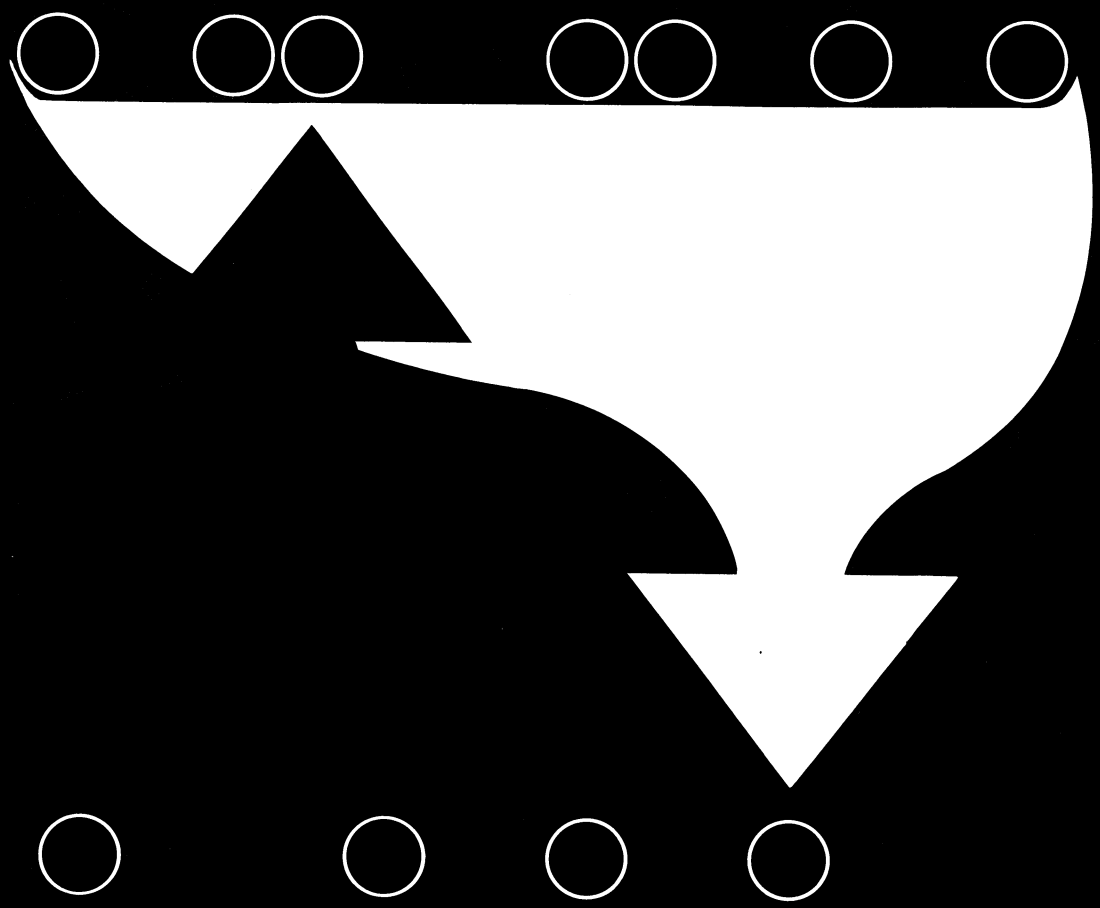
VOLUME
2

Applications Track of the Proceedings of the International Joint Conference on Neural Networks

January, 1990

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 II
Applications Track

---

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


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

General Conference Chair:
Leon Cooper, *Brown University*

Technical Program Co-Chairs:
James Anderson, *Brown University*
Andras Pellionisz, *New York University Medical School*

Finance Chair and Proceedings Coordinator:
Maureen Caudill, *Adaptics*

Exhibits Chair:
Jennifer Humphrey, *Science Applications International Corporation*

Volunteer Coordinator:
Karen Haines, *Carnegie-Mellon University*

Local Arrangements Committee:
Chair:
Harold Szu, *Naval Research Laboratory*
Committee:
Y. C. Chien, *National Science Foundation*
Lee Giles, *NEC Labs*
Richard Nakamura, *National Institute of Health*
Robert Pap, *Accurate Automation Corporation*
Paul Werbos, *National Science Foundation*
Barbara Yoon, *DARPA*

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)

Officers:

Bernard Widrow *President*
Gail Carpenter *Vice-President*
David Rumelhart *Secretary, Executive Board*

Governing Board

Shun-ichi Amari
James Anderson
Gail Carpenter
Leon Cooper
Walter Freeman
Kunihiko Fukushima
Lee Giles
Stephen Grossberg
Morris Hirsch
Teuvo Kohonen
Bart Kosko
Christoph von der Malsberg
Carver Mead
Demetri Psaltis
David Rumelhart
Terrence Sejnowski
George Sperling
Harold Szu
Bernard Widrow

IEEE Technical Activities Board, Neural Networks Committee

Robert J. Marks, II, *University of Washington,* Chair

IEEE Acoustics, Speech and Signal Processing Society
Dolores M. Etter *University of New Mexico,* President
B. H. Juang *AT&T Bell Laboratories,* Representative
Richard P. Lippman *MIT Lincoln Labs,*Representative

IEEE Circuits and Systems Society
Anthony Michel *Notre Dame University,* President
Robert J. Marks, II *University of Washington,* Representative
Robert Newcomb *University of Maryland,* Representative

IEEE Communications Society
John C. McDonald *Contel Corporation,* President
Kesh Bakhru *Cubic Corporation,* Representative

IEEE Control Systems Society
Jane K. Cullum *IBM,* President
William S. Levine *University of Maryland,* Representative
Herbert Rauch *Lockheed Corporation,* Representative

IEEE Engineering in Medicine and Biology Society
Willis Thompkins *Univeristy of Wisconsin*
Russell C. Eberhart *Johns Hopkins Applied Physics Laboratory,* Representative
Evangelia Tzanakou *Rutgers University,* Representative

IEEE Industrial Electronics Society
Fernando Aldana *University of Madrid,* President
Troy Nagle *North Carolina State University,* Representative
Yoichi Okabe *University of Tokyo,* Representative

IEEE Information Theory Society
Ian F. Blake *University of Waterloo,* President
Yasser S. Abu-Mustafa *California Institute of Technology,* Representative
Edward C. Posner *California Institute of Technology,* Representative

IEEE Lasers and Electro-Optics Society
Melvin Cohen *AT&T Bell Laboratories,* President
Joseph W. Goodman *Stanford University,* Representative
Kristina Johnson *University of Colorado,* Representative

IEEE Robotics and Automation Society
Arthur C. Sanderson *Rensselaer Polytechnic Institute,* President
Wesley Snyder *North Carolina State University,* Representative

IEEE Systems, Man, and Cybernetics Society
Arye R. Ephrath *Bell Communications Research Lab,* President
Don Bouldin *University of Tennessee,* Representative
Nicholas DeClaris *University of Maryland,* Representative

VOLUME I

NEURAL AND COGNITIVE SCIENCES

Table of Contents, Volume I

Table of Contents, Volume I

Table of Contents, Volume I

VOLUME II

PLENARY LECTURE BY BERNARD WIDROW

SPECIAL LECTURES ON SELF-ORGANIZING NEURAL ARCHITECTURES

APPLICATION SYSTEMS AND NETWORK IMPLEMENTATIONS

Table of Contents, Volume II

Table of Contents, Volume II

Table of Contents, Volume II

Table of Contents, Volume II

EXPERT SYSTEMS AND OTHER REAL-WORLD APPLICATIONS

Table of Contents, Volume II

Table of Contents, Volume II

Plenary Lecture by Bernard Widrow

Development of Neural Network Interfaces for Direct Control of Neuroprostheses

Eric A. Wan, Gregory T. A. Kovacs, Joseph M. Rosen, and Bernard Widrow
Stanford University Departments of Electrical Engineering and Surgery (Division of Plastic and Reconstructive Surgery), and the Palo Alto Veterans Administration Medical Center, Stanford, CA 94305-4055

Abstract

This article describes technologies and strategies proposed for the development of prosthetic devices which would be directly interfaced to the nervous system. A chronic neural interface device is under development which should permit the establishment of permanent bidirectional communication with peripheral nerves in a human limb. An Artificial Neural Network would be used to interpret the neural signals and drive a prosthetic limb in the case of an amputation. For nerve repair in an intact limb, the neural network would be used to reroute misdirected signals into appropriate neural channels. Design considerations for the neural interfaces and supporting circuitry are discussed along with information processing strategies. It is intended that a fully integrated prosthetic device should be capable of adapting to the individual needs of the patient to provide a natural user interface.

1 Introduction

This article describes an ongoing project which is concerned with the development of neural interfaces to the human nervous system. Silicon based interfaces, perforated by arrays of via holes, are implanted between re-apposed ends of deliberately severed peripheral nerve fascicles. Regenerating axons grow through the holes and become physically isolated and spatially fixed with respect to the microelectrodes adjacent to each via hole. This should allow for permanent access to neural signals at or near the level of individual axons. An artificial neural network can then be used to interpret and process the information contained in the neural signals. To our knowledge, this is the first attempt to directly link biological neural networks to artificial neural networks. The artificial neural network, controlled by the central nervous system (CNS), would form a natural extension to the peripheral nervous system.

This article begins with a description of the silicon based neural interface itself, which is being developed by two of the authors (Kovacs & Rosen). The emphasis of this paper, however, is on the systems level considerations in designing a complete prosthesis and the role of neural networks in adapting the prosthesis to meet the individual needs of a patient. Two applications will be discussed. The first, is concerned with the use of interfaces to redirect neural activity in an intact arm so as to aid in the recovery of a severe nerve injury. The second deals with full limb prosthesis in the case of an amputation. A more comprehensive paper which addresses such issues as fabrication techniques, power dissipation, and biological feasibility, can be found in [1].

2 The Stanford/VA Neural Interface

This group has been working on a direct neural interface for peripheral nerves for several years under Veterans Administration funding [1]. The goal of this work is to develop, largely by modification of existing commercial technologies, a microelectronic neural interface which will permit direct, chronic connection of electronic circuitry to the human nervous system.

[1]Veteran's Administration Rehabilitation Research and Development (RR&D) Merit Review Grant B003 Hentz/Rosen "Towards Better Methods of Nerve Repair and Evaluation"

In order to achieve this goal, a microelectronic neural interface capable of repeatably sensing and stimulating action potentials in small groups or individual axons of peripheral nerves is under development. While a number of investigators have experimented with implantable nerve-regeneration type recording devices using semiconductor materials [2, 3, 4, 5, 6, 7, 8], this group would be the first to attempt chronic access at resolutions approaching individual axons, in addition to the use of active electronics in the implant itself.

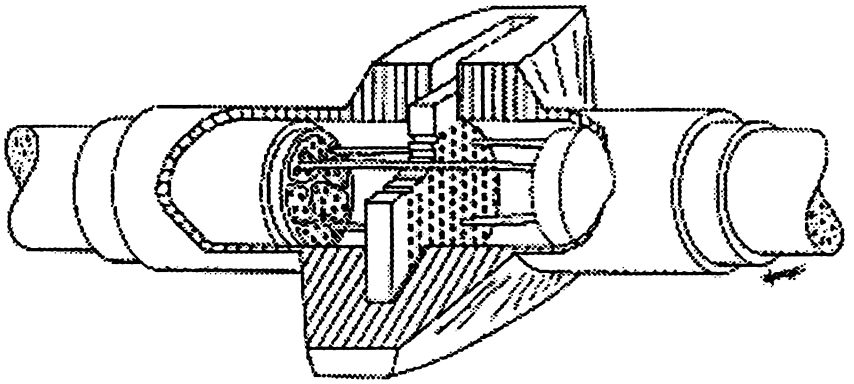


Figure 1: Drawing showing re-apposed ends of a peripheral nerve held against a microelectronic neural interface in a surgical coupler. Regenerated axons through the via holes in the silicon (not to scale) are shown in the cut-away view. (Reprinted with permission from [13].)

