = f'(y_{jk}) \sum_{i} e_{j+1,i} \; w_{j+1,i,k}$$ for every node in the jth layer.

6. Find the desired summation output
   Calculate the desired summation output at the Lth layer by using the inverse function

   $$d_k = \frac{1}{a} \ln \left[\frac{1 + o_k}{1 - o_k} \right]$$ for every kth node in the output layer.

7. Update the weights
   The weight vectors in the output layer L, are updated by $\mathbf{W}_{Lk} = \mathbf{W}_{Lk} + \mathbf{K}_L \, (d_k - y_{Lk})$ for every kth node. For each hidden layer j = 1 through L–1, the weight vectors are updated by $\mathbf{W}_{jk} = \mathbf{W}_{jk} + \mathbf{K}_j \, e_{jk} \, \mu_j$ for every kth node.

8. Test for completion
   If the network has not yet converged, go back to step 2.

PATTERN RECOGNITION EXPERIMENTAL RESULTS

The following pattern recognition example illustrates the difference in training times of the two algorithms when presented with the same pattern set. The 7 x 7 pixel patterns, Fig. 1, are the inputs to a 2 layer feed-forward perceptron with 16 nodes in the hidden layer. The desired output of the network is a 2, 3, or 4 bit binary word depending upon the number of patterns used to train the network. As stated for the examples in [1] and [2], both algorithms were started from exactly the same initial weights, and patterns were presented to both algorithms in the same order. Fig. 2 shows the mean squared error vs. the iteration number for both algorithms during training.

Table 1 presents numerically, the performance comparisons of the two algorithms plotted in Fig. 2. These comparisons take into consideration the computational efficiency of each algorithm as well as the number of iterations required for the algorithm to reach a specified mean squared error. The result is a time ratio of the two algorithms when run on a sequential machine. The computation ratio is the number of operations required by the back-propagation algorithm, divided by the number of operations required by the new algorithm per iteration. This ratio, multiplied by the iteration ratio (iterations required by the new algorithm divided by iterations required by the back-propagation algorithm) produce the total improvement ratio. A mean squared error convergence of slightly less than 0.25 was chosen since this value is the minimum that can be used and still produce correct results, assuming that the outputs are eventually passed through a hard limiter to produce a binary word.

RESULTS & CONCLUSIONS

As illustrated in Fig. 2 and table 1, the new algorithm converges much faster than the back-propagation algorithm for pattern recognition type problems. Even with its additional arithmetic operations, improvement ratios of the new algorithm over the back-propagation algorithm ran higher than 10,000 in some cases and it is not unlikely that this number may be further increased by considering patterns with higher resolution.

The new algorithm is also more predictable in its training. In Fig. 2, we notice that the back-propagation algorithm tends to reach a certain mean squared error and remain there for quite a while making little or no progress. At some point, it either rapidly converges, or jumps to a new level where it would again make little or no progress for quite a while. In contrast, the new algorithm continues to make steady progress toward improving the mean squared error throughout the training period.

REFERENCES

[1] R. S. Scalero and N. Tepedelenlioglu, "A Fast New Algorithm for Training Feed-Forward Neural Networks", Submitted for publication to IEEE Trans. Acoust., Speech, Signal Processing on May 17, 1989.

[2] R. S. Scalero and N. Tepedelenlioglu, "A Fast New Algorithm for Training Feed-Forward Neural Networks: Application Pattern Recogmition", Submitted for publication to IEEE Trans. Acoust., Speech, Signal Processing on July 14, 1989.

[3] D. E. Rumelhart and J. L. McClelland, "Parallel Distributed Processing", Vol. I, Cambridge, MA, MIT Press, 1986

[4] S. S. Haykin, "Adaptive Filter Theory", Prentice-Hall, 1986, pp. 312-314

[5] S. S. Haykin, "Adaptive Filter Theory", Prentice-Hall, 1986, pp. 381-390

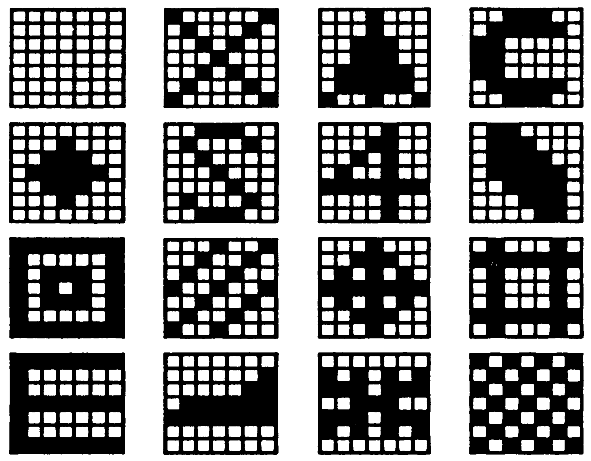


Figure 1 7 x 7 patterns used for training.

7 x 7 PATTERNS	4 patterns	8 patterns	12 patterns	16 patterns
16 INPUT NODES	(2 Outputs)	(3 Outputs)	(4 Outputs)	
NEW ALGORITHM Iterations	6	9	18	24
BACK-PROP Iterations	> 300000	> 300000	66500	50400
ITERATION RATIO	> 50000	> 33333	3694.44	2100.00
COMP. RATIO	0.3391	0.3458	0.3523	0.3523
TOTAL IMPROVE.	> 16953	> 11525	1301.64	739.88

Table 1 Improvement ratio of the new algorithm over back-propagation with the ***MSE convergence set at 0.25***.

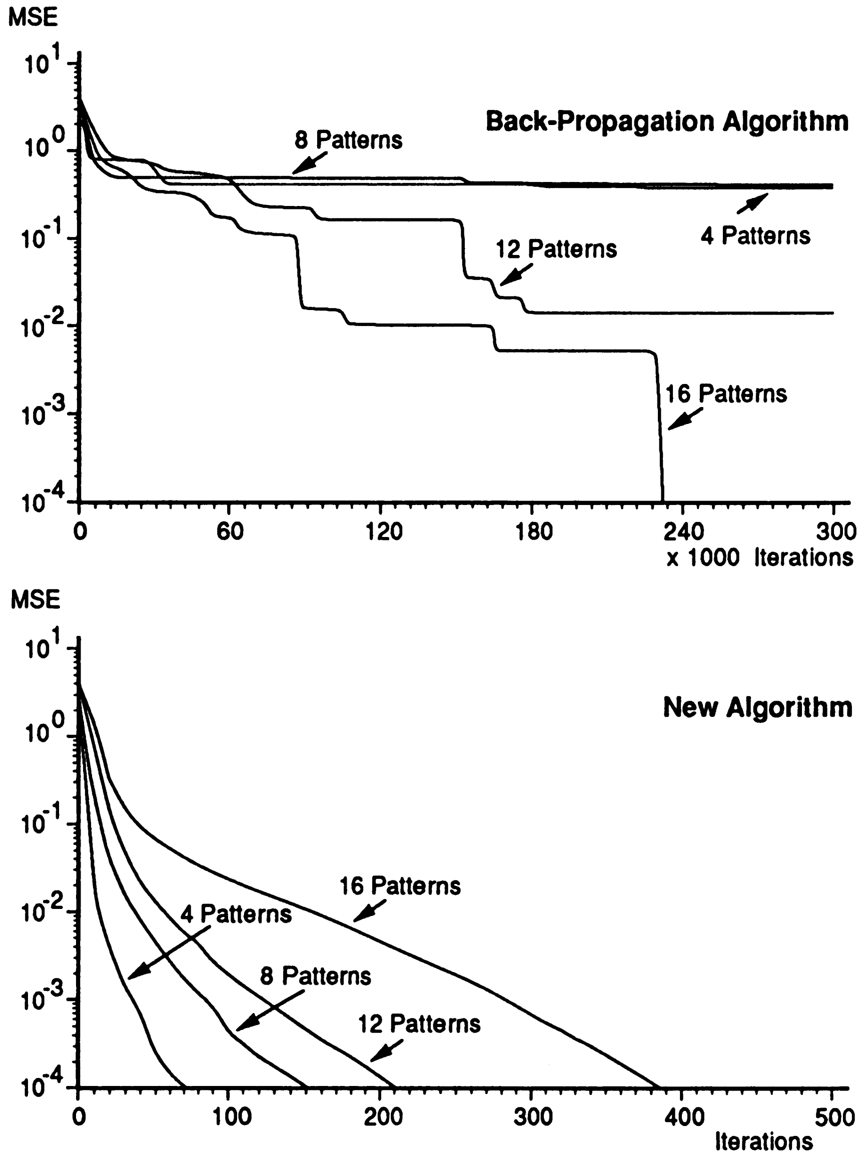


Figure 2 Learning curves for a 2 layer pattern recognition network with ***16 nodes*** in the hidden layer. 7 x 7 patterns were used for training.

Recurrent Networks Adjusted by Adaptive Critics

Jürgen Schmidhuber
Institut für Informatik
Technische Universität München
Arcistr. 21, 8000 München 2, Germany
schmidhu@tumult.informatik.tu-muenchen.de

Abstract

This paper is concerned with spatiotemporal credit assignment in recurrent networks by local computations only. An approach to on-line reinforcement learning and an approach to on-line supervised learning are considered. Common to both is that state transitions in a completely recurrent network are observed by a second non-recurrent network which receives as input the complete activation vectors of the recurrent one. The output of the second network serves to provide error information for the recurrent network.

We also consider how in a changing environment a recurrent dynamic *supervised learning critic can interact with a recurrent dynamic reinforcement learning network in order to improve its performance.*

Introduction

Few learning schemes in principle are applicable to dynamic tasks with temporally varying inputs and outputs, including 'extended REINFORCE' algorithms (Williams, 1988) and the 'Neural Bucket Brigade Algorithm' (Schmidhuber, 1988) for reinforcement learning. For supervised (and non-local) learning in completely recurrent dynamic networks see e.g. (Pearlmutter, 1988), (Williams and Zipser, 1988), (Rohwer, 1989), (Gherrity, 1989).

The most important aspect of the algorithms described in this paper is their locality in both space and time. 'Local in time' is meant to say that weight changes should take place continually, and that changes should depend only on information about activations and weights from a fixed recent time interval. (In this paper the recent time interval degenerates to the last time tick for discrete time versions.) This contrasts to weight changes that take place only after externally defined episode boundaries, which require additional *a priori* knowledge. 'Local in space' is meant to say that for arbitrary network size changes of a unit's weight vector should depend solely on information from connected units, and that the update complexity per time tick should be only proportional to the dimensionality of the weight vector. This implies that for a completely recurrent network the weight update complexity at a given time is $O(n^2)$ where n is the number of units. (In this paper the 'connected units' are the set of source units which provide input to a given unit, plus one more unit from a second network which is used to compute error information. The additional unit can be thought of as 'connected in a special way'.)

In both cases described herein there is a primary network which has to be adjusted such that it shows desireable input-output behavior. The primary network is a completely recurrent dynamic one, where certain units serve as input units and others serve as hidden units or output units. In both cases there also is a second network (termed 'the critic' hereafter) which has to be capable of static pattern association. It receives as input the whole (or parts of the) current state of the primary network. Both networks are adaptive: The output of the critic serves to adjust the primary network. Various methods can be used to adjust the critic. Two examples of pairs of interacting networks will be described in the next two sections.

Reinforcement Learning in Dynamic Recurrent Networks

We propose the discrete time version of an algorithm for adjusting a recurrent network in order to let it solve tasks by delayed reinforcement learning (i. e. tasks where an external teacher only indicates once a while whether the system is in a desireable state or not, *without providing detailed knowledge about the desired outputs at each time tick*). Essentially this algorithm can be viewed as an application of 'Temporal Difference Methods' (TD-methods) (Sutton, 1988) to the temporal evolution of recurrent networks. The fully recurrent primary network consists of linear input units and binary probabilistic non-input units. We consider the case where the learning phase is dividable into 'episodes'. An episode starts with the initialization of the system's activations and is finished when the final reinforcement R becomes known. Here is a description of the algorithm:

First all weights are randomly initialized with real values.