The current device, consisting of a silicon chip perforated by an array of via holes, will be held between the re-apposed ends of a deliberately severed peripheral nerve fascicle in a limb stump utilizing a surgical coupler (as shown in Figure 1). It has been shown experimentally [9] that for 8 to 12 μm via holes, individual regenerating axons will grow through the holes and become physically isolated and spatially fixed with respect to microelectrodes adjacent to each via holes. This arrangement will thus form a stable interface between the microelectronic circuitry on the neural interface device and the axons.

The present approach is to provide an individual neural interface for each fascicle of a major nerve. The design of monofascicular interfaces requires the use of approximately a 1 mm^2 surface area for the microelectrode array corresponding to each fascicle. The microelectrodes will be arranged in a two-dimensional grid at densities approaching those of the axons in peripheral nerves (1-2,000 axons per mm^2) to maximize access to the information present. The intimate contact of the microelectrodes with the axons should provide for good signal selectivity between microelectrodes.

Details on the development of specialized microelectrodes, surgical couplers, and fabrication processes, along with other technical and biological considerations can be found in earlier papers (see Kovacs et al [11, 12, 13, 14, 15]). It should be noted, that care was taken to utilize only processing techniques which are compatible with commercial CMOS fabrication so that the final arrays can be produced in a timely and inexpensive manner.

2.1 Preliminary *In-Vivo* Studies

A section of a blank neural interface (without microelectrodes or active microelectronic circuits) fabricated using a plasma etching technique [16] is shown in Figure 2. The blank neural interfaces were mounted in polycarbonate or resorbable GTMC (Glycolide Trimethylene Carbonate) surgical couplers and interposed between the surgically severed ends of rat and monkey nerves.

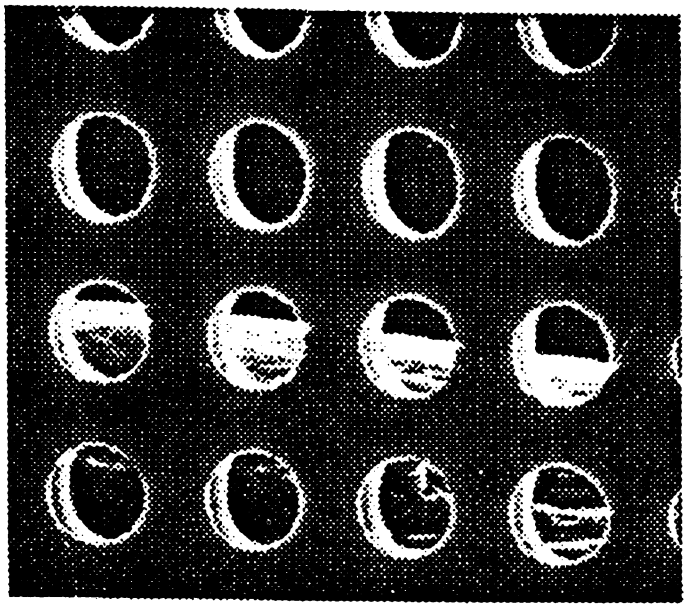


Figure 2: SEM (scanning electron micrograph) view of a section of a plasma etched blank neural interface (no microelectrodes). These via holes are approximately 12 μm in diameter.

Figure 3: SEM view of monkey axons which have regenerated through via holes in a preliminary version (without microelectrodes) of the neural interface. (magnification = 1060X) Reprinted with permission from [12].

Using functional, electrophysiological and histological techniques, it was demonstrated that viable axons had regenerated through the via holes (see Figure 3).

2.2 Current Status of Neural Interface Research

Further work on the neural interface has been carried out, with the goal of incorporating the elements required to form a functional interface [11]. Passive neural interfaces (with microelectrodes but without active microelectronic circuitry) have been fabricated and implanted in the peroneal nerves of Sprague-Dawley rats [17]. Preliminary studies indicate that both recording and stimulating with the neural interface is possible [11]. Current work is focused on determining the degree of selectivity for each microelectrode site. As well, active neural interface prototypes have been fabricated which incorporate microelectronic circuitry to permit time-multiplexing of the neural signals in order to reduce the number of connections to and from the neural interface [13].

3 Nerve Repair

One of the originally envisioned applications for the neural interface deals with nerve repair in an injured arm. In a severe arm injury, it is often possible to surgically reattach main nerve bundles at the fascicle level. However, as individual axons regenerate a sort of *scrambling* occurs when the axons grow back to the "wrong" locations. The result is a severe functional limitation in an otherwise intact hand [21]. While therapy may result in increased usage, the process is time consuming and never complete. The use of neural interfaces, however, should allow one to electronically intercept misdirected efferent (motor) and afferent (sensory) neural impulses and reroute them into appropriate channels, thus "descrambling" the signals and providing increased recovery after a severe nerve injury. In order to block the original signals, two neural interfaces for each fascicle will be required (see Figure 4). Since fascicles contain both afferent and efferent signals each interface must be capable of either recording "incoming" signals or initiating "outgoing" signals.

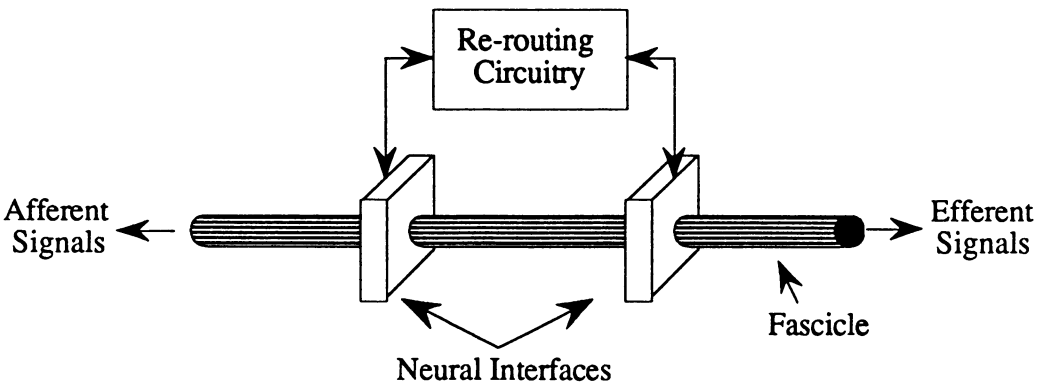


Figure 4: Diagram illustrating two neural interfaces being used for nerve repair. Two interfaces are used to reroute misdirected neural signals. The original signals are blocked between the two interfaces.

In order to learn the proper mapping from *input* microelectrode to *output* microelectrode, the use of a "neural network" is proposed. The network must be bidirectional to accommodate both efferent and afferent signals. Furthermore, both the input and output of the network must correspond to

the firing rates of axons. The difficulty with this problem is that there is no way to gain access to a *desired* response for the output of the network as would be required to train a traditional neural network. There is only a desired response for the *hand* as a whole, not for individual axon signals. In order to overcome these problems, a new algorithm, based on a variety of neural network techniques has been developed which appears to solve the mapping problem for a one-to-one descrambling.

The problem of rerouting axons is analogous to the classical linear assignment problem. We wish to assign N input axons to N output axons (N people to N tasks). The assignment can be made based on a *cost matrix*, $\mathbf{C}$, whose coefficients, $c_{i,j}$, give the cost, or gain in performance, of assigning input axon i to the output j. As will be shown, $\mathbf{C}$ corresponds to *learned* information, with the coefficients being somewhat analogous to the synaptic weights in a neural network. To formulate the assignment it is also necessary to define an *assignment matrix*, $\mathbf{T}$. The coefficient $t_{i,j} = 1$ if input axon i is actually assigned to output j, else $t_{i,j} = 0$. $\mathbf{T}$ is a sparse matrix since there is a single 1 in each row and each column. An optimal assignment is made by minimizing the cost function

$$J(\mathbf{C},\mathbf{T}) = \sum_{i=1}^{N}\sum_{j=1}^{N} c_{i,j}t_{i,j} \tag{1}$$

subject to the constraints

$$\sum_{i=1}^{N} t_{i,j} = 1 \tag{2}$$

$$\sum_{j=1}^{N} t_{i,j} = 1 \tag{3}$$

$$t_{i,j} \in \{0,1\} \forall i,j \tag{4}$$

There are many classical linear programming algorithms to solve this problem. However, $\mathbf{C}$ may also be used directly to determine the weights for a Hopfield network [18] which may then be used as an efficient method to find a "good" solution. What makes the assignment problem difficult is that we are never given the cost matrix. Initially there is no knowledge of how individual axons are to be assigned. This is similar to a case of the *traveling salesman* problem in which the distances between the cities, the $\mathbf{C}$ matrix, is initially unknown. Only the **total** path length after a trial journey to the cities is known. From this, the salesman is expected to learn the best path. Learning is reflected in the proper formulation of a cost matrix.

First define the following parameters: Let $\overline{E}$ be the average error in the hand's performance over the training period. Let E_k be the instantaneous error for a given volitional command. Then $\varepsilon_k = E_k - \overline{E}$ is a semi-quantitative measure of how well the hand performs for a given command relative to past performance. Basically, ε_k, can be thought of as a subjective measure of how well the hand is currently performing. Also define x_i as the average impulse frequency or firing rate along input axon i during the course of the current command.

The rule for adapting the elements of the cost matrix can now be defined as follows:

$$(c_{i,j})_{k+1} = (c_{i,j})_k + (\Delta c_{i,j})_k \tag{5}$$

$$(\Delta c_{i,j})_k = \mu \varepsilon_k x_i t_{i,j} \tag{6}$$

The form of this learning algorithm is similar in nature to the LMS algorithm which is used in almost all neural networks. Individual cost coefficients are adapted in proportion to the strength

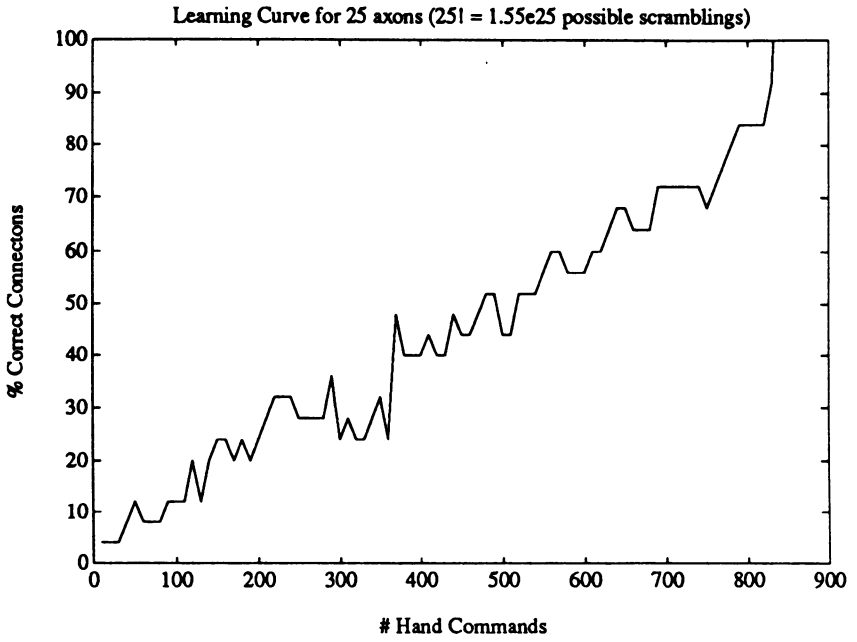


Figure 5: Learning curve for axon descrambling simulation.

of the input; the greater the input the greater its contribution to the total output error. Unlike LMS, which adapts each output neuron with respect to an individual error, all coefficient are changed in proportion to the total error. This is necessary since the desired output for each axon is unavailable. In summary, we adapt the individual cost coefficients associated with the current assignment in proportion to the total output error and the input strength at each axon. In this way, the cost matrix is adapted to reflect the overall performance of the current assignment. The complete sequence of training would proceed as follows:

1. Start with a random cost matrix.
2. Use a Hopfield Net to form an initial assignment matrix.
3. Based on the assignment matrix electronically reroute the axon signals.
4. Have the patient give his hand a command (i.e. make a fist).
5. Based on the error adapt the coefficients of the cost matrix.
6. Based on the new cost matrix use the Hopfield Net to find the new assignment matrix.
7. Go to step 3.

This procedure would continue until a suitable level of performance is achieved. This algorithm is intuitively motivated. A proof of convergence is unknown and possibly intractable considering the qualtitative nature of the error used for adaptation. While it will be years before the algorithm can be fully tested on human patients, a learning curve for a computer simulation with 25 axons is shown in Figure 5. The simulation assumed a worst case situation in which all axon signals were taken to be uncorrelated. For 25 axons there are approximately 1.551×10^{25} possible reroutings. A complete descrambling was achieved in under 900 attempts.

4 Limb Prosthesis

In an amputation, rerouting neural activity as explained above will be of little use to the patient. In this case, the interfaces can be used to establish direct communication to a limb prosthesis (i.e. a mechanical hand) [2].

Interfaces which could presently be applied chronically in a limb prosthesis application can not provide access to anything but gross averages of neural activity. Current techniques utilize mechanical command signals from unaffected tissues (e.g. shoulder movement) or electromyographic (EMG) signals from isometric contraction of muscles to control prosthesis movements. Problems commonly cited with respect to myoelectric prostheses include lack of reliability of the EMG electrodes (e.g. susceptibility to faulty operation in the presence of perspiration), the need to concentrate constantly on the muscles used to maintain a grip, and the lack of any shear (slippage) force feedback [19, 20]. As well, inconsistent placement of the electrodes can make the requisite signal processing extremely difficult [20]. In fact, old-fashioned, purely mechanical "claw" devices, which provide rudimentary proprioceptive feedback via their shoulder harnesses (Bowden cables), are preferred by many patients over more modern myoelectric prostheses [19]. The problems with these systems are all a result of limitations in the available interfaces between the patient and the prosthesis.

An ideal interface should allow for use of the limb via both the *normal* efferent (motor) and afferent (sensory) neural channels. Furthermore, since the ensemble behavior of the axons in peripheral nerves is what allows, for example, the fine motor control of the hand, it is clear that a successful interface must provide simultaneous access to the information carried in small groups or ultimately individual axons. All these requirements should be met with the current neural interface under development.

4.1 System Overview

The nature of the signals utilized by the peripheral nervous system to control various hand motions are both complicated and case specific. Thus, in order to utilize the thousands of signals available from the interfaces, it will be necessary to have an adaptive system capable of both utilizing the information content available in the signals and learning how the signals can be used to control an existing and fixed mechanical prosthesis. These requirements will be met through the use of artificial neural networks. Thus the majority of the burden involved in training will be placed on the prosthetic device themselves, rather than the patient.

A block diagram of a complete prosthesis system is shown in Figure 6, followed by an artist's conception of such a system shown in Figure 7. For *efferent* signals, following the neural interfaces and prior to the neural network, it would be necessary to perform some feature extraction to reduce the overall data rate of the system. This is performed in several stages. Initial feature extraction would consist of demodulating the neural signals from each microelectrode site on the neural interface into numerical representations of their effective axonal firing rates. Within the stump, these demodulated signals would be multiplexed into a common signal and then transmitted to the external prosthesis hardware via a telemetry system [3]. Following this, an *adaptive* feature extractor would be utilized to cluster signals into functionally similar groups and then form an average demodulated signal for each feature signal. This additional data reduction is necessary to reduce the complexity of further processing. Finally, downstream from the feature extractor, an

[2] For the remainder of this discussion a "prosthesis" will refer to a full artificial limb prosthesis

[3] Suitable telemetry systems for bidirectional transmission of data in this application appear to be achievable with present technologies. For example, the simultaneous transmission of data and power via high-frequency electromagnetic coils has been demonstrated in prosthetic applications [10]

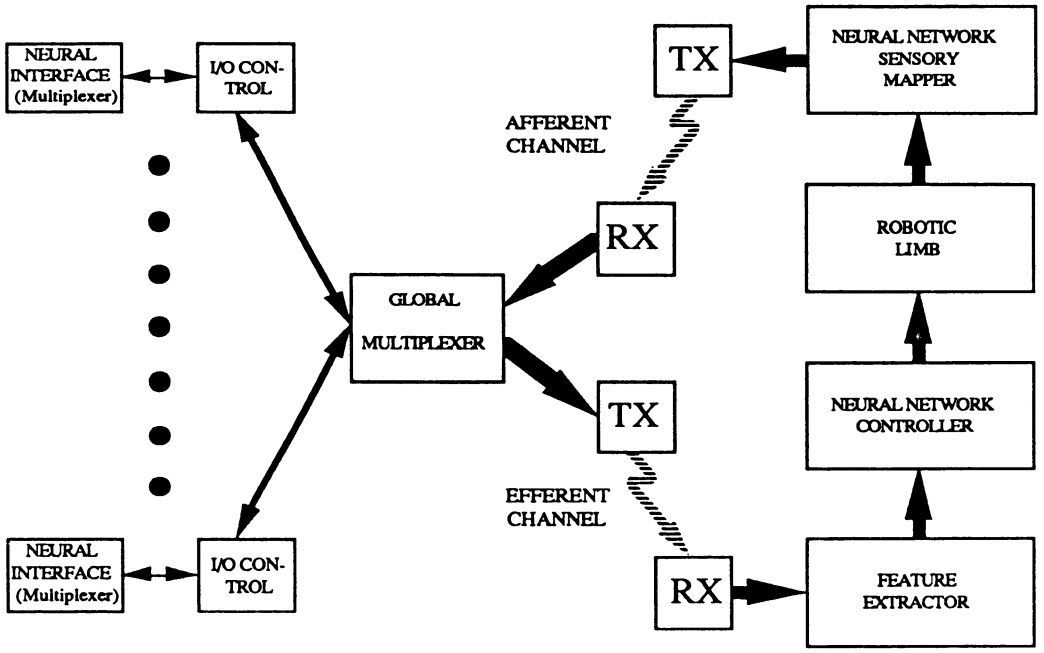


Figure 6: Block diagram of a directly interfaced prosthesis system. Signals to and from the neural interface for each fascicle are routed by I/O controllers which either demodulate or stimulate as appropriate for each microelectrode site. The data for each fascicle flows through a global multiplexer prior to the implanted transceiver. (Each transceiver is depicted as separate transmit (TX) and receive (RX) blocks.) Broken areas denote transcutaneous transmission of information. For the efferent channel, demodulated neural signals are processed by a feature extractor, followed by a neural network used to control the robotic limb. Information from transducers in the robotic limb is mapped onto the afferent channel by a neural network sensory mapper to provide feedback.

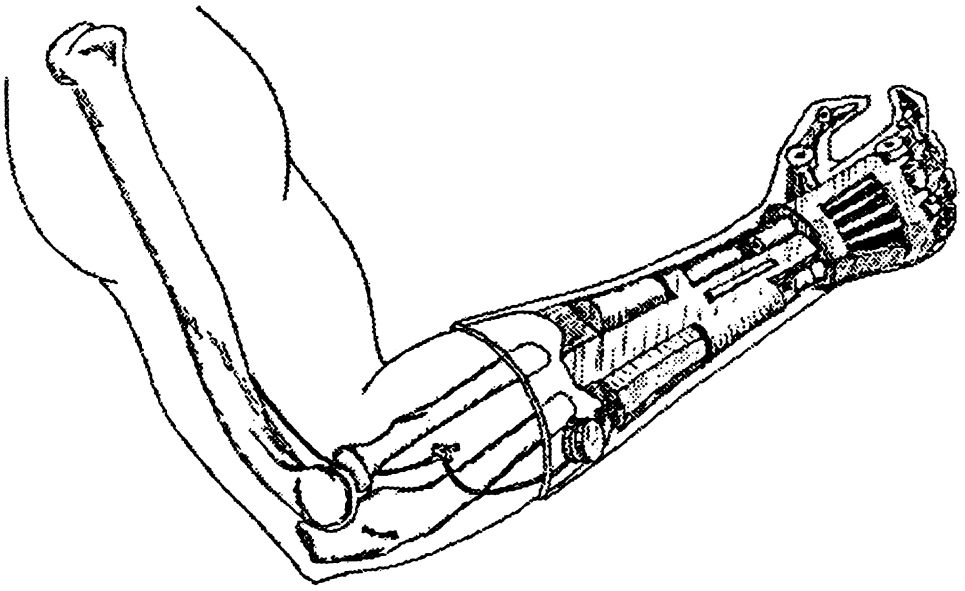


Figure 7: Artist's conception of a directly interfaced below-the-elbow limb prosthesis.

adaptive neural network in the prosthesis would carry out the interpretation of the neural command signals and the control of the mechanical systems of the prosthesis.

For afferent information, signals would be processed in a similar manner, but in a reverse direction. Signals from transducers in the prosthesis would be mapped onto the appropriate sensory channels using a second adaptive network. The outputs of the adaptive network would consist of numerical representations of the desired stimulation rates at the microelectrodes. Signals from this network would then be multiplexed into a common signal and transmitted into the stump. This information would be demultiplexed and routed to the appropriate microelectrode sites to determine their stimulation rates.

4.2 Training Methodology

The following technique could be used to establish a basic set of neural command patterns with which to train the *control* of the prosthesis and to simultaneously sort axons into afferent or efferent groups. The patient would be asked to mimic, with his or her "phantom" hand, a predefined set of motions which could be presented using a computer-generated representation of a hand. Several records of the neural firing patterns (demodulated as explained below) corresponding to each motion would be stored for later use in "off-line" training of the feature extractor and neural network control system. It may also be useful to have the patient carry out a set of hand motions with a normal hand (if present) using a device such as the "data glove" [4] to directly measure the joint angles and positions in the state space of the hand. This would allow some of the motions to be defined by the patient to better suit his or her individual needs. In order to separate afferent from efferent axons, the axonal signals which showed no consistent electrical activity during these tests would be classified as afferent. The individual microelectrode sites corresponding to them could then be defined as sites for stimulation if the prosthesis is to be equipped with sensory input transducers. The process for incorporating sensory capability in the prosthesis will require a separate procedure. While the characteristics of the transducers on the prosthesis itself will be known, sorting out the different classes of afferent fibers (tactile sense, proprioception, pain and temperature sense) is likely to be a much more arduous task. The major difficulty would be that the user, when presented with various stimuli would somehow have to report his or her perceptions to the sorting algorithm. A proposed method for training sensory information into the prosthesis which also utilizes a neural network will be presented in a later section.

5 Feature Extraction

5.1 Initial Data Reduction: Demodulation

Since the information contained in the peripheral nervous system is encoded using pulse-frequency modulation and recruitment [5], only the presence and rate of occurrence of individual action potentials would need be detected in efferent signals. Axon firing rates for normal levels of excitation fall within the range of 5 - 100 action potentials per second [26] with a conservative upper limit of 500 Hz. If each microelectrode site was equipped with a simple circuit for registering the occurrence of an action potential signal (threshold detection) between sampling intervals, it could be assumed that scanning the array at 500 Hz would provide all of the necessary information. This is in contrast to the sampled recording of action potential waveforms, which have frequency components extending out to approximately 10 KHz and hence require a higher sampling rate.