For all episodes:

In the beginning of each episode the activations of input units of the recurrent network are initialized with values determined by the environment, and the activations of hidden and output units are initialized with 0. *For all time ticks, until there is external reinforcement R (a real number) indicating failure or success :*

At a given time tick t:

1. The critic (e.g. a back-propagation network with one output) receives as input the complete activation vector $x(t-1)$ of all units of the recurrent network. So the dimensionality of the input vector of the critic equals the number of units in the recurrent network. Its one-dimensional output r is interpreted as a prediction of the final reinforcement to be received in the future (Barto et al., 1983)(Sutton, 1984)(Anderson, 1986).

2. The recurrent network performs one update-step: Each probabilistic non-input unit i sums its weighted inputs, this sum is passed to the logistic function which gives the probability that the activation $x_i(t)$ becomes 1, *or* 0, *respectively. Each unit i also stores its last activation $x_i(t-1)$. Output units may cause an action in the environment, this may lead to new activations for the input units (external feedback).*

3. If there is external reinforcement R (this means the end of the current episode) then the variable r' is defined to be equal to R.

Otherwise r' is defined to be a new estimation of final reinforcement, obtained by letting the critic judge the new state of the recurrent network.

Using its static learning algorithm (e.g. the generalized delta rule) the critic associates the last activation vector of the recurrent network with r', thus 'transporting expectation back in time' for one time step.

4. Each directed weight $w_{ij}(t)$ from unit i to unit j of the recurrent network is immediately altered according to $\Delta w_{ij}(t) = \lambda(r' - r)x_i(t-1)x_j(t)$ (with λ being a positive constant), thus encouraging (or discouraging, respectively) the last transition.

State transitions from states associated with low expectation of reinforcement leading to states with a higher evaluation are encouraged. State transitions from states associated with high expectation of reinforcement leading to states with a lower evaluation are discouraged. So the learning algorithm implements Samuel's principle for delayed reinforcement, as described in the context of learning to play checkers (Samuel, 1959): 'We are attempting to make the score, calculated for the current board position, look like that calculated for the terminal board position of the chain of moves which most probably will occur during actual play.'

The algorithm described above has been applied successfully to a 'delayed XOR' problem (Schmidhuber, 1989). An interesting aspect is that a *linear* critic was sufficient, while the task to be solved was of the 'non-linearily separable' type.

A few modifications were made to the algorithm, in order to make it applicable to tasks where the goal is to maximize the duration of an episode. These modifications are motivated by the work of Sutton (Sutton, 1984), who studied and generalized Samuel's principle for the case of networks consisting of single units. Sutton introduced a discount rate $0 < \gamma < 1$ in order to give more weight to reinforcement to be received in the nearer future. Following Sutton we modified step 3 of the algorithm. The corresponding statement now says:

Otherwise r' is defined to be a new estimation of final discounted reinforcement, obtained by letting the critic judge the new state of the recurrent network and multiplying the new value with γ .

We also modified the learning rule for the recurrent net (step 4 of the algorithm) such that unlikely transitions were credited more strongly (Anderson, 1986):

$\Delta w_{ij}(t) = \lambda(r'-r)(x_i(t-1)(x_j(t) - P(x_j = 1 \mid x(t-1), w(t-1))$, where $w(t-1)$ is the last weight vector.

We tested the modified algorithm on the pole balancing task described in (Anderson, 1986). (Detailed parameter settings and results are given in (Schmidhuber, 1989).) Following Anderson we made the task more difficult than the similar task described in (Barto et al., 1983), where a prewired decoder was used to provide binary multi-dimensional input to a single-unit 'network', with all components being zero except for one. Instead the input was real valued, and additionally scaled in an asymmetric manner, in order to force the system to discover a non-trivial internal representation by itself.

It is worth noting some differences to Andersons system (Anderson, 1986). In contrast to Anderson we did not use a back-propagation network but a single *linear* unit for the critic. We also did not use a static feed-forward network for computing output actions, but a continually running recurrent network[1]. Since one update-step of the recurrent network also involves a change of input activations due to the external loop through the environment, there is a delay of at least 2 time ticks between inputs that have to be transformed in a non-linear fashion, and the corresponding actions. A new input can be available before the response to the last one is computed.

An Approach to Supervised Learning in Recurrent Networks

In this section we describe an idea for a local learning scheme for supervised learning in continually running recurrent networks, where each non-input unit at each time can receive an individual error signal. The method is based on back-propagation (BP) (Werbos, 1974) in recurrent networks unfolded in time (Rumelhart et al., 1986), the global error measure to be minimized is the sum of all errors received at the non-input units over time. The important difference is that the method is local in space *and* time, while conventional BP is not. In conventional BP each unit needs a stack for remembering past activations which are used to compute contributions to weight changes during the error propagation phase. Starts and ends of sequences have to be indicated by an external teacher. (Williams' and Zipser's on-line algorithm (Williams and Zipser, 1988) is local in time, but not in space.)

Instead of allowing unlimited storage capacities in form of stacks we introduce a second adaptive, but static network (again termed the 'critic'). The dimensionality of its output now equals the number of non-input units of the recurrent net. Its task is to associate states of the recurrent (primary) network with error-vectors. The behavior of both interacting networks can be described like this:

Activations spread through the primary network in the same manner as with conventional BP. At each discrete time tick the critic receives as input the state vector of the non-input units of the primary network. The sum of the critics output and the new error which may have been observed at certain output-units is used as an error-vector. This error-vector is propagated backwards through the primary network, but only one step 'back in time'. (So each unit of the primary network has to store only its last activation.) The involved weights are changed immediately afterwards, assuming that the learning rate is sufficiently small to avoid instabilities.

The new error vector received at the non-input units after the one-step-back propagation phase becomes associated with the last state of the primary network. This association has to be done by the static learning algorithm of the critic, which can be a Boltzmann machine, or a feedforward BP network, or something else.

A critical assumption of this scheme is that the state of the non-input units at a given time uniquely represents the history which led to this state. Two different histories leading to the same internal state can not be distinguished by the critic. In such cases it is likely that incorrect error vectors are one-step-back-propagated during further training. The self-healing effect could be that weight modifications caused by this process lead to new errors which in turn split 'critical' states into two or more distinguishable states representing different histories. However, at the moment the precise nature of the interactions between two networks like those described above is not clear.

The advantage of the scheme is that it is both local in space and local in time: At every time tick the system in principle performs the same local operations, there is no need for storing past activations (except for the last ones), and there is no such thing as epoch boundaries.

For several reasons the method does not implement exact gradient descent (in the sense of (Rumelhart et al., 1986) where epoch boundaries are used.). Two of them have been mentioned above: At every time tick there are weight changes, and different histories leading to the same state will cause incorrect error vectors. Another (pragmatic) reason is that the critic often will not exactly mirror the relations between primary states and error vectors, since its learning algorithm will not be perfect either. 'Similar' primary states will be blamed with 'similar' error-vectors, where the measure of similarity depends on the complexity of the critic. It remains to be verified whether such a learning scheme will face serious problems or whether the inertia of the static network could even lead to beneficial effects, comparable to the effects induced by momentum terms in conventional BP. In some preliminary experiments with a constrained *linear* critic (adapted by the delta-rule) the system sometimes learned but more often failed to learn a dynamic task (the dynamic delayed XOR problem described in (Williams and Zipser, 1988)). An interesting point is, again, that the linearity of the critic did not necessarily prevent the recurrent network from sometimes solving its non-linear task.

[1] We also did not use different learning rules for hidden and output units.

Two Interacting Recurrent Networks

Why should not the critic's own output *directly* depend on past states of the recurrent reinforcement learning system? Although the primary network is able to memorize information about past states by means of its recurrent links, one should expect advantages by introducing a continually running 'self-supervised' recurrent critic. In the sequel we informally describe one scenario for such a system consisting of two interacting recurrent networks. The basic principle is similar to the one of the algorithm described in the first section:

There is a continually running recurrent primary network with external and internal feedback, and a critic whose task is to predict the sum of (discounted) reinforcement to be received in the future. However, now the critic itself is a continually running recurrent network whose input at a given time is the complete current state of the primary network (including the current environmental input). One of the critic's non-input units is interpreted as the predictive output.

The desired value for the critics output unit at a given time tick is given by the sum of the external reinforcement and its own (discounted) output at the next time tick.

The critic's error also is the reinforcement for the primary network's reinforcement learning algorithm.

The latter needs to consider only the last and the present state, as above. But it also might be an on-line version of Williams 'extended REINFORCEMENT' algorithm. The learning algorithm for the critic should be local in time, at least. So Williams and Zipsers algorithm (Williams and Zipser, 1988) or Gherrity's slightly more general version (Gherrity, 1989) are promising candidates.

Again the computation of error signals for the critic's output is very much inspired by Sutton's TD-methods. TD methods, however, require two successive predictions during the same time tick in order to remove dependencies on weight changes. Since the recurrent critic's output already depends on past states (by means of its internal feedback) and also on past weights, the scheme described above makes only one critic update at a time.

References

Anderson, C. W. (1986). *Learning and Problem Solving with Multilayer Connectionist Systems.* PhD thesis, University of Massachusetts, Dept. of Comp. and Inf. Sci.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13.

Gherrity, M. (1989). A learning algorithm for analog fully recurrent neural networks. In *IJCNN International Joint Conference on Neural Networks, Vol 1.*

Kindermann, J. and Linden, A., editors (1989). *Proceedings of Workshop 'Distributed Adaptive Neural Information Processing', St.Augustin, 24.-25.5, to appear.*

Pearlmutter, B. A. (1988). Learning state space trajectories in recurrent neural networks. Technical report, Dept. of Comp. Sci., Carnegie Mellon Univ., Pittsburgh.

Rohwer, R. (1989). The 'moving targets' training method. In (Kindermann and Linden, 1989).

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing*, volume 1. MIT Press.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, 3.

Schmidhuber, J. H. (1988). The neural bucket brigade. In Pfeifer, R., Schreter, Z., Fogelman, Z., and Steels, L., editors, *Proceedings of the International Conference 'Connectionism in Perspective', Zürich, Switzerland*, Amsterdam: Elsevier.

Schmidhuber, J. H. (1989). Networks adjusting networks. In (Kindermann and Linden, 1989).

Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts, Dept. of Comp. and Inf. Sci.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3.

Werbos, P. J. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University.

Williams, R. J. (1988). Toward a theory of reinforcement-learning connectionist systems. Technical Report NU-CCS-88-3, College of Comp. Sci., Northeastern University, Boston, MA.

Williams, R. J. and Zipser, D. (1988). A learning algorithm for continually running fully recurrent networks. Technical Report ICS Report 8805, Univ. of California, San Diego, La Jolla.

Speeding up Back Propagation by Gradient Correlation

David V. Schreibman
Grumman Data Systems
McLean, Virginia
and
Eugene M. Norris
Department of Computer Science
George Mason University

Problem Statement:

Observations of the behavior of the conventional back propagation algorithm [1] indicates that the ability to adjust the learning rate parameter η and the momentum parameter α dynamically is highly desirable for at least two reasons: efficiency and quality of learning. The efficiency issue is one of keeping η and α as large as possible in order to minimize time required to learn to a specified error level. The quality issue is one of keeping η and α small enough to allow non-oscillatory descent into deep, possibly small-diameter local minima, thus permitting learning to proceed to a small error tolerance. Ideally, these are resolved by keeping the learning rate as large as possible without being so large as to interfere with successful, low-error learning. In practice, a strategy is needed to decide when and how to adjust the learning parameters dynamically, with no operator intervention.

Prior Results

An approach to the efficiency issue has been reported by Vogl *et al* [2]. In summary that method is as follows: (1)Weights are modified only after all patterns have been presented, and (2)The learning rate η is varied after a back propagation step according as the following forward propagation increases the error. If the error is decreased, the learning rate is multiplied by a factor $\phi > 1$. If the error increase exceeds some threshold, typically 1-5 percent over the previous error value, then the back propagation step is rejected (all weights are set back to what they were before the back propagation step), the learning rate is multiplied by a factor $\beta < 1$, and the momentum is set to zero. When a successful step is taken, the momentum α is reset to its original value. This <u>reactive</u> approach is costly in that it is likely that some back propagation steps and the following forward propagation steps will be discarded.

Gradient Correlation Approach

We examine a new approach to the efficiency and quality issues using a <u>proactive</u> dynamic decision mechanism for parameter adjustment that is based upon the correlation of present and immediate past gradient vectors.

The error gradient, a vector of weight error derivatives, points in the direction of the maximum increase of error in weight space. The correlation between the gradient values across successive training epochs indicates how stably the gradient direction changes from epoch to epoch. A large (near 1.0) positive value indicates that the present direction of weight change is substantially the same as the previous direction, while a negative value for this correlation measure indicates that the gradient is changing direçtion by a large value, greater than $\frac{\pi}{2}$.

The gradient correlation, gcor [3], is the angle between vector weight derivatives,

$$\text{gcor}(t) = \frac{<\text{wed}(t-1),\text{wed}(t)>}{\sqrt{<\text{wed}(t-1),\text{wed}(t-1)> <\text{wed}(t),\text{wed}(t)>}},$$

where wed(t) is the vector of weight error derivatives at epoch t, and <u,v> is the vector inner product, $<u,v> = \sum_i u_i v_i$. Gradient correlation value is used as a cue for automatic learning rate adjustment during back propagation. The heuristic is that when the local topography of the error surface causes a large change in the direction of the step being taken, oscillation of the gradient vector is likely, and therefore it is prudent to reduce the step size. In the method in [2], the rate of travel is adjusted only after a "bad" weight change has driven the network away from its approach to a local minimum.

Implementation

A backprop simulator [3] was modified to incorporate gradient correlation and the Vogl *et al.* method. Networks using the two modifications were trained on the same data; results were compared with those obtained from the "classical" unmodified backprop simulator.

We chose to simulate a small expert system problem of independent interest to us. This system chooses a commuter's route to work based upon the day of week, time of day, traffic accidents, and weather conditions. A brief description of this system follows.

Expert Rules

The rules for the expert system were derived from the knowledge and experience of a seasoned commuter. They are applicable for travel during morning hours between suburban Fairfax, Virginia and urban Washington, D.C. The rules recommend, for example, travel via train (Metro Rail) on weekdays when the time is early. The reason for this is that parking at the Metro station becomes scarce after 7 A.M (it is assumed that the commuter will drive his car to the station; riding by bus to the train station would eliminate the parking concern and alter the expert system's rules). Between 7 A.M. and 9 A.M., Route 50 is recommended because it's too late to find a parking space at the train station, yet too early to drive on Route 66 because of time-of-day-related vehicle occupancy (HOV) restrictions -- it is assumed that the commuter travels alone. Finally, Route 66 is recommended after 9 A.M. because HOV restrictions are lifted and traffic on Route 66 is generally better than on Route 50.

Also considered are possible travel delays due to accidents which may have occurred on the available routes: accidents on the Metrorail in fine weather, for example, will result in a recommendation for Route 50 before 9 A.M. and Route 66 is recommended after 9 when HOV is lifted. Accident severity is not considered because even minor accidents can sometimes cause unacceptable delays.

The rules also take the weather into consideration. For example, bad weather is considered to pose a risk to driving an automobile and thus the Metro is recommended independently of time of day.

On weekends, the rules recommend Route 66 at all times except during severe weather. The motivation for driving rather than riding the train is that weekend trips are for pleasure and having a car provides flexibility to accommodate deviation from initial plans. For example, one might suddenly decide to go to the beach.

Each of the expert system's 45 rules is encoded as a 5-tuple of discrete real numbers: two values respectively represent travel on a weekday or holiday; three values are needed for time of day represent time of day; four values represent absence of an accident or its location; four represents weather conditions. A three-bit value represented the recommended route. The results described below are averages obtained from a single 4-5-3 network topology using full layer-to-layer connectivity but no additional feedforward links.

Results

Because the aim of the project is to study error reduction rate improvements, the basic gauge of performance is the level of error vs. number of training epochs.A basic, no-frills back propagation simulator was only able to train this network to an error value of 0.7 after several thousand training epochs. In this simulation, $\eta = 0.5$ typically and $\alpha = 0.9$.

The algorithm was then modified so that the learning rate, η, would be set to a minimum value (0.01) as soon as the gradient correlation went negative. By thus reducing the learning rate, we avoid a loss of learning due to a step size that was too large for the error surface topography. Simulation results confirmed the effectiveness of this modification.

We also modified the momentum parameter, α, in a way similar to [2]. Whenever the gradient correlation became negative, the momentum parameter was set to 0, the heuristic being that a large change in the gradient vector's direction means that the direction of the next step should not be influenced by the previous step. However, instead of immediately returning the momentum to .9 for the next step, the momentum is gradually built up over time. This new implementation of the gradient correlation approach was then tested by training a network on the commuter rules data. Training led to an error of 0.04 after some 800 epochs, which allowed the network to make 100% correct route recommendations for all the patterns in the training set.

Comparison

The simulator code was also modified to implement the accelerated convergence method described by Vogl *et al* and tests were conducted on randomized 4-5-3 networks using the commuter rules data, as before. Our results show this method to perform quite well. In fact it performed nearly as well as the gradient correlation method, but it usually required a little more training to reach a given error level Their method requires about 1200 epochs to achieve an error of 0.04 whenever β was high (0.7 as in their paper). At this β level, many steps were rejected because the learning rate is not reduced quickly enough. The method reduced large error very quickly but then had trouble converging to a small error value because it waits for errors before adjusting the learning rate, at which time the error had increased. On the other hand, when β was smaller (0.3), fewer steps were rejected but more training was required to converge.

Without using gradient correlation, we were unable previously [4] to train a smaller (4-3-3) network to an error of less than 12.8 after 31,000 epochs. Using gradient correlation, we trained down to an error of 6.1 after only 300 epochs. Other tests have given similarly positive results.

Conclusions:

Dynamic learning parameter adjustment is a significant improvement over conventional back propagation. Vogl's *et al.* reactive method for accelerated convergence is confirmed to be somewhat less effective than proactive gradient correlation. Further investigation may show which method more useful with large networks.

The gradient correlation approach saves the time the other method uses to compute backward and forward propagation steps that are later rejected, but spends some time computing the correlation. As far as computational complexity is concerned, computing the gradient correlation (indeed, just the inner product is needed) is not nearly as complicated as a back propagation pass (which in the other method is sometimes rejected).

References

1. Rumelhart, D.E, J.L. McClelland *et al.*, Parallel Distributed Processing, volume 1,MIT Press, Cambridge, 1986.

2. Vogl, T., P., J.K. Mangis, A.K. Rigler, W.T. Zink and D.L. Alkos, Accelerating the Convergence of the Back-Propagation Method, Biol. Cyb. 59, 257-263.

3. McClelland, J.L. *et al,* Explorations in Parallel Distributed Processing, MIT Press, 1986.

4. D. Schreibman, unpublished ms.

Learning to Identify Letters with REM Equations

Wayne E. Simon & Jeffrey R. Carter
Martin Marietta Astronautics Group
P. O. Box 179, Denver, CO 80201
(303) 977-3449

Abstract

Learning equations derived from Recursive Error Minimization (REM) are applied to the identification of letters defined on a twenty-pixel field. Learning is from ten to ten thousand times faster than conventional back propagation and is guaranteed to converge to a global minimum.

A simple measure of the strength of an identification is developed in terms of output magnitude. Measures of validity (correctness) and strength are then presented as functions of training.

Introduction

In our previous paper [1], we derived REM learning equations for neural networks and applied them to simple problems. In this paper we present the results of applying REM learning to a more complex and more realistic problem.

Equations (1), (2), and (3) describe the propagation of node state forward and the propagation of error and derivative backward. Equation (4) gives the rule for correcting the value of connections. The numerator of Equation (4) is, except for the method of presenting the constants, equivalent to the "generalized delta rule." The denominator is the second derivative of the mean square error with respect to the specific connection and has a lower bound of 0.000001 to prevent computational difficulties.

$$Y_d = \tanh\left(\frac{1}{2}\sum_c Y_c w_{cd}\right) \tag{1}$$

$$E^*_c = \left(\frac{1-Y_c^2}{2}\right)\cdot\left[E_c + \sum_d w_{cd}\cdot E^*_d\right] \tag{2}$$

$$H^*_c = \left(\frac{1-Y_c^2}{2}\right)\cdot\left[H_c + \sum_d w_{cd}\cdot H^*_d\right] \tag{3}$$

$$\Delta w_{cd} = \frac{RM_Q\left[\frac{\beta}{P}\cdot Y_c\cdot E^*_d\right]}{RM_P\left[(Y_c\cdot H^*_d)^2\right]} \tag{4}$$

Y_d	= state of a node of the d^{th} layer
c, d	= unspecified layers in a neural network
w_{cd}	= connection from a node of layer c to a node of layer d
E_c^*	= generalized error of a node of layer c (equivalent to the delta of the "generalized delta rule")
E_c	= $D_c - Y_c$, if the node has a desired state (D_c)
	= 0, otherwise
H_c^*	= generalized derivative of a node of layer c
H_c	= 1, if the node has a desired state (D_c)
	= 0, otherwise
β	= learning rate parameter ($\approx$ 1.0)
RM_P [...]	= recursive mean of parameter in brackets with P as parametric memory

Finally, Equation (5) incorporates the technique which guarantees convergence to a global minimum. The initial desired output is defined to be the initial output of the network, and is then smoothly changed to the final desired output as learning progresses. If the rate of change of the desired output is sufficiently small, the network is always in the neighborhood of a global minimum, and will still be in the neighborhood of a global minimum when the desired output has transitioned to the final desired output. Note that Equation (5) has been simplified from that presented previously [1].

$$D^{n}_{q,l} = D^{n-1}_{q,l} \cdot \left(\frac{R-1}{R} \right) + D^{\infty}_{q,l} \cdot \left(\frac{1}{R} \right) \tag{5}$$

$D^{0}_{q,l}$ = initial desired output (= initial observed output) of the l^{th} output node to the q^{th} experience
$D^{\infty}_{q,l}$ = final desired output of the l^{th} output node to the q^{th} experience
R = parametric memory of initial desired output

Problem

We applied REM equations to the problem of identifying the twenty-six capital letters of the English alphabet. The letter representations, shown in Figure 1, are given on a four by five grid of binary pixels, for a total of twenty pixels corresponding to twenty inputs to the network. The desired output was a number from one to twenty-six corresponding to the letters A through Z. In binary notation it requires five bits to represent twenty-six, resulting in five outputs from the network.

We used a feed-forward network of forty-five nodes: twenty input nodes, twenty intermediate nodes, and five output nodes. The intermediate and output nodes had biases modeled as the connections from a "node zero" which always had an output of one.

Results

As we pointed out previously [1], the node transfer function used here has a range from -1 to +1. This choice has no effect on the basic characteristics of the network, but does make possible a very convenient method for presentation of the validity and strength of identification. If the signs of all the output nodes are correct, the identification is valid. The product of the magnitudes of the output nodes gives a measure of strength of the identification which ranges from zero to one.

The network learned to give valid identifications in about fifteen epochs and strong identifications in thirty epochs. Figure 2 shows the distribution of identification strength with thirty epochs of training. The distribution was computed with a twenty-slot histogram, so the resolution is only 0.05 and the maximum function value is twenty. Figure 2a shows that all letters of the training set were identified with strengths greater than 0.85. In order to give some indication of the effect of imperfect input, a test set was constructed by removing each pixel from the input field, one at a time. Figure 2b presents the distribution of strength of the correct responses (91.2%) and Figure 2c presents the distribution of incorrect responses (8.8%) to this test set. Figure 3 presents the same information after ninety epochs of training. Note that the identification strengths on the training set are all greater than 0.95, but the number of correct responses to the test set is somewhat lower (89.2).