[4] The data glove is a device worn on the human hand and is used for measuring joint angles [25].

[5] Recruitment refers to increasing the number of active motor units.

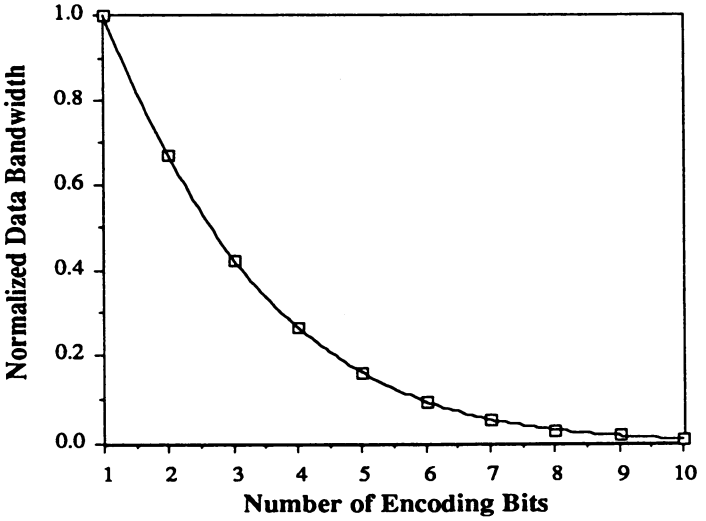


Figure 8: Graph showing the decrease in net data bandwidth with increasing number of encoding bits used to store the frequency of action potentials at a given microelectrode site.

In considering efferent data reduction, one can observe that the action potential frequency may be relatively high for each axon, while the rate of change of this frequency with changing commands for desired motions will generally be much lower. These commands (*symbols*) have a low data rate due to the relatively long mechanical time constants of the musculoskeletal system. [6] If one attempts to perform a repetitive simple hand motion, such as tapping the index finger against a tabletop as rapidly as possible, the maximum rate is a few Hertz. Thus the number of action potentials over a given sampling interval could be converted to a numerical or demodulated representation of the average firing rate without undue loss of information. The output data rate for each microelectrode site would then be the product of the number of data bits used to specify the firing rate and the reciprocal of the sampling interval. Mathematically, this is expressed as,

$$DR_{demod} = N\left(\frac{DR_{raw}}{2^N - 1}\right) \tag{7}$$

where DR_{demod} is the demodulated data rate, N is the number of bits used to specify the firing rate of the demodulated signals, and DR_{raw} is the raw data rate. A plot of this function is shown in Figure 8. The cost of adding encoding bits is both one of increased power dissipation and of increased real estate usage on the integrated circuit. The benefit in terms of reduced bandwidth is clear.

In order to estimate the total efferent bandwidth for the system, one first needs to consider the total number of axonal signal sources. Referring to the intraneural (fascicle) maps of Sunderland [22, 23]. it can be seen that the radial nerve is divided into 8-10 fascicles near the elbow. The ulnar nerve is divided into approximately 13 fascicles, and the median nerve into approximately 14, at this level. Thus roughly forty monofascicular neural interfaces would be required, for a total of 40,000 microelectrode signals. Assuming that half of these are efferent, with each site being sampled at 500 Hz, and a 5-bit symbol sample rate of 16 Hz, the system data rate would be only 1.6 Mbits/s (versus the raw signal data rate of 10.2 Mbit/s). For a slower-responding prosthesis

[6] And perhaps by bandwidth limits of the cerebrum and cerebellum which evolved concurrently and presumably without needless excess speed.

with an 8-bit encoding and a 2 Hz symbol sample rate, the system data rate would be only 320 Kbits/s (roughly a 60-fold reduction).

The demodulation of the action potential stream, from each microelectrode site, into its respective numerical signal strength could be accomplished by several methods. One such method would use what amounts to frequency counters for each axon. Either digital counters or analog charge-integrators could be utilized for this purpose. The digital counters would simply count action potentials over a given sampling period (each would require a comparator or the multiplexed output of a shared fast comparator for its clock). Each counter would be interrogated and then reset for the next sampling period. Another method, suggested by Franz [19], consists of the sequential storage of bits corresponding to the individual action potentials as time sequences for each microelectrode site [7]. Encoding logic could quickly scan the time histories for each site and output the desired numerically encoded rate value.

In a two-way neural interface, afferent information would be conveyed to the PNS as low-data-rate commands to stimulation circuitry located in the neural interface. This information would be encoded by a sensory mapper which would map signals from transducers in the prosthesis into pulse-frequency modulated signals for stimulation.

To minimize the problem of power dissipation into the neural tissue, the inclusion of the demodulation and telemetry functions in the circuitry of the neural interfaces would be avoided. The demodulators would be implemented in separate, synchronized companion chips, located in less thermally sensitive areas. The outputs from the microelectrode sites' demodulators would be scanned sequentially at the appropriate sampling intervals and routed to the telemetry circuits. Relatively high-power circuits could be located in such areas as alongside blood vessels or attached to bone. The implanted telemetry circuits could also be affixed to bone in order to facilitate heat dissipation if necessary.

5.2 Efferent Signal Clustering

In the normal neuromuscular system, thousands of functionally similar axons innervate a single group of muscle fibers to regulate its overall contraction. While individual signals could be applied directly as input to the neural network, this would be an unnecessary complication. From an engineering standpoint, we are interested only in the total muscular force level. It should thus be possible to perform massive data reduction by clustering the individual microelectrode site signals into functionally similar groups. The average firing rate within each cluster can then be used as a single representative signal for that cluster. The average of the demodulated signals from each cluster will henceforth be referred to as a *feature extracted* signal.

Ideally we would want only one feature extracted signal corresponding to each muscle in the hand. However, it will probably be necessary to have several feature extracted signals for each muscle to account for such factors as motor neuron size and recruitment order. This corresponds to simply grouping the individual axons into finer clusters. The number of necessary feature extracted signals will ultimately be determined by both the physiology of the human hand and the dexterity capabilities of the prosthesis mechanism to be used. This data reduction subsystem is shown in Figure 6 in relation to the overall prosthetic system. It is estimated that this subsystem will enable a reduction in data by roughly two orders of magnitude.

The time histories of the axon firing rates, recorded from the patient as previously described,

[7] The memory required per neural interface, based on the above estimates, is less than that of the now obsolete 64 K-bit dynamic RAM memory. Thus, for all forty fascicles a 2 M-bit RAM would be sufficient. (For 1,024 sites per neural interface and a 10 Hz sampling rate of the microelectrode time sequences, 64 storage locations per site would be adequate with a 500 Hz maximum action potential rate.)

will comprise a set of training vectors for the clustering algorithm. The dimension of the input vectors is extremely large corresponding to the sampling period and interval over which the time histories are recorded. The number of signals corresponds to the number of efferent microelectrode sites. The goal is to cluster the sites into groups which are highly correlated. This clustering can be performed in many ways, using a traditional approach such as the Linde-Buzo-Gray (LBG) algorithm [27] used in vector quantization, or a neural network approach such as Kohonen's self organizing feature maps [28].

After training, the microelectrode signals will be clustered into groups which have similar time histories. Thus the axons should be functionally grouped according the various muscles that they originally controlled. It is important to note that the adaptive process by which the clusters are determined is performed only once and off-line, using a computer. Once the appropriate cluster for each axon site is identified, the information is used to program the actual feature extraction subsystem. The programmed subsystem merely needs to average the firing rates from each axon site within each known cluster during normal operation. This average firing rate for the cluster forms the feature extracted signal which can then be used as input to the next stage of the prosthetic system, the neural network.

A possible hardware implementation for the feature extractor would entail the use of digital logic to compute the feature extracted signals. Each microelectrode site would be assigned a numerical mapping address corresponding to the cluster into which it had been classified. As the demodulated outputs from the microelectrode arrays are scanned by the feature extractor circuitry, the mapping addresses could act as pointers to summing registers into which each demodulated output would be added. These *cluster sums*, normalized by the number of microelectrode signals within each cluster, would represent the feature extracted signals at the end of each scan through the array. With such an implementation, the off-line adaptation of the feature extractor would merely produce a look-up-table of mapping addresses. These mapping addresses could then be downloaded into a non-volatile memory structure within the feature extractor to enable its operation.

More sophisticated feature extraction methods have also been considered but appear to be impractical. In EMG analysis, for example, signals are often modeled as ARMA (auto-regressive moving-average) processes [29]. One then uses a Bayesian classifier based on the parameters of the ARMA model to map neural commands onto a limited number of control sequences. Unfortunately this method is usually limited to on/off control with the intensity (force or velocity) being regulated by signal power. Also, extracting the ARMA parameters cannot be done on-line. A short time window of data must be stored and then processed. This results in unavoidable time delays. Furthermore, an EMG measures muscular activity which is based on an ensemble of neural activity. It is thus naturally amenable to stochastic analysis. Since we are working at a higher resolution than EMG signals, we would require a more sophisticated stochastic model for feature extraction. However, stochastic models are not a feasible option when considering the number of microelectrode sites involved. In addition, they do nothing to reduce the total number of distinct signals in the system.

6 Efferent Neural Network Interface

In the prosthetic system, the neural network would act as the bridge between the neural signals and the actual robotic hand. It can be considered as the *intelligent* interface between man and machine. Its function is to interpret the microelectrode signals and drive the robotic hand so as to make the use of the prosthesis transparent to the patient. To the patient, controlling the prosthesis should seem the same as controlling the original hand.

The training of the neural network will be done completely off-line using computer models of

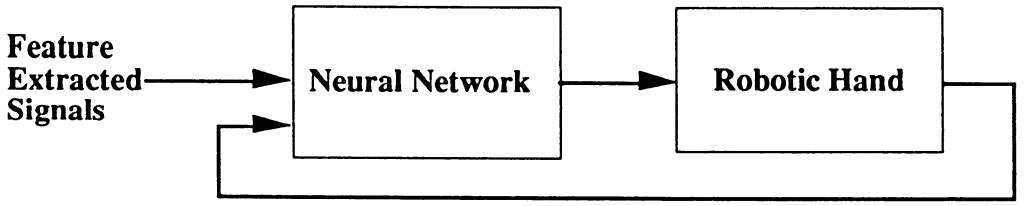


Figure 9: Block diagram of a proposed neural network based prosthesis controller.

the neural network and signal recordings made while the patient mimics desired hand movements. The patient's hand motions are sensed using a data glove whose outputs are used to form a desired response for the neural network. Once the neural network computer model is trained, the values of the synaptic weights can be downloaded to the actual neural network hardware. The customization of the prosthetic system for the individual patient should then be complete. However, additional training cycles may be necessary to emphasize certain motions to achieve fine motor control.

Prior to describing in detail the design of the neural network, it is necessary to briefly discuss how the CNS uses axons to control a limb. Simplistically, agonist and antagonist muscle contractions determine tendon tensions resulting in joint torques which ultimately determine limb position. Muscular activity is, of course, directly related to neural activity. The multiple axons associated with a single muscle regulate its contraction. A muscle is typically modeled as a simple second order dynamic system. A static hand and arm position corresponds to an *equilibrium point* in both neural and muscular activity. During the course of a movement the CNS specifies a *virtual trajectory* [30]. A point along the virtual trajectory corresponds to what the limb position would be, given that the current neural activity specifies an equilibrium point. The virtual and true trajectories are related through the inherent inertial and viscoelastic properties of the limb and muscles. (In other words, the virtual trajectory is the control input to a dynamic system, the output of which is the actual trajectory.) The underlying principle which dictates the necessary virtual trajectory control is believed to be a simple smoothness constraint on the limb trajectory [30].