It is obvious that if the training set is redefined to include the test set, correct responses should be obtained for all cases. Figure 4 shows that thirty epochs of training on this expanded set gives valid identifications with strengths greater than 0.85 for the original training set and strengths greater than 0.65 for the set with missing pixels.

Finally, we made a comparison with conventional back propagation using the "bp" program supplied by McClelland and Rumelhart [2] with the identical network configuration on the same problem. Figure 5 presents the epochs of training vs. RMS error for REM learning and conventional back propagation. The conventional back propagation is extrapolated from 7500 epochs (twenty-one hours) to 900,000 epochs. Note that REM is exponentially convergent and back propagation is power convergent. Thus the ratio of epochs required varies from 0.1 to 0.0001, depending on the allowable error.

We estimate the computational load per epoch for the REM equations to be about twice that of the back propagation equations. This is based on observed running time on an IBM AT. The back propagation program, written in C and optimized for speed, runs at ten seconds per epoch for this problem. The REM program, written in Pascal with no consideration given to speed, runs at forty seconds per epoch, with an assumed factor of two improvement with optimization.

Conclusion

Learning with REM equations has been demonstrated to be orders of magnitude faster than learning with conventional back propagation equations, both in number of epochs required and total computational time. The advantage of the REM equations increases at lower levels of acceptable error. The node transfer equation chosen for this work is shown to provide a convenient method of presenting the validity and strength of identification.

References

[1] Simon, W., and Carter, J., "Back Propagation Learning Equations from the Minimization of Recursive Error," *Proceedings of the IEEE International Conference on Systems Engineering*, August 24-26, 1989.

[2] McClelland, J., and Rumelhart, D., *Explorations in Parallel Distributed Processing*, MIT Press, 1988.

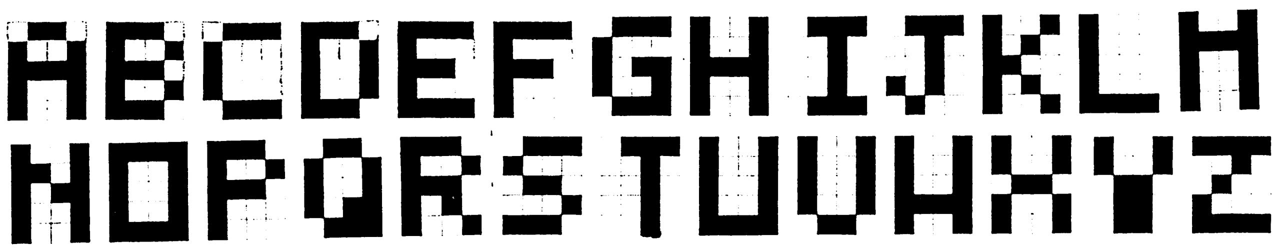
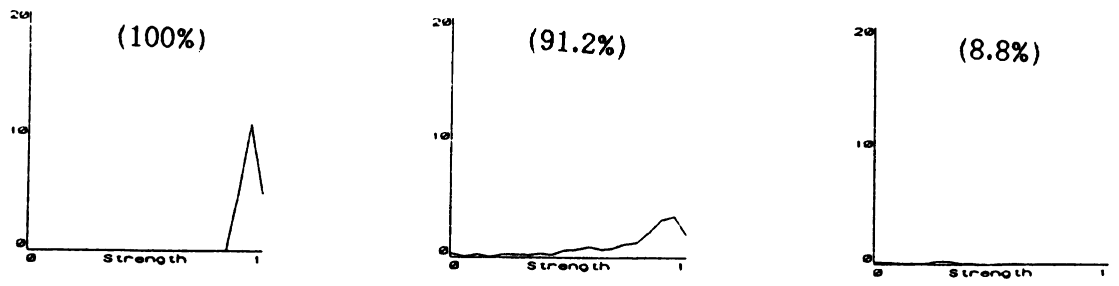
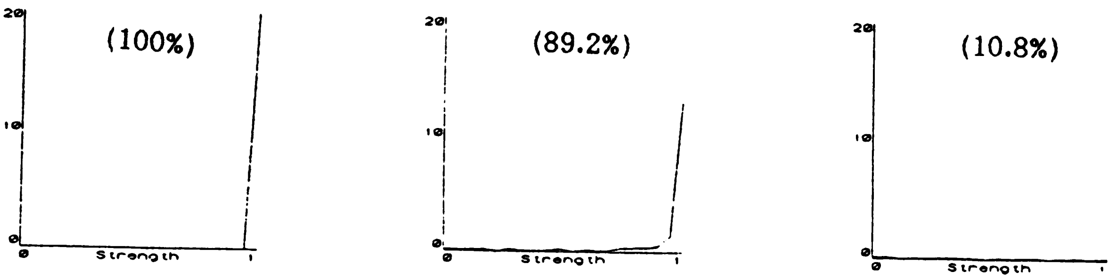


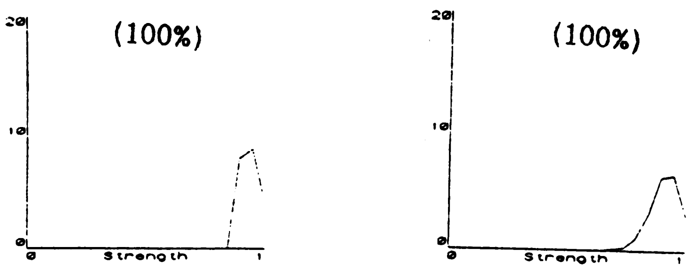
Figure 1. Capital Letters Given on a Four by Five Pixel Grid



a) Training Set--Valid b) Test Set--Valid c) Test Set--Invalid
Figure 2. Identification Strength--Thirty Epochs of Training Set



a) Training Set--Valid b) Test Set--Valid c) Test Set--Invalid
Figure 3. Identification Strength--Ninety Epochs of Training Set



a) Training Set--Valid b) Test Set--Valid
Figure 4. Identification Strength--Thirty Epochs of Training and Test Sets

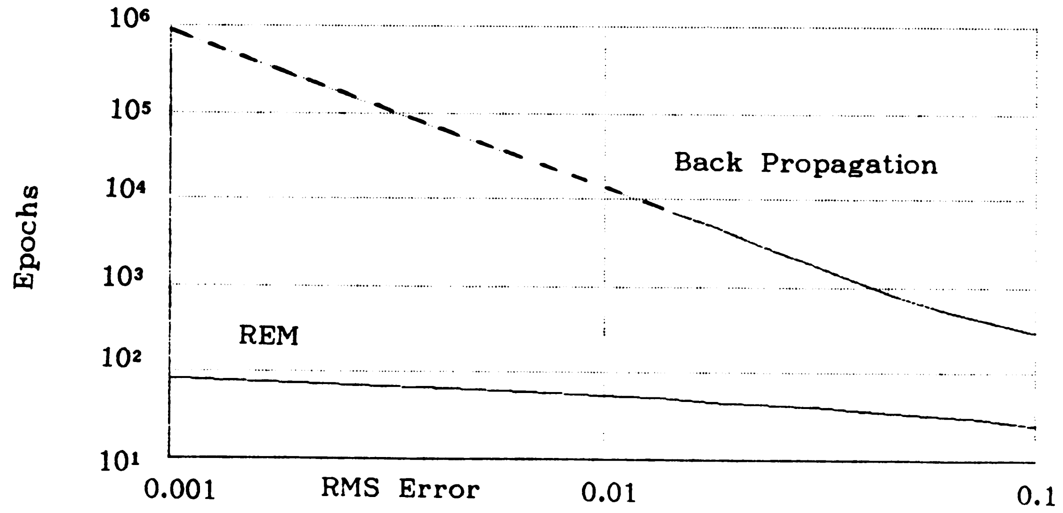


Figure 5. Required Training as a Function of Allowable RMS Error

Improved Back-Propagation Combined with LVQ

Takehisa Tanaka, Motohiko Naka, and Kunio Yoshida

Matsushita Research Institute Tokyo, Inc.
3-10-1, Higashimita, Tama-ku,
Kawasaki, 214 Japan
ttake@mrit.mei.junet

Abstract

Back-propagation(BP) is the most popular learning method for multi-layer perceptrons. Lately many improved methods of BP have been proposed, but most of them are about how the network learns. In this paper we propose a new model considering what the network learn. In this model a LVQ network learns input patterns prior to a BP network which learns the reference vectors adjusted through the pre-learning of the LVQ. We test our model with computer simulation.

1. Introduction

BP is the most popular learning method for multi-layer perceptrons and various improved methods have been proposed. Lately Takagi proposed one variation using DSC search method of non-linear optimization.[1] This method accelerates convergent speed, but it is necessary to calculate the gradient of squared error function of all learning patterns and consequently it is time consuming when there are many learning patterns. To emphasize the feature of the method, we must reduce the number of learning patterns. For this purpose we adjust reference vectors with LVQ learning method which T. Kohonen proposed[2] and use them as learning patterns of BP. Only the learning algorithm is changed in most variations of BP, but in this model learning patterns of BP are transformed, too. In other words we change what the network learns in addition to how the network learns.

2. Improved BP using DSC search

DSC search was originally proposed by Davies, Swann, and Campey as line search algorithm for an optimization subproblem.[3] Recently Takagi proposed to use the DSC search for finding the best learning rate in BP and showed the learning with DSC search converged more than three times faster than the original BP. The algorithm proposed by Takagi is:

1) Initialize weight vector $\vec{w}_0$ and set ε_1 to a small value.

2) Calculate the outputs for every input pattern and sum up them to the total squared error $E(\vec{w}_0)$. After that, calculate the gradient $\Delta\vec{w}$ of the total squared error in the same way as the original BP without changing the weight vector.

3) Calculate $\vec{w}_1$ by

$$\vec{w}_1 = \vec{w}_0+\varepsilon_1\Delta\vec{w} \tag{1}$$

Then calculate $E(\vec{w}_1)$.
If $E(\vec{w}_0) < E(\vec{w}_1)$, set ε_1 to a smaller value and repeat this step.

4) If $E(\vec{w}_i) > E(\vec{w}_{i-1})$ (i = 1,2,$\cdots$), go to 6).

5) Set ε_{i+1} to doubled ε_i and calculate $\vec{w}_{i+1}$ and $E(\vec{w}_{i+1})$. Then go to 4).

6) Calculate $\varepsilon_{i+1/2}$, $\vec{w}_{i+1/2}$, and $E(\vec{w}_{i+1/2})$ as following,

$$\varepsilon_{i+1/2} = (\varepsilon_i+\varepsilon_{i+1})\ /\ 2 \tag{2}$$

$$\vec{w}_{i+1/2} = \vec{w}_0+\varepsilon_{i+1/2}\Delta\vec{w} \tag{3}$$

7) Select minimum three values among $E(\vec{w}_{i-1})$, $E(\vec{w}_i)$, $E(\vec{w}_{i+1/2})$, and $E(\vec{w}_{i+1})$. Then set E_1, E_2, and E_3 to them respectively and set E_1, E_2, and E_3 to ε corresponding to E_1, E_2, and E_3 respectively.

8) Calculate E as

$$E = E_2 - \frac{E_2 - E_1}{2} \frac{E_3 - E_1}{E_3 - 2E_2 + E_1} \quad (4)$$

9) Calculate new weight vector as

$$\vec{w}_0 = \vec{w}_0 + E\Delta\vec{w} \quad (5)$$

10) Go to 2) until $E(\vec{w}_0)$ gets less than the pre-determined value.

3. LVQ (Learning Vector Quantization)

Learning Vector Quantization is one of nearest-neighbor methods. In this model there is a pre-determined number of processing units and each unit has a reference vector of the same dimensions as input pattern vectors. Each processing unit is associated with one of the categories of the input patterns. If $\vec{m}_c$ is the closest reference vector to an input pattern $\vec{x}$ in some appropriate metric, the category which $\vec{m}_c$ belongs to is the classification of $\vec{x}$. During learning, $\vec{m}_c$ is updated as follows,

If $\vec{x}$ belongs to the same category as the nearest unit $\vec{m}_c$,

$$\vec{m}_c(t+1) = \vec{m}_c(t) + \alpha(t)(\vec{x}(t) - \vec{m}_c(t)) \quad (6)$$

If $\vec{x}$ belongs to the different category with the nearest unit $\vec{m}_c$,

$$\vec{m}_c(t+1) = \vec{m}_c(t) - \alpha(t)(\vec{x}(t) - \vec{m}_c(t)) \quad (7)$$

where $0 < \alpha(t) < 1$ and $\alpha(t)$ is decreasing monotonously with t.

4. Combining improved BP and LVQ

Though DSC search accelerates the convergent speed of BP, there are still some problems. For example,

1) Because the initial weights and the sequence of learning patterns influence learning process, a learning result differs from another learning result using different initial states or different learning patterns.
2) It is difficult to evaluate the optimal network size.
3) Learning time increases inversely proportional to the number of learning patterns.

On the other hand there are some problems with LVQ. For example,

1) Initial values of reference vectors must be set to a good state.
2) Since the LVQ learning algorithm makes only look-up table, many reference vectors are often necessary.
3) LVQ doesn't make a continuous function.

In order to suggest the solution of problems mentioned above, we propose to use the LVQ reference vectors as the learning patterns of the improved BP. Our algorithm is:

1) Classify a large number of input patterns to some categories and then adjust initialized reference vectors with LVQ learning. In the pattern classification problem, execute LVQ learning directory.
2) After LVQ learning converges, let the multi-layer perceptron network learn the reference vectors with improved BP using DSC search.

Our proposed method has some merits. In short they are

1) Because our method suppresses a number of learning patters, the improved BP converges quickly.
2) Since we can evaluate complexity of relation between inputs and outputs by analyzing reference vectors adjusted with LVQ, the optimal network size of the perceptron can be estimated.
3) By analyzing distribution of reference vectors, it is able to scale each element of input vectors before BP. It makes the learning faster.
4) We can easily tune BP learning process by hand. For example, if squared error of input patterns belonging to a certain category is greater than the others, the category may be learned more frequently than the others.

But there are some disadvantages in our model also. That is

1) Since our method needs both LVQ learning and improved BP using DSC search, it takes larger time.
2) It is not easy to apply this model to non-classifier problem.
3) Reference vectors after LVQ learning reflects the distribution of learning patterns but they are not always the best learning patterns for the BP learning.

5. Simulation and Results

We carried out simulation to test our model.

We applied our model to 2-dimensional pattern classify problem. Each learning pattern was extracted form one of seven categories. The input patterns of each category are distributed according to the normal distribution in Table 1, but category II and IV consisted of two separated normal distribution. We made 10,000 input patterns randomly.

We used both 900 and 270, in total, reference vectors and each category had the same number of reference vectors except category II and IV that had twice as many processing units as the others. So category II and IV had 200 or 60 reference vectors. We made initial reference vectors randomly according to the distribution in Table 1. Initial distribution of reference vectors in case of 270 reference vectors are in Fig.1.

During LVQ learning, we made reference vectors learn all input patterns 600 times altogether. The distribution of reference vectors after learning in case of 270 reference vectors are in Fig.2, and the misclassified rates to all input patterns are in Table 2.

After LVQ learning, we carried out BP learning using DSC search in 2-layer perceptron. We used 10 nodes for the hidden layer and 7 nodes for the output layer. Each output node corresponded to each category and the category which the output node generating maximum output belonged to was the classification to the input pattern at that time. We used reference vectors before learning, after 100 LVQ learning, and after 600 learning for learning patterns of BP. The misclassified rates to 10,000 input patterns are in Table 3.

When 900 reference vectors were used as the BP learning patterns, there was no difference between reference vectors before training and after 600 LVQ training. But learning using reference vectors after 600 LVQ training was explicitly better when 270 reference vectors were used.

6. Conclusion

We proposed a new model which used both LVQ for prior data conversion and improved BP using DSC search for calculating optimal learning path of BP. Since our algorithm reduce the number of learning patterns for BP, it makes learning faster. In addition, as LVQ makes the structure of the learning data clear, it enables us to estimate the perceptron network size. We showed that our algorithm improved learning efficiency and did no harm to learned ability by simulations.

Acknowledgements

Many thanks to our colleagues, especially Mr. Takagi for his advice about DSC search.

References

1. H. Takagi, S. Sakaue, and H. Togawa, "Fast Learning Algorithm for Artificial Neural Network using Non-Linear Optimization Method," *1989 Spring National Convention Record IEICE*, pp. 7-305, 1989. (in Japanese)
2. T. Kohonen, G. Barna, and R. Chrisley, "Statistical Pattern Recognition with Neural Networks:Benchmarking Studies," *Proc. ICNN*, vol. 1, pp. 61-68, 1988.
3. W.H. Swann, "Report on the Development of a New Direct Search Method of Optimization," *I.C.I. Ltd., Central Instrument Laboratory Research Note*, vol. 64/3, 1964.

Category		I	II-1	II-2	III	IV-1	IV-2	V	VI	VII
X	Average	3.0	14.0	11.5	26.0	6.5	22.5	16.5	4.0	22.5
	SD	1.5	4.0	1.75	2.0	3.25	1.25	1.75	2.0	3.75
Y	Average	2.5	2.5	0.5	2.5	1.5	1.5	1.5	0.5	0.5
	SD	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25

Table 1. Averages and standard deviations(SD) of each category

Learning Times	0	100	300	600
Error rate in case of 900 reference vectors (%)	9.07	6.84	5.81	5.44
Error rate in case of 270 reference vectors (%)	12.0	6.21	5.44	5.42

Table 2. Results of the LVQ learning

Number of reference vectors	900			270		
Learning times by LVQ	0	100	600	0	100	600
Error rate after 2000 times DSC (%)	36.2	35.3	36.0	36.2	34.0	33.0
Error rate after 4000 times DSC (%)	27.2	26.1	27.0	25.4	23.4	16.2
Error rate after 14000 times DSC (%)	6.39	6.03	6.04	12.0	6.92	6.19

Table 3. Result of the improved BP using DSC search

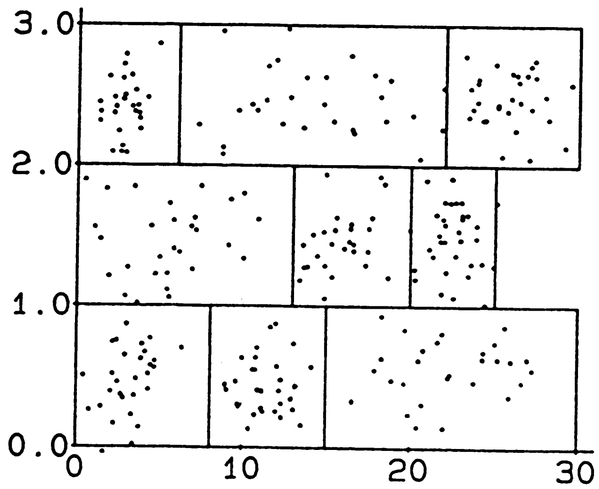


Fig.1 Reference vector distribution before LVQ

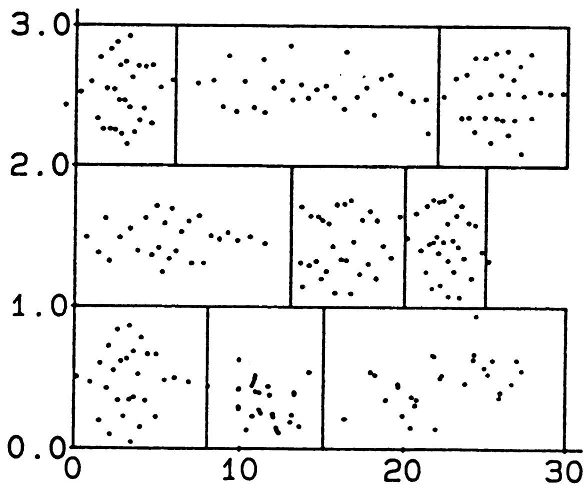


Fig.2 Reference vector distribution after LVQ

ADDING TOP-DOWN EXPECTATION INTO THE LEARNING PROCEDURE OF SELF-ORGANIZING MAPS

Lei Xu †and Erkki Oja
Department of Information Technology, Lappeenranta University of Technology
Box 20, SF-53850, Lappeenranta, Finland
†Permanent address: Dept. of Mathematics, Peking University, P.R.China

Abstract. The self-organizing topological map is provided with a top-down expectation mechanism. The analysis and computer simulations show that this generalization can not only retain the charateristics of the original topological map, but also has new characteristics of converging faster, working for nonstationary input data, having adjustable discriminative ability and resistance to abnormal noise.

1. Introduction. As one of the most notable recent developments of competitive learning, Kohonen's self-organizing topological map has been widely investigated in the aspects of applications, theoretical analysis, extensions and variations [1-3]. Two recent variations were given in [4,5]. Paper [4] made a modification on convergence speed and data representation by adding a "conscience" to the competition. In paper [5], the ordering of data with different variances in each dimension is improved by using accordingly weighted distances, and better and faster approximations of prominently structured density functions are obtained by introducing the minimal spanning neighborhoods.

Deviating from those approaches, it is suggested in this paper that the maps be generalized by using top-down expectation. The importance of top-down expectation in human perception has been emphasized by Grossberg [6]. Particularly, in the ART model, top-down expectation is used for self- stabilizing adaptive pattern recognition in real-time nonstationary input environments. Top-down expectation is here introduced into topological maps by checking whether the present input is compatible with the previously learned information of its best-matching unit, and if not, by finding another candidate for the best matching unit. This results in a modified self-organizing procedure. Both analysis and computer simulations have shown that the procedure can not only retain the properties of the original topological maps but also has some advantages such as converging rapidly, working for nonstationary input data, having adjustable discriminative ability and resistance to abnormal noise.

2. A Modified Self-Organization Procedure. For a 2D unit lattice with each unit i having a weight vector m_i, the original self-organization procedure consists of the following two steps [1]:
Step 1: For each input x, find the best-matching unit c with its weight vector m_c being the best one to match x among all the m_i under a matching criterion, e.g., Euclidean distance,

$$\|x - m_c\| = min\|x - m_i\| \forall i. \qquad (1a)$$

Step 2: Improve the match between x and the weight vectors m_j of units within a topological neighborhood N_c of unit c,

This work was supported by Tekes Grant 4196/1988 under Finsoft project

$$m_i := \begin{cases} m_i + \alpha[x - m_i] & \text{if } i \in N_c; \\ m_i & \text{if not } i \in N_c \end{cases} \tag{1b}$$

where α and N_c are monotonically decreasing with the iteration step.

The top-down expectation process is introduced between Step 1 and Step 2. When a unit which has already adapted to previous inputs becomes the best- matching unit for the present input x, it should retain the earlier learned information as much as possible in the updating. It is especially desirable that the closer the unit is to its converged state, the more strongly it will resist x from erasing the information learned earlier. It should become only possible for it to adapt to new input patterns which are compatible or similar with the previously learned patterns. Any input which is incompatible with (i.e., significantly different from) the weight vector of the unit will be rejected. In such a case, the second best-matching unit to the input is found as the new best- matching unit and another top-down expectation process starts in the similar way.

On the other hand, if the unit is far from its converged state, it is more sensitive for learning any new input pattern although this learning may erase some information previously learned.

One problem is how to know when a unit is close to convergence. We use a scalar measure for each unit i, denoted σ_i, to describe the state of convergence. At each step, these scalars are updated according to

$$\sigma_i := \begin{cases} \beta\sigma_i + (1-\beta)\|\Delta m_i\| & \text{if } i \in N_c; \\ \sigma_i & \text{if not } i \in N_c. \end{cases} \tag{2}$$

There Δm_i is the change in the value of m_i during the last step (it is nonzero only if $i \in N_c$). In effect, σ_i is a kind of weighted mean of the norms of all the previous differences Δm_i. Since the parameter β is on (0,1), the σ_i will become small when the unit is close to convergence.