It is thus clear what the neural network must accomplish. The network is provided with a set of neural recordings (the feature extracted signals) which contain information about the virtual trajectory, and the corresponding desired true trajectory (taken from data glove measurements). In order to control the robotic hand, the neural network must be capable of extracting and internally learning the original hand dynamics. In addition, it must be capable of compensating for the fixed dynamics of the robotic hand as it learns the proper control of the prosthesis. With the above in mind, we now propose two methods of implementing the neural control system.

6.1 Neural Control System

The first system shown in Figure 9 is attractive in its simplicity. A single feedforward layered neural network is used to both interpret the feature extracted signals and drive the robotic hand (RH). The output of the neural network corresponds to input levels for the various actuators which directly control torques or tensions in the mechanical hand. Training can be accomplished using a variation of the *backpropagation through time* algorithm [31]. This is a non-linear control

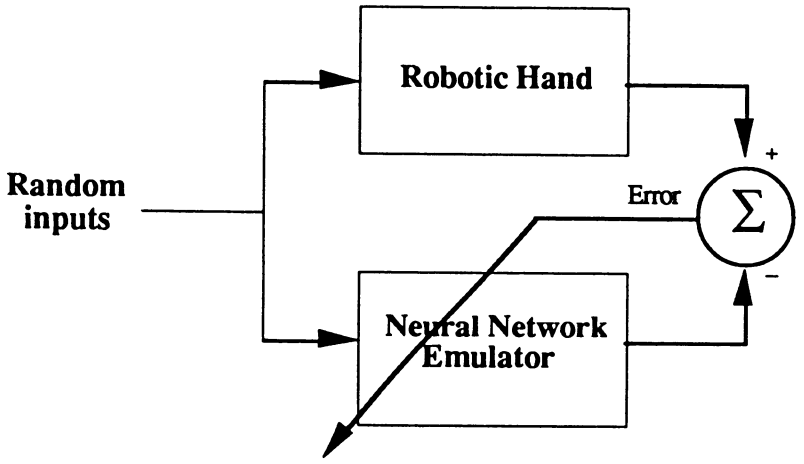


Figure 10: Block diagram of a neural network as configured for non-linear system identification to form an emulator for the robotic hand.

problem similar to the *Truck-Backer-Upper* problem [8] with some important differences. First of all, the inputs to the system are not simple command step functions, but complicated neural inputs which contain virtual trajectory information. In fact, it is somewhat misleading to think of the neural network as a controller. The CNS is the actual controller. The neural network is more of a dynamic precompensator to the mechanical system which insures the CNS control signals are properly interpreted. Furthermore, during training a desired response is available throughout the entire trajectory. This is in contrast to many control problems which only specify an end point. Thus, to properly adapt the network, it is necessary to use both the instantaneous trajectory error and prior gradients accumulated while brackpropagating through the system from previous time steps.

This training algorithm also requires a model of the robotic hand. This is necessary since the known desired response occurs at the output of the mechanical hand and not at the output of the neural network. In order to formulate the appropriate desired response for the output of the network, one needs the *Jacobian transformation* which relates small changes in the neural network output to small changes in the mechanical hand output.

In general, detailed modeling (e.g. coriolis, centrifugal, and mass matrix terms) for existing robotic systems are not readily available. As an alternative, one can form a neural network emulator of the RH. Using a neural network in a non-linear system identification mode is illustrated in Figure 10. One can then use backpropagation through the emulator to form the necessary desired response for the neural network.

6.2 Neural Interpreter System

As an alternative to the above approach, which requires the modeling of the RH, a second system is proposed which involves decoupling of the interpretation of axon signals from the low level control of the RH (see Figure 11). This allows one design of the neural network *interpreter* virtually independent of all robotic hand modeling considerations.

Existing robotic hands normally utilize low level control systems (LLCS). The LLCS for the Utah/MIT Hand [33] includes 16 variable-loop-gain position servos to operate the finger joints

[8]The *Truck-Backer-Upper* problem is a non-linear control problem in which a neural network is trained to back a truck up to a loading dock [32].

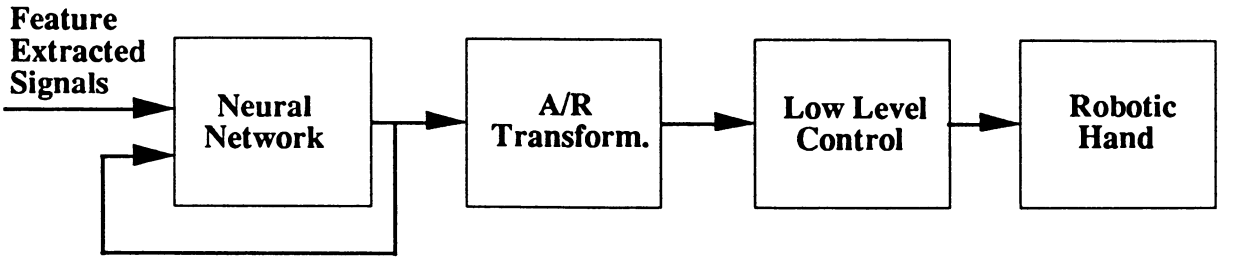


Figure 11: Block diagram of the multi-level scheme for the neural network prosthesis controller.

and 32 variable-loop-gain tension servos. For example, as input to the LLCS one can specify independently a specific joint angle. The LLCS is designed to insure that the desired angle is achieved.

Achieving higher level control of the RH using a LLCS can be accomplished via teleoperation (i.e. a human operator using a data glove). A simple linear transformation can be used to map anthropomorphic joint angles from the data glove to robotic joint angles for input to the LLCS. This transformation (referred to as the *anthropomorphic/robotic* or A/R transformation) is easily found by performing a least-squares fit over a predefined set of anthropomorphic hand poses with the desired robotic hand poses [9] [34].

Returning to the prosthetic system of Figure 11, it would now be necessary for the neural network to act only as an interpreter whose outputs correspond to the joint states of the anthropomorphic hand. Since the desired hand states that must be recorded for training the prosthesis may be taken directly from a data glove, there is now a one-to-one correspondence between the measurements taken and the desired output of the neural network. The neural network can then be adapted using backpropagation without the need for a model of the RH to be controlled. Note, backpropagation through time is still required due to the state feedback from the output of the neural network. Once trained, the output of the neural network feeds into the A/R transformation which produces inputs for the LLCS that ultimately drives the RH.

Thus it would be possible to initially train the prosthetic system without any need for an actual robotic hand and/or model. It is important to note, however, that the neural network must be trained on trajectory information so as to be able to extract the original hand dynamics. If one merely trains static mappings from neural activity into desired joint angles then one is in essence training the network to learn equilibrium points. In this case, during actual trajectory formation, the network's output will correspond to the virtual trajectory rather than the actual desired trajectory. This could, in fact, be desirable if the composite LLCS and RH dynamic system could be made to correspond to actual human hand dynamics. In this case the virtual neural trajectory formed by the CNS would be transformed into a virtual joint trajectory by the neural network which would then be transformed into the actual trajectory by the LLCS and RH. Unfortunately, it would be incorrect to assume that the mechanical prosthesis could be designed to match the dynamic properties of a human limb.

Initially, it is more reasonable to assume that the dynamics of the LLCS and RH are fast enough

[9] A transformation which includes both joint angle and joint stiffness will probably be desirable. This will require further research in order to implement.

to be ignored. In this case, without appreciable mechanical time constants, the neural network output must be made to correspond to the actual trajectory. This resulting inability to adaptively specify the overall dynamic characteristics, including robotic dynamics, is a minor drawback to this scheme. The patient's own ability to compensate for a "mismatch" in dynamics will need to be studied. However, it is evident that a person can easily compensate for fluctuations in dynamics in the natural hand as demonstrated by the ease with which we can manipulate objects.

It may appear from Figure 11 that control of the prosthesis is essentially being performed in an open loop fashion. Feedback does occur, however, both visually and through the afferent pathways. Even in the natural hand no feedback occurs internal to the hand itself.

7 Afferent Neural Network Sensory Mapper

The mapping of signals from transducers in the prosthesis onto the afferent neural channels would be controlled by a neural network based sensory mapper. The prosthesis could readily be equipped with transducers for tactile sensation, force, joint angle, temperature, and other sensory modalities. Utilizing such transducers would require the assignment of their output signals to neural pathways of the appropriate sensory modalities and perceived locations on the prosthesis.

The input to the neural network corresponds to the transducer outputs whose characteristics will be known. Forming the *desired response* for the network, however, will require the knowledge of the sensory modality with which each afferent microelectrode site is associated. The process of sorting out the various sensory modalities will likely prove to be a more difficult task than the utilization of the efferent information. Fortunately, the biological organization of the PNS is such that the fascicles are somewhat distinct in terms of dermatomes, muscles, tendons and joints innervated. Utilizing fascicle function maps such as those of Sunderland [22, 23] and Jabaley, et al [24], the fascicles could at least be tentatively identified and this information utilized to speed their initial classifications (or to coordinate the global functions of simple prostheses) using techniques described below. Thus the gross somatotopic location of each neural interface would likely be known as the nerve and perhaps the fascicle into which each interface were implanted would be known at the time of surgery. A proposed method for determining the modality and perceived location of each afferent microelectrode site involves a process of systematic stimulation as outlined below.

Initially the perceived locations of the entire group of afferent sites on each neural interface would be verified by their stimulation *en masse*. The patient would report the location at which the sensation appears. At this point, low-resolution areas of perceived sensation, or *fascicular dermatomes*, would be known and may well prove adequate for initial prostheses. The subdivision of each fascicular dermatome into higher-resolution regions and perhaps distinct sensory modalities could subsequently be attempted.

Stimulation of small groups of microelectrode sites, in numbers chosen such that they are at or slightly above the perceptual threshold, should allow the patient to indicate more precisely the locations of the perceived stimuli. While it is presently unknown if distinct modalities could be resolved in this fashion, future experimentation will undoubtedly yield a better understanding of how this could be accomplished. Currently one can only speculate about what the patient would actually perceive under stimulation. The only reports of such work known at this time [35, 36], indicate that only "tingling sensations" were described under gross stimulation of nerves.

Regardless of the ultimate resolution at which stimuli could be delivered to the patient, a neural network sensory mapper would be required to correctly distribute signals from the transducers on the prosthesis to the neural interfaces. The neural network inputs would be the transducer signals and the outputs would be numerical values representing the stimulus intensity required at each microelectrode site. These numerical outputs would be transmitted transcutaneously, as shown

in Figure 6, and converted to pulse-frequency modulated streams of stimulus current pulses by implanted circuitry.

In order to train the neural network sensory mapper, one could use the stimulation intensities at each microelectrode site associated with perceived sensory locations and modalities (determined as described above) for the training set of desired outputs. The corresponding inputs would be derived from the known characteristics of the transducers used. Given this training set, the feed-forward neural network could be trained to form an appropriate mapping from transducers to the microelectrode sites. Initial training could be done off-line, with fine tuning carried out with the interaction of the patient.

8 Additional Applications for Neural Network Interfaces

Additional uses for the neural interfaces and processing circuitry would be abundant. Once the prosthesis interface is established, the processing circuitry could be trained for alternative devices which could be connected to the patient. Thus the neural interface and associated circuits would constitute an extremely versatile man/machine interface.

Control of mechanical devices could be accomplished without the mechanical lag of the hand. For example, the control surfaces of an aircraft could be directly mapped to hand motions. Sensory ranges could be compressed or expanded to suit many applications. Microscopic manipulations, such as those of microsurgery, could be mapped into perceived macroscopic motions. New sense modalities could also be introduced. For example, radiation could be sensed using the appropriate transducers and mapped into temperature sensations. It should be noted that since information is bidirectionally transmitted into and out of the neural interfaces in a form suitable for telemetry, remote operation would also be inherently possible in these and other applications.