In summary, the generalized self- organizing procedure is proposed as follows:

```
repeat until converged or for a fixed number of steps
  begin
  input new $x$, place all the units $i$ in set $U$;
  compatible := false;
  repeat until compatible or $U = \emptyset$
    begin
    $c := bestmatch(x, U)$;
    (bestmatch gives the index of the best matching unit for $x$ in set $U$);
    $U := U - \{c\}$;
    if $\alpha\|x - m_c\| < \nu\sigma_c$ then compatible := true;
    end;
  if $U \neq \emptyset$ then begin
    update all $m_i$ according to (1b);
    update all $\sigma_i$ according to (2)
    end
  end.
```

Some remarks:

(1) Within the procedure, the condition

$$\alpha\|x - m_c\| < \nu\sigma_c \tag{3}$$

is used for testing whether x is compatible with unit c. There $\nu > 1.0$ is a given parameter (the "vigilance"); the smaller ν is, the more sensitive the test is, thus, only a small difference $\|x - m_c\|$ can pass the compatibility test. Similarly, the opposed effect is obtained if ν is large. Furthermore, it follows from Eq. (1) that $\Delta m_c = \alpha[x - m_c]$, and so checking whether (3) holds is in fact checking whether the increment $\|\Delta m_c\|$ of the weight vector by learning the present x is not larger than ν times the weighted mean of all the previous increments. Since σ_c becomes smaller as the unit converges, the increment of the weight vector by learning this x must be more strictly constrained to be quite small such that the erroneous erasure may be avoided.

(2) When $U = \emptyset$ happens in the loop, it means that the present x is significantly different from patterns previously learned on all the units of the map. So we have to give up this x to prevent erasing already learned events.

(3) Some ways to initialize σ_i are given in [7].

(4) Our procedure will reduce into the original self- organization procedure if we let the vigilance level be very low (i.e., ν is a very large constant). So, it can still retain the properties of the original procedure.

3. Charateristics of the Modified Procedure.

Faster Convergence: In the self-organization procedure, when a new input pattern finds its best matching unit, the unit will adapt to it even if it is significantly different from the earlier learned patterns. As a result, the prior learned knowledge may be erased too strongly and need be relearned in future steps, which results in repeated learning. In the procedure suggested here, the best- matching unit will check whether or not the input pattern may refine its already established weight vector. If not, it will be passed to a unit on which there is no prior knowledge, or the prior knowledge is not well established (i.e, which is far from convergence). Thus the repeated learning may be partially avoided, which makes the convergence speed faster.

Works In a Nonstationary Environment: The self-organization procedure was designed for stationary inputs in the first place, since the main goal is to find an optimal vector quantization for a stationary input distribution. Understandably, for nonstationary inputs, it may lead to an unstable cycle of learning and forgetting. Also, some unexpected new inputs may disturb or even destroy the already organized map unless it is frozen on purpose. In the modified procedure, nonstationarities will not lead to unstable learning-and-forgetting cycles since they will be prevented by the compatibility test of top-down expectation. When an unexpected new pattern comes to a best-matching unit of the already organized map, the unit will test whether or not the pattern can refine it, and if not, the pattern will be sent to another unit. Thus the earlier learned map is retained. The new procedure can maintain its ability to learn new unexpected inputs until all the units on a 2D lattice have converged. In this case, a new unexpected input will not be learned by any unit and also not disturb the already learned ordering, so there is no need for externally switching off.

Adjustable Discriminative Ability: It is possible to adjust the sharpness of discrimination via the parameter ν. If a unit has high attentional vigilance (small ν), then only an input having a high degree of similarity to the weight vector of its best-matching unit can satisfy Eq. (3). As a result, a unit of the organized map will only response to patterns which are very similar. On the other hand, if a unit has low attentional vigilance (large ν), then a best- matching unit can tolerate larger mismatches between the input pattern and the weight vector of the unit, i.e., the map has a rough discriminative ability. For the special case $\nu = \infty$, the procedure will reduce to the original self-organization procedure.

Resistance for Abnormal Noise: Even in a stationary input process, if some abnormal noise is contained in some of the inputs, this can affect an already organized map. In our procedure, when an input with abnormal noise comes, it will either be passed to a new unit to learn it or simply not be learned by any unit if there is no spare unit, since the compatibility test on the already learned units will reject this noise input. Therefore the learned map will not be contaminated or demolished by abnormal noise.

4. Simulation Results. Computer simulations were conducted both by the original self-organization procedure (1a,b) and the generalized procedure to show each of the above four advantages. Due to limited space, here we have to omit the experimental details and illustrations of the results. All of them are given in [7]. The results have shown that: (1). Our procedure is faster than the original self-organization and seems to be more robust for parameters α and N_c. (2). For nonstationary input, e.g., samples are alternatively from two separated classes in such a way that samples consecutively come from one class for a certain period and then from the other class for another period, a learning and erasing cycle makes the topological map difficult to organize. In contrast, our procedure only learns samples from one specific class at a time and can rapidly converge to a good organized map. (3). When some abnormal noise was introduced to an already organized map over a training period, the map was at least partly destroyed if self- organization in the original form was carried out, but could keep unchanged by the new procedure. (4). With different values of ν , the new procedure did give organized maps with different discriminative ability.

Acknowledgement: The first author expresses his thanks to J.Lampinen, P.Kultanen and P.Koikkalainen for their help on his work of C language programming and computer simulations.

REFERENCES

[1]. T. Kohonen, *Self-Organization and Associative Memory*. Springer, Berlin, 1988.
[2]. Self-Organization Section, Proc. of IJCNN'89, Washington,D.C.
[3]. Self-Organization Section, Proc. of IEEE ICNN'88, San Diego.
[4]. D.DeSieno, i.b.i.d., pp I117 - I124.
[5]. J.Kangas and T.Kohonen, pp II517 - 522 in [2].
[6]. S.Grossberg, Neural Networks, Vol.1, 1988, pp 17 - 61.
[7]. L. Xu and E.Oja, "Adding Top-Down Expectation into The Learning Procedure of Self-Organizing Maps", Technical Report, Lappeenranta Univ. Tech., Aug. 1989.

Analyses of the Hidden Units of Back-Propagation Model by Singular Value Decomposition(SVD)

Qiuzhen Xue, Yuhen Hu, and Willis J. Tompkins

Department of Electrical and Computer Engineering
University of Wisconsin-Madison
Madison, WI 53706

ABSTRACT

The singular value decomposition (SVD) method is used to analyze the output covariance matrix of the hidden units of a back-propagation model and the weight matrix consisting of the connections to the hidden layer. The results of the SVD method indicate the rank of the matrices, which helps to decide on the appropriate number of hidden units. A network with an optimal number of hidden units is efficient for both computation of learning iteration and forming of the classification regions. For excising the redundant hidden units, two steps are adopted. The first substitutes a new weight matrix formed by SVD reduction for the original weight matrix under a controllable error; The second lets the network establish a new weight matrix by relearning based on either the beginning random weights or a reduced weight matrix.

Introduction

Among the many Artificial Neural Network (ANN) models, the Back-Propagation (BP) model is often considered the most useful learning model [1]. A BP model usually consists of three or more layers: one input layer which connects input patterns, one output layer which is the output of the system, and one or more hidden layer(s) which play a key role in the learning processes. To the present, the characteristics of this hidden unit, unfortunately, remain largely unknown. When the BP model is used for pattern classification, an important task is the selection of the number of hidden units. On the one hand, hidden units should form enough generalization to classify the input patterns. On the other hand, if too many hidden units are used, the generated classification regions will be more complex than necessary, also many more training patterns and computations will be needed [2]. Some research on the topic of deciding the number of hidden units has been reported. Based on empirical data, some researchers applied heuristic rules to selecting hidden unit numbers [3], in which, different number of hidden units were tried and their performance was compared, so that a reasonable number of hidden units was decided. Some other work provides an analytic way for hidden units selection [4]-[7].

In this paper, the SVD method is used as an objective way to find the redundancy of the dimensionality formed by hidden units. In the system analyses, the SVD method is used to provide the rank of a rectangular matrix or a singular matrix [8]. By using the results of SVD, one can manipulate matrix computation easily or can do data reduction. The redundancy of the hidden units is implied either by high correlation of the weight vectors which consist of fan-in connections from the units of the lower layer to the units of the upper layer, or by the similarity of the outputs of the hidden units. The former correlation can be found from the weight matrix, each column of which corresponds to a fan-in vector, and the latter similarity can be obtained from the output covariance matrix of the hidden units. Actually, the weight matrix and the output covariance matrix are equivalent on the aspect of the redundancy, although there is a nonlinear relationship between them.

Method

In a BP model(without loss of generality, a two-layer model is used), the output as a function of the inputs is:

$$O_{ol}(x_1, \ldots, x_M) = \sum_{i=1}^{N} c_{li}\ f\,(\,net_H\,(i)\,) \qquad l = 1,2, \ldots ,P \tag{1}$$

where the $net_H\,(i)$ is the output of the ith hidden unit

$$net_H\,(j) \;=\; \sum_{i=1}^{M} w_{ij}\ x_{ip} \;+\; \theta_j \tag{2}$$

$$j = 1, \ldots .N; \qquad i = 1, \ldots .M$$

And the output of the hidden net is the sigmoid function of the net activation:

$$\vec{O_H} \;=\; f\,(\,\vec{Net_H}), \qquad where\,,\ f\,(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

For the convenience of analysis, we can express (2) in matrix form:

$$\vec{\mathbf{Net}}_{\mathbf{Hp}} \;=\; \mathbf{W}\,\vec{\mathbf{X}}_{\mathbf{p}} \tag{4}$$

where

$$\vec{\mathbf{Net}}_{\mathbf{H}} \;=\; [\ net_H\,(1),\ net_H\,(2),\ \ldots\ net_H\,(N)\ ]^T$$

$$\vec{\mathbf{X}}_{\mathbf{p}} \;=\; [\ x_1,\ x_2,\ \ldots\ x_M\ ]^T \quad \textit{input vector}$$

$$\mathbf{W} \;=\; \begin{bmatrix} w_{11} & w_{12} & . & w_{1N} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ w_{M1} & \cdot & \cdot & w_{MN} \end{bmatrix} \quad \textit{weight pattern}$$

The covariance matrix is defined as

$$\mathbf{C_{HH}} \;=\; \mathbf{E}\,[\ \vec{\mathbf{O}}_{\mathbf{H}}\,\vec{\mathbf{O}}_{\mathbf{H}}^{\mathbf{T}}\ ] \;= \mathbf{E}[\ \mathbf{f}(\mathbf{W}\,\mathbf{X_p})\ \mathbf{f}(\mathbf{W}\,\mathbf{X_p})^{\mathbf{T}}\ ] \tag{5}$$

We will first discuss the redundancy of the dependent hidden units. Assuming the output of one hidden unit is a linear combination of the rest hidden units for all input vectors, we are able to prove that the output function will not changed by excising that hidden unit and modify other weights according to that linear combination.

From the pattern recognition point of view, above statement implies that the redundant hidden units will not contribute to the forming of the classification sub-regions which, for one hidden layer networks, are some kinds of hyperplanes[9].

We can also prove that the rank of the output covariance matrix is the minimum of the rank of the weight matrix and the rank of the input pattern covariance matrix.