In addition, these devices will allow for a great deal of basic science research which should answer fundamental questions regarding the nature of neural information conveyed by the PNS.

9 Conclusion

This article has presented an overview of the neural interface technology under development by this group. Also presented was the use of such interfaces in nerve repair as well as proposed implementations for direct neural network interfaced hand prostheses. Naturally, other types of limb prostheses could eventually be realized using similar approaches. In order to achieve the goal of realistic and cost-effective devices, an active effort is being made to avoid the use of expensive and esoteric materials and fabrication processes.

The effort to realize such a prosthesis is a long-term, multidisciplinary project. It is expected that it will be on the order of a decade before clinically useful devices can be produced. In the meantime, it is hoped that some of the technologies developed in the course of the overall project will find uses in rehabilitation and basic science research.

10 Acknowledgments

This research has been funded under Veteran's Administration Rehabilitation Research and Development (RR&D) Merit Review Grant B003 Hentz/Rosen "Towards Better Methods of Nerve Repair and Evaluation". In addition, the authors wish to extend their sincere thanks to Doug Franz of Hewlett-Packard Inc. for his constant enthusiasm and intellectual input to the project, to Chris Storment for his ongoing efforts to realize the fabrication technologies required and for providing many of the scanning electron micrographs presented herein, to Woody Knapp for his

illustrations, to Nancy Kelly and Joanne Kipp for their help in producing the SEM's and TEM's of biological tissues, to Martin Brooks for his comments on this paper, and to our many friends and collaborators who have "lent us a hand" with the research.

References

[1] G. T. A. Kovacs, E. A. Wan, B. Widrow, J. M. Rosen, "Development of Limb Prostheses with Direct Neural Interfaces", submitted to *IEEE Transactions on Biomedical Engineering.*

[2] A. F. Marks, "Bullfrog Nerve Regeneration into Porous Implants", *Anatom. Rec.*, 163, 226, 1969.

[3] R. Llinas, C. Nicholson, K. Johnson, "Brain Unit Activity During Behaviour", *M. Phillips (Ed.)*, Thomas, Springfield, IL, pp.105-111, 1973.

[4] A. Mannard, R. B. Stein, D. Charles, "Regeneration Electrode Units: Implants for Recording from Peripheral Nerve Fibers in Freely Moving Animals", *Science*, Vol. 183, pp. 547-549, February 1974.

[5] G.E. Loeb, W. B. Marks, P. G. Beatty, "Analysis and Microelectronic Design of Tubular Electrode Arrays Intended for Chronic, Multiple Single-Unit Recording from Captured Nerve Fibers", *Medical & Biological Engineering & Computing,* Vol.15, pp.195-201, March 1977.

[6] T. Matsuo, A. Yamaguchi, M. Esashi, "Fabrication of Multi-Hole-Active Electrode for Nerve Bundle", *Journal of Japan Soc. ME & BE*, July 1978.

[7] D. J. Edell, "Development of a Chronic Neuroelectric Interface", *Doctoral Dissertation,* University of California, Davis, 1980.

[8] D.J. Edell, L. D. Clark, V. M. McNeil, "Optimization of Electrode Structure for Chronic Transduction of Electrical Neural Signals", *IEEE Engineering in Medicine and Biology Society, Proceedings of the 9th Annual International Conference*, Texas, November 1986.

[9] J.M Rosen, G. T. A. Kovacs, M. C. Stephanides, D. Marshall, M, Grosser, V. R. Hentz, "The Development of a Microelectronic Axon Processor Silicon Chip Neuroprosthesis", *Proceedings of the 10th Annual Conference of the Association for the Advancement of Rehabilitation Technology,* pp. 675-677, June 1987.

[10] Galbraith, D. C., "An Implantable Multichannel Neural Stimulator", Doctoral Dissertation, Stanford University Department of Electrical Engineering, Technical Report No. G908-2, December, 1984.

[11] G. T. A. Kovacs, Doctoral Dissertation in the Department of Electrical Engineering, *Stanford University*, in preparation.

[12] G. T. A. Kovacs, M. C. Stephanides, W. R. Knapp, J. P. McVittie, V. R. Hentz, J. M. Rosen, "Development of Chronic Implant Neural Prosthesis Microelectrode Arrays", *Proceedings of the IEEE Montech Conference on Biomedical Technologies*, Montreal, PQ, pp. 152-155, November 1987.

[13] G. T. A. Kovacs, M. C. Stephanides, W. R. Knapp, J. P. McVittie,J. M. Rosen, "Design of Two-Dimensional Neural Prosthesis Microelectrode Arrays", *IEEE Engineering in Medicine and Biology Society, Proceedings of the 9th Annual International Conference,* Boston, MA, pp. 1651-1652, November 1987.

[14] G. T. A. Kovacs, C. W. Storment, J. P. McVittie, W. R. Knapp, V. R. Hentz, J. M. Rosen, "Advances Toward Development of Microelectronic Axonal Interface Neuroprostheses", *Proceedings of the International Conference of the Association for the Advancement of Rehabilitation Technology*, Montreal, PQ, pp. 362-363, June 1988.

[15] G. T. A. Kovacs, C. W. Storment, B. Jemes, V. R. Hentz, J. M. Rosen, "Design and Implementation of Two-Dimensional Neural Interfaces", *IEEE Engineering in Medicine and Biology Society, Proceedings of the 10th Annual International Conference*, New Orleans, LA, pp. 1649-1650, November 1988.

[16] G. T. A. Kovacs, et al, "A Deep Plasma Etch Process for the Fabrication of Chronic Neural Interfaces", *in preparation.*

[17] G. T. A. Kovacs, et al, "Development and In-Vivo Testing of Thin-Film Iridium Microelectrode Arrays for Chronic Neural Interfaces", *in preparation.*

[18] J. J. Hopfield, D. W. Tank, " *Neural* Computations in Optimization Problems", *Biological Cybernetics*, 52. pp 141-152, 1985.

[19] D. Franz, personal communication

[20] C.J. De Luca, "Control of Upper-Limb Prostheses: A Case For Neuroelectric Control",*Journal of Medical and Engineering Technology*, Vol.2, No. 2, pp. 51-61, March 1978.

[21] Dellon, A.L., Evaluation of Sensibility and Re-Education of Sensation in the Hand, Williams & Wilkins, Baltimore, 1981.

[22] S. Sunderland, *Nerves and Nerve Injuries*, Churchill Livingstone, Edinburgh, 1972.

[23] S. Sunderland, "Intraneural Topography of the Radial, Median and Ulnar Nerves", *Brain*, Vol. 68, 243, pp.17-299, 1945.

[24] M. E. Jabaley, H. W. Wallace, F. R. Heckler, "Internal Topography of Major Nerves of the Forearm and Hand: A Current View", *Journal of Hand Surgery*, Vol. 5, No. 1, pp. 1-18, January 1989.

[25] VPL Research, "DataGlove Operation Manual", Redwood City, CA.

[26] E. R. Kandel, J. H. Schwartz, "Principles of Neural Science", Elsevier, pp. 22-24, 1985.

[27] Y. Linde, A. Buzo, and R.M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Comm.*, vol. COM-28, pp. 84-95, Jan. 1980.

[28] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics,* vol. 43, pp. 59-69, 1982.

[29] D. Groupe, W. Cline , "Functional Separatation of EMG Signals via ARMA Identification Methods for Prosthesis Control Purposes," *IEEE Transactions on Systems, Man, and Cybernetics*, vol, smc-5, no. 2, pp. 252-259, march 1975.

[30] N. Hogan, "An Organizing Principle for a Class of Voluntary Movements," *Journal of Neuroscience*, vol. 4, no. 11, pp. 2745-2754, November 1984.

[31] D.E. Rumelhart, J.L. McClelland, and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, The MIT Press, Cambridge, MA, 1986.

[32] D. H. Nguyen and B. Widrow "The truck backer-upper: An Example of Self Learning in Neural Networks," *Proceedings of the International Joint Conference on Neural Networks,* Washington, June 1989.

[33] S. Jacobsen, E. Iverson, et al., "Design of the Utah/Mit Dextrous Hand," *IEEE International Conference on Robotics and Automation,* TJ 211.I21, pp. 1520-1532, 1986.

[34] L. Pau. T. Speeter, "Transformation of Human Hand Positions for Robotic Hand Control," *in preparation.*

[35] R. B. Stein, T. Gordon, J. A. Hoffer, L. A. Davis, D. Charles, Long-Term Recordings from Cat Peripheral Nerves During Degeneration and Regeneration: Implications for Human Nerve Repair and Prosthetics, In: Nerve Repair: Its Clinical and Experimental Basis, Edited by D.L. Jewett and H.R. McCarroll, St. Louis: C.V. Mosby, pp. 166-176, 1980.

[36] C. De Luca, L. D. Gilmore, L. J. Bloom, S. J. Thomson, A. L. Cudworth, M. J. Glimcher, " Long-Term Neuroelectric Signal Recording from Severed Nerves", *IEEE Transactions on Biomedical Engineering*, Vol. BME-29, No. 6, pp. 393-402, June 1982.

Special Lectures on Self-Organizing Neural Architectures

Time—the Essential Dimension

Carver Mead

The architectures of animal nervous systems are shaped by evolution and carried in the genetic code. The essential quality of such an architecture is that it must *learn* from the environment in which the animal lives. The single common element of learning is the time coincidence in the arrival of nerve impulses. The arrival of one impulse or burst of impulses closely followed by another is taken by the nervous system as evidence for a cause-and-effect relationship between the two. This paradigm has proven effective because much of the animal's world experience consists of temporally coherent phenomenon. All laws of physics can be phrased in terms of the expected evolution of objects in the world through time. These physical laws are the source of coherence in sensory input, which drives the fine-grained organization of the developing nervous system. Marr and others have emphasized the constraints that the laws of physics exert on what sensory input is possible. More recently it is becoming clear that these same laws of physics constrain the possible architectures of the brain itself. Highly effective computing structures result when the representation within the nervous system takes advantage of these physical laws to mirror the behavior of the physical world.

Self-Organizing Neural Architectures for Motion Perception, Adaptive Sensory-Motor Control, and Associative Mapping

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

1. Introduction

This lecture will describe recent results that are based upon two lines of previous research: the Grossberg and Rudd (1989) Motion OC Filter, which has been used to explain and predict data concerning short-range and long-range apparent motion; and the Bullock and Grossberg (1988a, 1988b) Vector-Integration-To Endpoint, or VITE, neural command circuit, which has been used to explain and predict data concerning arm and speech articulator trajectories and self-organization of associative maps.

2. Experimental Evidence for a Motion OC Filter

Part I of the lecture will describe analyses by Grossberg and Rudd of additional properties of apparent motion that lend further support to the Motion OC Filter. These properties include a unified explanation of Korte's Laws (Korte, 1915), delta (or reverse) motion (Kolers, 1972), visual inertia (Anstis and Ramachandran, 1987), and competitive selection of motion pathways (Burt and Sperling, 1981).

3. CC Loop: Cooperation and Resonance During Emergent Segmentation

Properties of coherent motion segmentation may be derived by attaching the Motion OC Filter to a cooperative-competitive feedback loop called the CC Loop. Grossberg and Mingolla (1985a, 1985b) have suggested that the CC Loop plays a key role during static form perception. Their predictions have recently received additional support from neurophysiological experiments demonstrating cooperative sharpening of receptive fields by cortical interactions (Peterhans and von der Heydt, 1989) and cortical resonance during cooperative linking of oriented receptive fields (Eckhorn *et al*, 1988; Gray *et al*, 1989). Here CC Loop circuits are suggested to also play a basic role in motion segmentation, notably in motion capture by real and illusory boundaries of global surface properties (Ramachandran, 1985, 1986).

The Motion OC Filter and CC Loop comprise a Motion Boundary Contour System (BCS). This system generates motion segmentations that are insensitive to direction-of-contrast but sensitive to direction-of-motion. The BCS for static form perception generates segmentations that are insensitive to direction-of-contrast and insensitive to direction-of-motion. Why are both systems needed? Why not just use a Motion BCS?

4. The Symmetric Unfolding of Cortical Opponent Processes

A further analysis suggest why cortical systems for the analysis of both static form and moving form exist (Grossberg, 1989). These parallel systems compute all possible ways of symmetrically gating opponent pairs of sustained cells with transient cells to generate output signals that are insensitive to direction-of-contrast. The result is a symmetric Four Fold Way (Figure 1) that constrains the development of visual cortex. Opponent cell pairs in the static OC filter define complex/orientation/on-cells and complex/orientation/off-cells. Opponent cell pairs in the Motion OC Filter define complex/direction cells. Using this scheme, opponent cell pairs in the static BCS define orientations that differ by 90° as opposites, and in the motion BCS define directions that differ by 180° as opposites. When these opponent cell pairs are embedded into gated dipole opponent processes (Grossberg, 1982, 1988), their antagonistic rebounds clarify data about negative afterimages of Moiré

This research was supported in part by the Air Force Office of Scientific Research (F49620-87-C-0018), the Army Research Office (DAAL-03-88-K-0088), and the National Science Foundation (IRI-87-16960).

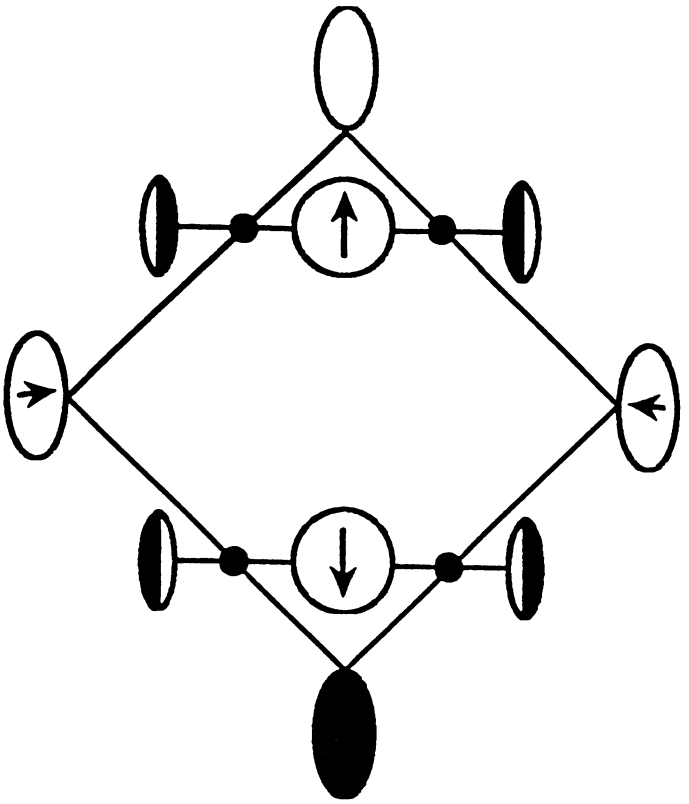


Figure 1. Four-fold symmetry of orientation cells and direction cells: Oriented sustained cells that are sensitive to direction-of-contrast are gated by transient on-cells and off-cells before being combined into opponent pairs of orientation cells and direction cells whose output signals are independent of direction-of-contrast.

patterns (MacKay, 1957), the waterfall illusion (Sekuler, 1975), adaptation of the short-range motion system (Anstis and Mather, 1985), and the existence of opponent direction hypercolumns in MT (Albright, Desimone, and Gross, 1984).

5. Preattentive Resonance and Reset

Another consequence is rapid reset of resonating segmentations. The CC Loop is capable of nonlinear resonance due to its positive feedback loops. Such resonant feedback generates a sharp and stable segmentation of a scene. It could also cause a perseverative neural trace that could prevent rapid adjustment to changing scenes. Opponent cell pairs capable of antagonistic rebound in response to input offset can rapidly reset a resonating segmentation and prepare it to form the next segmentation with minimal bias. Coexistence of persistent resonance and rapid reset is hereby achieved using preattentive mechanisms, rather than the attentive mechanisms of Adaptive Resonance Theory.

6. A Self-Organizing VITE Circuit

Part II of the lecture will derive architectures for self-organization of sensory-motor coordination. The Bullock and Grossberg (1988a) VITE model generates synchronous, variable-speed trajectories of a multi-jointed limb in response to a target position command (TPC) and a GO signal, or a sequence thereof. Several components of this circuit are in close accord with recent neurophysiological data: TPCs with parietal cortex (Hyvärinen, 1982; Lynch, 1980), the GO signal with globus pallidus (Horak and Anderson, 1984a, 1984b), and the difference vector (DV) with motor cortex (Georgopoulos, Kettner, and Schwartz, 1988; Georgopoulos, Schwartz, and Kettner, 1986). See Bullock and Grossberg

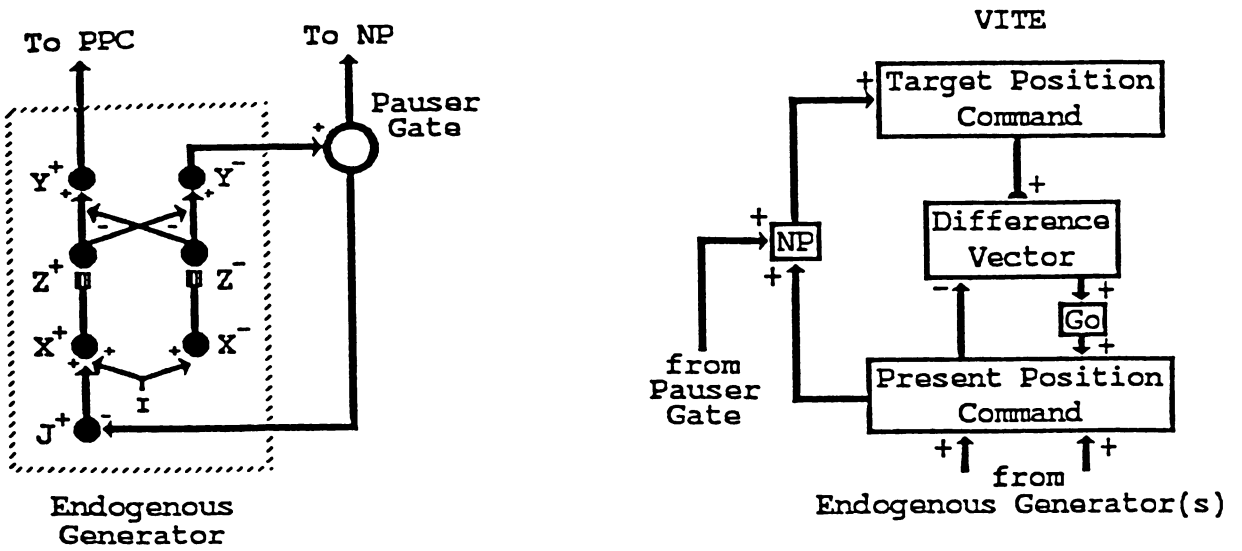


Figure 2. An AVITE circuit: The left column schematizes an endogenous generator of random training vectors composed of gated dipoles. The On channels (+ superscript) generate random unbiased training vectors, while the Off channels (-) activate the pauser gate.The right column shows how the endogenous generator influences VITE learning. The endogenous generator inputs random vectors to the Present Position Command (PPC), where they are integrated until the pauser gate is activated. Inputs to the On channels of the generator then terminate, the Now Print (NP) channel is inhibited, and the PPC is copied into the TPC. Then learning in the TPC→DV pathway zeroes the DV, and thereby adaptively calibrates TPC→DV signals to be computed in the same coordinates as PPC→DV signals. When the On channel transmitter gates recover, the cycle begins again.

(1988a, 1988b, 1989) and Grossberg and Kuperstein (1989) for further discussion. Bullock and Grossberg (1988b) have also outlined how the VITE circuit TPC may be activated during eye-hand coordination via an intermodal associative map by TPCs of the eye-head system. These results have recently been extended to show how VITE parameters are adaptively self-calibrated and how intermodal associative maps are learned through a self-organizing process.

7. The Role of Circular Reactions in Sensory-Motor Self-Organization

These analyses clarify how a child, or infant robot, can learn to reach for objects that it sees. Piaget (1963) has provided basic insights with his concept of a *circular reaction.* When an infant makes internally generated movements of his hand, the eyes automatically follow this motion. As he fixates the hand at a variety of positions, a transformation is learned between the eye-head system and the hand-arm system. As learning progresses, the reverse transformation is also learned, eventually enabling the child to touch what he sees. Thus the circular reaction is based upon endogenously generated actions whose commands are correlated with sensory feedback by means of an associative transform.

Grossberg and Kuperstein (1986, 1989) showed how eye movements in response to visual inputs could correct their parameters based upon visual error signals. Gaudiano and Grossberg (1990a, 1990b) have shown how the arm movement system can, without an external teacher, endogenously generate movements whose consequences are used to adaptively self-tune internal parameters of the arm's VITE circuit and to generate the data that are used to learn an intermodal associative transformation. Kuperstein (1988) has also utilized circular reactions to learn an eye-hand transformation. His model also builds upon Grossberg and Kuperstein (1986), but the circuits described herein differ from his.

The adaptive VITE circuit, also called an AVITE circuit, is schematized in Figure

2. It includes a self-regulating generator of random training vectors that can be used to learn sensory-motor or motor-motor transformations during a circular reaction. The generator is biphasic: The generation of each vector induces a complementary quiescent phase during which learning occurs. Then a new vector is generated and the cycle repeats itself. This biphasic behavior is controlled by a specialized gated dipole circuit (Figure 2). The remainder of the AVITE circuit design shows how adaptive parameter learning and random vector search can be combined into a fully self-organizing system. The result is a specialized type of *adaptive vector encoder* (Grossberg, 1988) whereby an autonomous agent may relearn its operating parameters while in the field. How intermodal associative maps self-organize between AVITE TPCs and other internal representations will also be discussed.