Let us first consider a linear case, i.e. the hidden unit output are linear, then

$$\begin{aligned} \mathbf{C_{HH}} \;&=\; \mathbf{E}[\ \mathbf{W}\,\mathbf{X_p}\,(\mathbf{W}\,\mathbf{X_p})^{\mathbf{T}}\ ] \;=\; \mathbf{E}[\ \mathbf{W}\,\mathbf{X_p}\,\mathbf{X_p}^{\mathbf{T}}\,\mathbf{W}^{\mathbf{T}}] \\ &= \mathbf{W}\,\mathbf{E}[\ \mathbf{X_p}\,\mathbf{X_p}^{\mathbf{T}}\ ]\,\mathbf{W}^{\mathbf{T}} \;= \mathbf{W}\,\mathbf{C_{XX}}\,\mathbf{W}^{\mathbf{T}} \end{aligned} \tag{6}$$

where $\mathbf{C_{XX}}$ is the covariance matrix of the input patterns. in the linear algebra [8],

$$rank\ (\mathbf{A}\ \mathbf{B}) \;\le\; \min\ [\ rank\,(\mathbf{A}),\ rank\,(\mathbf{B})\ ] \quad \text{and} \quad rank\,(\mathbf{A}) \;=\; rank\,(\mathbf{A}^T) \tag{7}$$

$$so \quad rank\,(\mathbf{C}_{HH}) \;=\; rank\,(\ \mathbf{W}\ \mathbf{C_{XX}}\ \mathbf{W}^{\mathbf{T}}) \;=\; \min\ [\ rank\,(\mathbf{W}),\ rank\,(\mathbf{C_{XX}})\ ]$$

For the general case, where the hidden unit outputs are nonlinear sigmoid functions, we also can justify the results by the theory of the random process[9].

SVD method are based on the following theorem of linear algebra: Any M by N matrix A whose number of rows M is greater than or equal to its number of columns N, can be written as the product of an M by N column-orthogonal matrix U, an N by N diagonal matrix S with positive or zero elements, and the transpose of an N by N orthogonal matrix V. It can be expressed as follows:

$$\mathbf{A} = \mathbf{U}\,\mathbf{S}\,\mathbf{V}^{\mathrm{T}} \quad \textit{where} \quad \mathbf{S} = [\sigma_1, \ldots, \sigma_N]^{diag} \tag{8}$$

After we obtain the rank of the matrix, the next question is how the matrix will be influenced by removing redundant hidden units. To answer this question we adopted two approaches. The first one is from the analysis of SVD matrix reduction; the second one is by relearning input patterns under the reduced hidden units circumstances.

For the first method, i.e. the SVD matrix reduction, by reducing the dimensions of the matrices U, V and S respectively, we can generate a new matrix $\mathbf{A}_1$ as

$$\mathbf{A}_1 = \mathbf{U}_1\,\mathbf{S}_1\,\mathbf{V}_1^{\mathrm{T}} \tag{9}$$

The second method is to excise the redundant hidden units from the original network. We tried two ways to achieve the reduction. One is to let the model relearn the input patterns from the beginning. This method works fine, at least in all our experiments. But the problem is the relearning time, especially for the large size problems. So our second way is to find which units are redundant, remove them while keeping the original weights unchanged or doing more iterative learning based on old weight values. The SVD method only tells us the rank of the weight matrix without telling us which units are redundant. We calculate the correlation coefficients of weight vectors (one vector corresponds to one column in the weight matrix) as a supplement to the SVD method. If the SVD indicates that q units are redundant, we try to find these q units which have the largest coefficients. After excising these q hidden units, we checked the network performance by letting it operate on the original training patterns. If the performance was as good, the network did not have to relearn. Otherwise, we let the network relearn based on original weights. Comparing the relearning iteration of this method and the first method which relearned from the beginning, we found that this method is very efficient in terms of computation time.

Simulation and Implementation

We developed a program which simulates back-propagation learning model on a Sun-workstation. All the SVD analyses are built into the program as options.

In order to clarify our explanation, we selected as our first experimental set as two kinds of geometric patterns; one is linearly separable by a hyperplane, the other is not linearly separable but is separable by a convex region. The dimension of both input patterns was 10. Figure 1(a) and 1(b) are their projections on two-dimensional space. The number of their classes was two. The training sets of these two cases all consisted of 50 vectors, 25 with class 1, another 25 with class 2. At the beginning, the structure of the networks was selected as 10 input units which corresponded to 10 components of the input patterns, 10 hidden units (one hidden layer), and 2 output units (1 0 represents class 1, 0 1 represents class 2). After the network converged, with total error between target patterns and actual activations less than 0.05, the SVD was applied to the weight matrix. For the linear separable patterns, the rank of the weight matrix was equal to 2, whereas for the convex patterns, the rank was 6. These results verify our statement in the previous section that the rank of the weight matrix reflects the complexity of the classification regions. To excise the redundant hidden units based on the SVD analysis, the two steps described previously were adopted. The results show that, after redundant hidden units were removed, the recognition performance was better and learning iteration was much fewer than before.

Our second experimental set is a group of random vectors with different dependency. Our SVD analyses found that the rank of the weight matrix and the output covariance matrix are related to the dependency of the input patterns.

The third experimental sets were the electrocardiogram (ECG). The automatic recognition of the ECG signals is of both practical concern and theoretical interest. Input patterns consist of five different kinds of ECG waveforms as shown in Fig. 2. The training set includes 30 different versions of these five kinds of patterns. These different versions were obtained by adding random obtained by adding random noise with a SNR of 10:1, then passing the noisy waveforms through a low-pass filter ($f_c = 70\ Hz$) to avoid aliasing. An additional 50 different versions of these five kinds of patterns were formed for the test set in the same way.

The training set is sent to the input nets. At the beginning, the BP model has 70 input nets, 10 hidden nets, and five output nets. After the first training process, the weight matrix **W** is analyzed by the SVD routine. All analyses are done automatically, yielding three outputs: reduction dimension, least-square error and new weight pattern. Then we let the system either relearn the input patterns based on new hidden nets or test both the training set and the test set based on the new weight pattern.

Conclusions

Our experiments shows that the SVD is a useful method to analyze the weight patterns. This method is an objective way for us to decide the number of hidden units, and help us to find the redundant information. Based on these results, we can excise the redundant hidden units to keep the network only generating necessary regions. Further research topics will be redundancy versus robustness, dynamic property of the SVD and so on.

References

[1] D.E. Rumelhart et al, "Learning Internal Representations by Error Propagation", in Parallel distributed Processing, MIT Press (1986).

[2] S.J. Hanson & L.Y. Pratt, "Comparing biases for minimal network construction with back-propagation", IEEE Conf. NIPS 88.

[3] R.P. Gorman & T.J.Sejnowski, " Learned classification of Sonar Targets Using a Massively Parallel Network", IEEE Trans. ASSP, Vol. 36, pp1135-1140, July 1988.

[4] S.Y. Kung & J.N. Hwang, "An Algebraic Projection Analysis for Optimal Hidden Units Size and Learning Rates in Back-Propagation Learning", IEEE 2nd Intl' Conf. on Neural Networks, San Diego, pp. I363-I370, July 1988.

[5] M. Bichsel & P. Seitz, "Minimum Class Entropy: A Maximum Information Approach to Layered Networks", Neural Networks, Vol.2, pp. 133-141, 1989.

[6] R.N. Shepard, "The Analysis of Proximities: Multi-dimensional Scaling with an Unknown Distance Function, I & II", Psychometrika, Vol.27, pp.125-140, pp. 219-246, 1962.

[7] J.K. Kruschke, "Improving Generalization in Back-Propagation Networks with Distributed Bottlenecks", IEEE 3rd Intl' Conf. on Neural Networks, Washington D.C., June 1989.

[8] V.C. Klema & A.J. Laub, "The Singular Value Decomposition: Its Computation and Some Applications", IEEE Trans Auto. Control, Vol. AC-25, pp164-176, April 1980.

[9] R.P. Lippmann, "An Introduction to Computing with Neural Nets", IEEE ASSP Magazine, pp. 4-22, April 1987.

[10] H.P. William, *Numerical Recipes in C*, Cambridge University Press, (1988).

[11] M. Minsky & S. Papert, *Perceptrons: An Introduction to computational Geometry*, extended edition, MIT Press (1988).

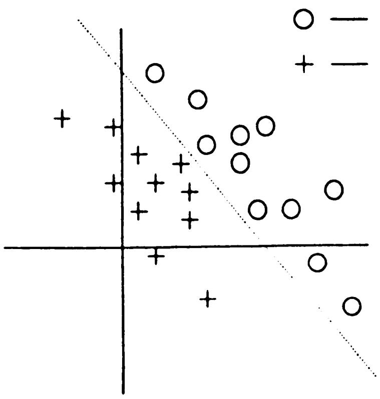


Fig. 1(a) Linear separable case

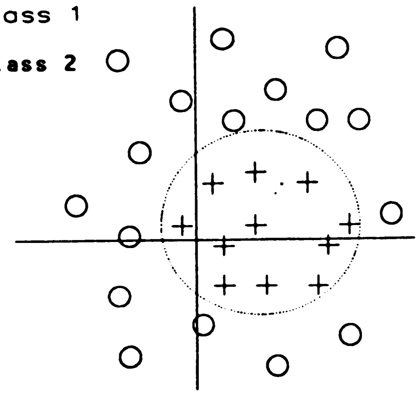


Fig. 1(b) Non-linear seperable case

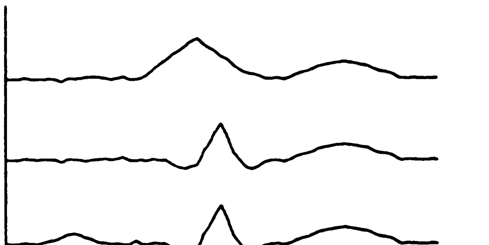


Fig. 2. Input ECG waveforms

Connections between Levels of Description of Perception

James L. McClelland
Carnegie Mellon University

Traditionally, the field of psychology has tried to characterize human information processing in terms of simple principles stated at the level of the behavior of the human cognitive system as a whole; yet we believe that the cognitive system is implemented in a large network of neuron-like computing elements. A question, then, becomes: How do we relate these different kinds of descriptions? In the past, my own research has taken the form of building computationally explicit models that make use of neuron-like elements and which can be used to simulate data obtained in psychological experiments. But there has been no effort to relate these models to the well-established general principles. In this paper, I describe results of an attempt to make such a connection.

The research begins with an observation by Massaro (1989) that the interactive activation framework developed by Rumelhart and me (McClelland and Rumelhart, 1981) did *not* accord with classical models of context effects in perception. I have analyzed the original model, discovered what lead it astray, and modified it in ways that have lead to a very satisfying result: I have found that interactive activation networks that incorporate variability directly into their processing activity do accord with classical models of context effects. I call such networks *stochastic interactive activation* networks. In what follows I briefly review the main elements of classical models of context effects. Then I describe simulation and mathematical results that demonstrate that stochastic interactive activation networks conform asymptotically to the characteristics of the classical models. This allows us to see how the principles captured by classical models might result from an underlying process of stochastic interactive activation.

Classical models. In classical models of context effects, it is assumed that decisions about the identity of a stimulus (let us say a phoneme in a word) are based on an essentially additive combination of independent cues, some of which are perceptual cues to the identity of the phoneme and some of which come from the context. For example, the decision about the identity of the last phoneme in the word *Christmas* would be based on acoustic cues arising from the sound of the final segment of the word, and from contextual cues arising from the fact that *s* is the only phoneme that fits together with the context to form a word. Both kinds of cues are graded in strength, and are combined together to produce a probabilistic outcome.

Two quantitative formulations have been proposed. In one of these, it is assumed that. associated with each alternative, there is an internal psychological continuum on which evidence for or against that alternative is added up. This evidence includes inputs from stimulus input and contextual sources, each of which has an additive effect on the total accumulated evidence. A third, also additive component is also assumed: normally distributed random noise. The process of response selection simply amounts to choosing the alternative with the largest net evidence value. This formulation arises from the theory of signal detectability (see Luce, 1963, for a discussion).

Mathematically this formulation is intractable when the number of alternatives is greater than two. For that situation, a slightly different formulation is used (Morton, 1969). The alternative formulation can also be used for the two-alternative case, and is the basis of a model called the *Fuzzy logical model of perception* (Oden and Massaro. 1978). In this formulation, each response is assigned a strength, which is the product of input and context terms. A bias term may also be included, in which case the formula for the strength of a particular alternative becomes:

$$S_i = B_i C_i I_i,$$

Where B_i represents the bias, C_i represent the influence of context, and I_i represents the influence of the stimulus input. The connection between this formulation and the signal detection formulation becomes clear if we note that an equivalent formula for S_i is $S_i = e^{b_i+c_i+i_i}$, where b_i etc. are the logs of B_i etc. Variability is not explicitly incorporated into the strengths used in this version of the model, but is introduced in the process of selection of a response. The probability of choosing alternative i out of a set of alternatives is given by Luce's (1963) choice rule:

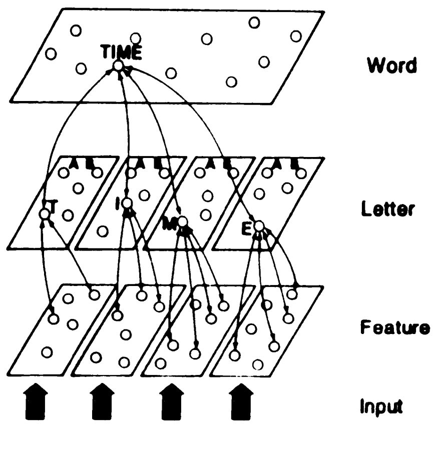
$$p(R=r_i) = \frac{S_i}{\sum_j S_j}$$

The index j in this rule runs over all of the alternatives under consideration.

It is well known that these two formulations produce essentially equivalent results for two-choice cases, and that for such cases either model generally provides a very good account of patterns of empirical response probabilities obtained in experiments that separately vary context and stimulus cues (Massaro, 1989). For multiple-choice or open-ended identification tasks, the Luce formulation provides a very succinct summary of the results of a large number of different experimental investigations. Indeed, this general account of context effects in word identification has stood since Morton (1969).

But these classical models do not describe the actual processing operations that give rise to response identification outcomes; they simply describe the asymptotic probabilities of these outcomes. Although investigators have built process description around one or the other of these formulations, it is important to note that the mathematical formulations themselves are silent about such matters as the time course of evidence accumulation, combination, and decision.

The interactive activation model. Let us now consider interactive activation models, such as the interactive activation model of visual word perception (McClelland and Rumelhart, 1981). A sketch indicating the structure of this model is shown. As the sketch brings out, the model consists of groups of processing units at several levels: Visual feature, letter and word. There are bi-directional excitatory connections between mutually consistent units at different levels (some of the relevant ones for the word TIME are shown), and bi-directional inhibitory connections (not explicitly drawn on the diagram) between mutually inconsistent alternatives at each level. So units for alternative letters in the initial letter position have mutual inhibitory connections.



Processing in the interactive activation framework takes place through a synchronous settling process. Each unit computes its net input from other units, which is simply a weighted sum of the output of each unit that projects to it times the strength of the connection between the units. (The outputs of units are equal to their activations or to 0, whichever is largest). Once all the net inputs are computed activations are updated according to the following rule:

$$\begin{array}{ll} \text{If } net_i > 0, & \Delta a_i = (M - a_i)net_i - D(a_i - r); \\ \text{Otherwise,} & \Delta a_i = (a_i - m)net_i - D(a_i - r). \end{array}$$

Here M is the maximum activation, m is the minimum activation, r is the resting activation level, and D is a decay rate constant. These equations are derived from similar equations used by Grossberg (1978).

Since the settling process was completely deterministic in the original formulation, a probabilistic method was required for translating these activations into response probabilities. For this purpose, the Luce choice rule was applied to the exponentials of the activations of the units, so that $p(R=r_i) = \frac{e^{a_i}}{\sum_j e^{a_j}}$.

A problem. This formulation was highly successful in allowing the model to account for a wide range of experimental findings on context effects in perception of briefly flashed letter strings. However, Massaro (1989) pointed out that the model as described above is incompatible with the quantitative characteristics of context effects found in experiments that systematically vary the strengths of contextual and stimulus cues; the same ones that are so closely fit by the classical models.

Massaro's claim rests on simulation results, and on the following intuitively appealing argument: Interactive activation models, he claims, cannot capture the classical context effects because of their inherent use of a bi-directional flow of information. Independence of the contributions of context and stimulus information is not maintained because of this bi-directionality. Stimulus input feeds in from the bottom up, propagates further upward, and then propagates down again, and can (when the context is conducive) become amplified in the process. Thus context and stimulus can produce synergistic, rather than strictly additive effects, contrary to the evidence.

My analysis reveals that this argument is incorrect. In fact, it turns out -- perhaps surprisingly -- that interactive activation can produce the classical effect of context on perception. There is a problem with the formulation described above, but it is not the fact that it relies on bi-directional flow of information.

Solution. To correct the model, we need simply to incorporate random variability into the actual processing activity. I have done this in three ways. Two of these involve minimal changes to the model: The first assumes that variability arises in the *input* to the network, in the form of random perturbations of inputs to units. The second assumes that each processing unit has its own *intrinsic* variability. This is modeled by assuming that on each update, the net input is perturbed by a small amount of normally distributed random noise.

In both of these cases, we can dispense with the use of the Luce rule, and simply make response choices in the following way: We present an input at time t=0, allow the network to settle, and then, at some arbitrary but long time after the beginning of processing, we simply choose the most active alternative from the candidate set. So, if the task is to identify the final letter in the word, we simply choose the letter corresponding to the unit with the largest activation. I have demonstrated that both assumptions about the source of variability lead to very close mimicry of the classical models in a number of different example cases (See McClelland, in press).

Why does this approach work where the deterministic version of the model does not? Basically, in the original model, Luce's choice rule was applied to variables which involve complex non-linear combinations of stimulus and contextual influences. Bi-directional information flow contributes to this, but the problem exists even without it. Why then are the distortions of the results eliminated in the stochastic formulation? In essence, the reason is that the non-linearities apply to the noise along with the signal. They do not, for example, remove effects of noise that make an input that is really a token of one alternative seem more like a token of another; they simply implement the same incorrect decision among these alternatives that would be made if the evidence from input and context were combined linearly.

For a highly simplified interactive activation network, I was able to demonstrate mathematically that it would produce the classical pattern of results when processing noisy inputs. However, I was not able to demonstrate this for the general case or for the case of intrinsic noise inside the network. However, a third variant can be shown to implement the classical context effects in a very general class of architectures.

Context effects in Boltzmann machines. In the third variant, the interactive activation model becomes a Boltzmann machine (Hinton and Sejnowski, 1983). The architecture and connections among units are as before, though resting activation levels are replaced with bias terms. The analysis applies to a wide range of architectures providing that they have the following characteristics: a) The set of units in the network can be partitioned into three sets, consisting of those representing the alternatives, those representing the (bottom-up) input to the units, and those representing the context. b) There are no direct connections between the units representing the input and the units representing the context. c) All connections are symmetric. Note that the interactive activation model of word perception has these characteristics (Figure 1). Suppose, for example, the task is to identify the letter present in a particular letter position. In this case, the alternatives are the units representing the letter in the position in question; the input is the set of feature units in the same position; and the context consists of all of the other units in the net. There are no connections between any of the input units and any of the context units; their influences converge on the units that represent the alternatives.

Processing in the Boltzmann machine is an asynchronous, stochastic settling process. Units are chosen sequentially for updating; the net input to each unit is computed as in the interactive activation model (with the unit's bias contributing to the net input), and the activation of the unit is set to 1 or 0 using the logistic function:

$$p(a_i=1) = \frac{1}{1+e^{-net_i/T}}$$

Where T is the temperature parameter. Suppose that we present an input at time t, and then continue updating until thermal equilibrium is reached at some particular temperature T. At equilibrium, we then inspect the units representing the alternatives and if one and only one of them is active we choose that alternative as our response. The probability of choosing a particular response i is then simply the probability that the network is in one of the many possible states of the entire net, in which alternative i is active and no other alternative is active, divided by the sum of the probabilities that it is in any one of the corresponding states associated with any of the other alternatives. The probability of being in one of the many possible states in which alternative i is active and no other alternative is active plays the role of the response strength S_i in the Luce choice formula.

At this point, we can use the fact that the probabilities associated with various states of the Boltzmann machine are related to their relative goodnesses to determine these probabilities directly. Because of the absence of connections from context units to input units, we can partition these goodnesses into three parts. One associated with the unit's bias; one associated with state of the input to the unit; and one associated with the context. These components of the goodness are in fact independent, in that the goodness of one component is not affected by the goodness of the other. Because of this, I show in McClelland (in press) that it is possible to write S_i as

$$S_i = B_i C_i I_i$$

where $B_i=e^{bias_i}$, C_i is the sum of the exponentials of the partial goodnesses of the possible states of the context, and I_i is the sum of the exponentials of the partial goodnesses of the possible states of the input. Thus the Boltzmann machine asymptotically exhibits the independence of contextual and input assumed in the Classical account.

Discussion. The intuition that interactive activation inherently distorts the classical picture of the role of context in perception has turned out not to hold up under scrutiny. I take the Boltzmann version of the interactive activation model to be a full-fledged interactive activation model, in that it has the same architecture and preserves the essential bi-directionality of influences among units. Yet it is clear that in this case the essential independence of contextual and input factors is preserved. The simulations show that this is also true, at least in some cases, when the interactive activation model is kept in its original form and variability is assumed to be present either in the input or in the processing itself.

What emerges is the observation that the classical picture of the way in which context influences perception is in fact a rather general consequence of connectionist architectures in which context and input maintain a structural independence. Just how general the consequence is remains to be established through further analyses. In any case, the analysis to date suggests that it may be possible to understand general principles stated at or near the level of the behavior of the cognitive system as a whole in terms of characteristics of the underlying processing mechanisms.

References

Grossberg, S. (1978). A theory of visual coding, memory, and development. In E. L. J. Leeuwenberg, & H. F. J. M. Buffart (Eds.), *Formal theories of visual perception.* New York: John Wiley and Sons.

Hinton, G. E., & Sejnowski, T. J. (1983). Analyzing cooperative computation. *Proceedings of the Fifth Annual Conference of the Cognitive Science Society.*

Luce, R. D. (1963). Detection and recognition. In R. D. Luce, R. R. Bush & E. Galanter (Eds.), *Handbook of mathematical psychology: Vol I.* New York: Wiley

Massaro, D. W. (1989). Testing between the TRACE model and the fuzzy logical model of speech perception. *Cognitive Psychology.*

McClelland, J. L. (in press). Stochastic interactive activation and the effects of context on perception. *Cognitive Psychology.*

McClelland, J. L., & Rumelhart, D. E. (1981). An interactive activation model of context effects in letter perception, Part I: An account of basic findings. *Psychological Review, 88,* 375-407.

Morton, J. (1969). Interaction of information in word recognition. *Psychological Review, 76,* 165-178.

Oden, G. C., & Massaro, D. W. (1978). Integration of featural information in speech perception. *Psychological Review, 85,* 172-191.

Author Index

Author Index
(all references are to the first page of the relevant paper)

Author Index
(all references are to the first page of the relevant paper)

Author Index
(all references are to the first page of the relevant paper)

Author Index
(all references are to the first page of the relevant paper)

Author Index
(all references are to the first page of the relevant paper)

Title Index

Title Index

(all references are to the first page of the relevant paper; titles may be truncated)

Title Index

(all references are to the first page of the relevant paper; titles may be truncated)

Title Index

(all references are to the first page of the relevant paper; titles may be truncated)

Title Index
(all references are to the first page of the relevant paper; titles may be truncated)

Title Index
(all references are to the first page of the relevant paper; titles may be truncated)

Title Index
(all references are to the first page of the relevant paper; titles may be truncated)

Title Index

(all references are to the first page of the relevant paper; titles may be truncated)

Subject Index

Subject Index

(all references are to the first page of the relevant paper)

Subject Index

(all references are to the first page of the relevant paper)

Subject Index
(all references are to the first page of the relevant paper)

Subject Index
(all references are to the first page of the relevant paper)

Subject Index
(all references are to the first page of the relevant paper)

ISBN 0-8058-0775-6 (Vol. 1)
ISBN 0-8058-0776-4 (Vol. 2)
ISBN 0-8058-0754-3 (Set)