References

[1] Albright, Desimone, and Gross (1984). *J. Neurophysiology*, **51**, 16.
[2] Anstis, S.M. and Mather, G. (1985). *Perception*, **14**, 167.
[3] Anstis, S.M. and Ramachandran, V.S. (1987). *Vision Research*, **5**, 755.
[4] Bullock, D. and Grossberg, S. (1988a). *Psychological Review*, **95**, 49.
[5] Bullock, D. and Grossberg, S. (1988b). In J.A.S. Kelso, J.A. Mandell, and M.F. Shlesinger (Eds.), **Dynamic patterns in complex systems.** Singapore: World Scientific Publishers.
[6] Bullock, D. and Grossberg, S. (1989). In W. Hershberger (Ed.), **Volitional action.** North-Holland.
[7] Burt, P. and Sperling, G. (1981). *Psychological Review*, **88**, 171.
[8] Eckhorn, R., Bauer, R., Jordan, W, Brosch, M., Kurse, W. Munk, M. and Reitboeck, H.J. (1988). *Biological Cybernetics*, **60**, 121.
[9] Gaudiano, P. and Grossberg, S. (1990a). In **Proceedings of the International Joint Conference on Neural Networks**, Hillsdale, NJ: Erlbaum.
[10] Gaudiano, P. and Grossberg, S. (1990b). A self-organizing neural circuit for control of planned movement trajectories. In preparation.
[11] Georgopoulos, A.P., Kettner, R.E., and Schwartz, A.B. (1988). *J. Neuroscience*, **8**, 2928.
[12] Georgopoulos, A.P., Schwartz, A.B., and Kettner, R.E. (1986). *Science*, **233**, 1416.
[13] Gray, C.M., Konig, P., Engel, A.K., and Singer, W. (1989). *Nature*, **338**, 334.
[14] Grossberg, S. (1982). **Studies of mind and brain: Neural principles of learning, perception, development, cognition, and motor control.** Boston: Reidel Press.
[15] Grossberg, S. (1988). **Neural networks and natural intelligence.** Cambridge, MA: MIT Press.
[16] Grossberg, S. (1989). *Mind and Language*, in press.
[17] Grossberg, S. and Kuperstein, M. (1986). **Neural dynamics of adaptive sensory-motor control: Ballistic eye movements.** North-Holland.
[18] Grossberg, S. and Kuperstein, M. (1989). **Neural dynamics of sensory-motor control**, Expanded edition. Elmsford, NY: Pergamon Press.
[19] Grossberg, S. and Mingolla, E. (1985a). *Psychological Review*, **92**, 173.
[20] Grossberg, S. and Mingolla, E. (1985b). *Perception and Psychophysics*, **38**, 141.
[21] Grossberg, S. and Rudd, M. (1989) *Neural Networks*, in press.
[22] Horak, F.B. and Anderson, M.E. (1984a). *J. Neurophysiology*, **52**, 290.
[23] Horak, F.B. and Anderson, M.E. (1984b). *J. Neurophysiology*, **52**, 305.
[24] Hyvārinen, J. (1982). *Physiol. Reviews*, **62**, 1060.
[25] Kolers, P.A. (1972). **Aspects of motion perception.** Oxford: Pergamon Press.
[26] Korte, A. (1915). *Zeitschrift für Psychologie*, **72**, 194.
[27] Kuperstein, M. (1988). *Science*, **239**, 1308.
[28] Lynch, J.C. (1980). *Behavioral and Brain Sciences*, **3**, 485.
[29] MacKay, D.M. (1957). *Nature*, **180**, 849.
[30] Peterhans, E. and von der Heydt, R. (1989). *J. Neuroscience*, **9**, 1749.
[31] Piaget, J. (1963). **The origins fo intelligence in children.** New York: Norton.
[32] Ramachandran, V.S. (1985). *Perception*, **14**, 127.
[33] Ramachandran, V.S. (1986). *Perception and Psychophysics*, **39**, 361.
[34] Sekuler, R. (1975). In E.C. Carterette and M.P. Friedman (Eds.), **Handbook of perception, vol. V: Seeing.** New York: Academic Press.

ART 3 HIERARCHICAL SEARCH: CHEMICAL TRANSMITTERS IN SELF-ORGANIZING PATTERN RECOGNITION ARCHITECTURES

Gail A. Carpenter and Stephen Grossberg
Center for Adaptive Systems, Boston University
111 Cummington Street, Boston, MA 02215

1. Introduction

A model to implement parallel search of compressed or distributed pattern recognition codes in a neural network hierarchy is described. The search process functions well with either fast learning or slow learning, and can robustly cope with sequences of asynchronous input patterns in real-time. The search process emerges when computational properties of the chemical synapse, such as transmitter accumulation, release, inactivation, and modulation, are embedded within an Adaptive Resonance Theory architecture called ART 3. Formal analogs of ions such as Na^+ and Ca^{2+} control nonlinear feedback interactions that enable presynaptic transmitter dynamics to model the postsynaptic short term memory representation of a pattern recognition code. Reinforcement feedback can modulate the search process by altering the ART 3 vigilance parameter or directly engaging the search mechanism. The search process is a form of hypothesis testing capable of discovering appropriate representations of a nonstationary input environment.

2. Equations for Transmitter Production, Release, and Inactivation

The search mechanism works well if it possesses a few basic properties. These properties can be realized using one of several closely related sets of equations, with corresponding differences in biophysical interpretation. An illustrative system of simplified equations is described below. Equations (1)–(3) govern the dynamics of the variables z_{ij}, u_{ij}, v_{ij}, and x_j at the ij^{th} pathway and j^{th} node of an ART 3 system.

Presynaptic Transmitter

$$\frac{du_{ij}}{dt} = (z_{ij} - u_{ij}) - u_{ij}[\text{release rate}] \tag{1}$$

Bound Transmitter

$$\begin{cases} \frac{dv_{ij}}{dt} = -v_{ij} + u_{ij}\ [\text{release rate}] & \text{if reset} = 0 \\ v_{ij}(t) = 0 & \text{if reset} >> 1 \end{cases} \tag{2}$$

Postsynaptic Activation

$$x_j(t) = \begin{cases} \sum_i v_{ij} + \ [\text{intrafield feedback}] & \text{if reset} = 0 \\ 0 & \text{if reset} >> 1. \end{cases} \tag{3}$$

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), and the National Science Foundation (NSF DMS-86-11959 and IRI-87-16960).

3. Transmitter Release Rate

ART Search Hypothesis 1:
Presynaptic transmitter u_{ij} is released at a rate jointly proportional to the presynaptic signal S_i and a function $f(x_j)$ of the postsynaptic activity. That is, in equations (1) and (2),

$$\text{release rate} = S_i f(x_j). \tag{4}$$

In the simulations, $f(x_j)$ is linear above a small negative threshold.

The form factor $S_i f(x_j)$ may be compared to interactions between voltages and ions, where S_i depends upon Na^+; $f(x_j)$ on Ca^{2+}; and transmitter release on their *joint* fluxes.

4. System Dynamics at Input Onset: An Approximately Linear Filter

Assume that $u_{ij}(0) = z_{ij}$ and $x_j(0) = v_{ij}(0) = 0$. During a time interval $t = 0^+$ immediately after a signal S_i arrives at the synapse, the ART equations approximate a linear filter:

$$\frac{dv_{ij}}{dt} \approx z_{ij} S_i f(0) \tag{5}$$

and so

$$v_{ij}(t) \approx K(t) S_i z_{ij} \quad \text{for times } t = 0^+. \tag{6}$$

By (3) and (6),

$$x_j(t) \approx \sum_i K(t) S_i z_{ij} = K(t)\mathbf{S} \cdot \mathbf{z}_j \quad \text{for times } t = 0^+. \tag{7}$$

5. System Dynamics After Intrafield Feedback: Amplification of Transmitter Release by Postsynaptic Potential

In the next time interval, the intrafield feedback signal contrast-enhances the initial signal pattern (7) via equation (3) and amplifies the total activity. These intrafield nonlinearities, coupled with the nonlinear transmitter dynamics described below, correct coding errors by triggering a parallel search, allow the system to respond adaptively to reinforcement, and rapidly reset to changing input patterns. Term $u_{ij} S_i f(x_j)$ for the amount of transmitter released per unit time implies that the original incoming weighted signal $z_{ij} S_i$ is distorted both by depletion of presynaptic transmitter u_{ij} and by the activity x_j of the postsynaptic cell. In particular, once activity in a postsynaptic cell becomes large, this activity dominates the transmitter release rate, via the term $f(x_j)$. Thus, although linear filtering properties initially determine the small-amplitude activity pattern of the target field, once intrafield feedback amplifies and contrast-enhances the postsynaptic activity x_j, it plays a major role in determining the amount of released transmitter v_{ij}. The postsynaptic activity pattern across the field that represents the recognition code is imparted to the pattern of released transmitter, which then also represents the recognition code, rather than the initial filtered pattern $\mathbf{S} \cdot \mathbf{z}_j$.

6. System Dynamics During Reset: Inactivation of Bound Transmitter Channels

The dynamics of transmitter release implied by the ART Search Hypothesis 1 can be used to implement the reset process, by postulating the

ART Search Hypothesis 2:
The nonspecific reset signal quickly inactivates postsynaptic membrane channels at which transmitter is bound.

The reset signal in equations (2) and (3) may be interpreted as assignment of a large value to the inactivation rate of bound transmitter in a manner analogous to the action of

a neuromodulator. Inactivation breaks the strong intrafield feedback loops that implement ART 3 matching and contrast-enhancement.

Following inactivation, the pattern of released transmitter forms a representation of the postsynaptic recognition code. The reset signal implies that the system has judged this code to be erroneous, according to some criterion. The ART Search Hypothesis 1 implies that the largest concentrations of bound extracellular transmitter are adjacent to the nodes which most actively represent this erroneous code. The ART Search Hypothesis 2 therefore implies that the reset process selectively removes transmitter from pathways leading to the erroneous representation.

After the reset wave has acted, the system is biased against activation of the same nodes, or features, in the next time interval: Whereas the transmitter signal pattern $\mathbf{S} \cdot \mathbf{u}_j$ originally sent to target nodes at times $t = 0^+$ was proportional to $\mathbf{S} \cdot \mathbf{z}_j$, as in equation (6), the transmitter signal pattern $\mathbf{S} \cdot \mathbf{u}_j$ after the reset event is no longer proportional to $\mathbf{S} \cdot \mathbf{z}_j$. Instead, it is selectively biased against those features that were previously active. The new signal pattern $\mathbf{S} \cdot \mathbf{u}_j$ will lead to selection of another contrast-enhanced representation, which may or may not then be reset. This search process continues until an acceptable match is found, possibly through the selection of a previously inactive representation. This search process is relatively easy to implement, requiring no new nodes or pathways beyond those already present in ART 2 modules. It is also robust, since it does not require tricky timing or calibration.

7. ART 3 Simulations: Mismatch Reset and Input Reset of STM Choices

The computer simulation summarized in Figure 1 illustrates three ART 3 system variables as they evolve through time. The time axis (t) runs from the top to the bottom of the square. A vector pattern, indexed by i or j, is plotted horizontally at each fixed time. Within each square, the value of a variable at each time is represented by the length of a side of a square centered at that point. In each figure, part (a) plots y_j^{c1}, the normalized STM variables at layer 1 of a field F_c. Part (b) plots $\sum_i v_{ij}^{bc}$, the total amount of transmitter released, bottom-up, in paths from all F_b nodes to the jth F_c node. Part (c) plots $\sum_j v_{ji}^{cb}$, the total amount of transmitter released, top-down, in paths from all F_c nodes to the ith F_b node. The ART Search Hypothesis 1 implies that the net bottom-up transmitter pattern in part (b) reflects the STM pattern of F_c in part (a); and that the net top-down transmitter pattern in part (c) reflects the STM pattern of F_b.

In Figure 1, the vigilance parameter is high and fixed at the value $\rho \equiv .98$. For $0 \leq t < .8$, the input is constant. The high vigilance level induces a sequence of mismatch resets, alternating among the category nodes $j = 1, 2$, and 4 (Figure 1a), each of which receives an initial input larger than the input to node $j = 5$. At $t = .215$, the F_c node $j = 5$ is selected by the search process (Figure 1a). It remains active until $t = .8$. Then, the input from F_a is changed to a new pattern. The mismatch between the new STM pattern at F_a and the old reverberating STM pattern at F_b leads to an input reset. The ART Search Hypothesis 2 implies that bound transmitter is inactivated and the STM feedback loops in F_b and F_c are inhibited. The new input pattern immediately activates its category node $j = 1$, despite some previous depletion at that node (Figure 1a).

Large quantities of transmitter are released and bound only after STM resonance is established. In Figure 1b, large quantities of bottom-up transmitter are released at the F_c node $j = 5$ when $.215 < t < .8$, and at node $j = 1$ when $.8 < t < 1$. In Figure 1c, the pattern of top-down bound transmitter reflects the resonating matched STM pattern at F_b due to Input 1 at times $.215 < t < .8$ and due to Input 2 at times $.8 < t < 1$.

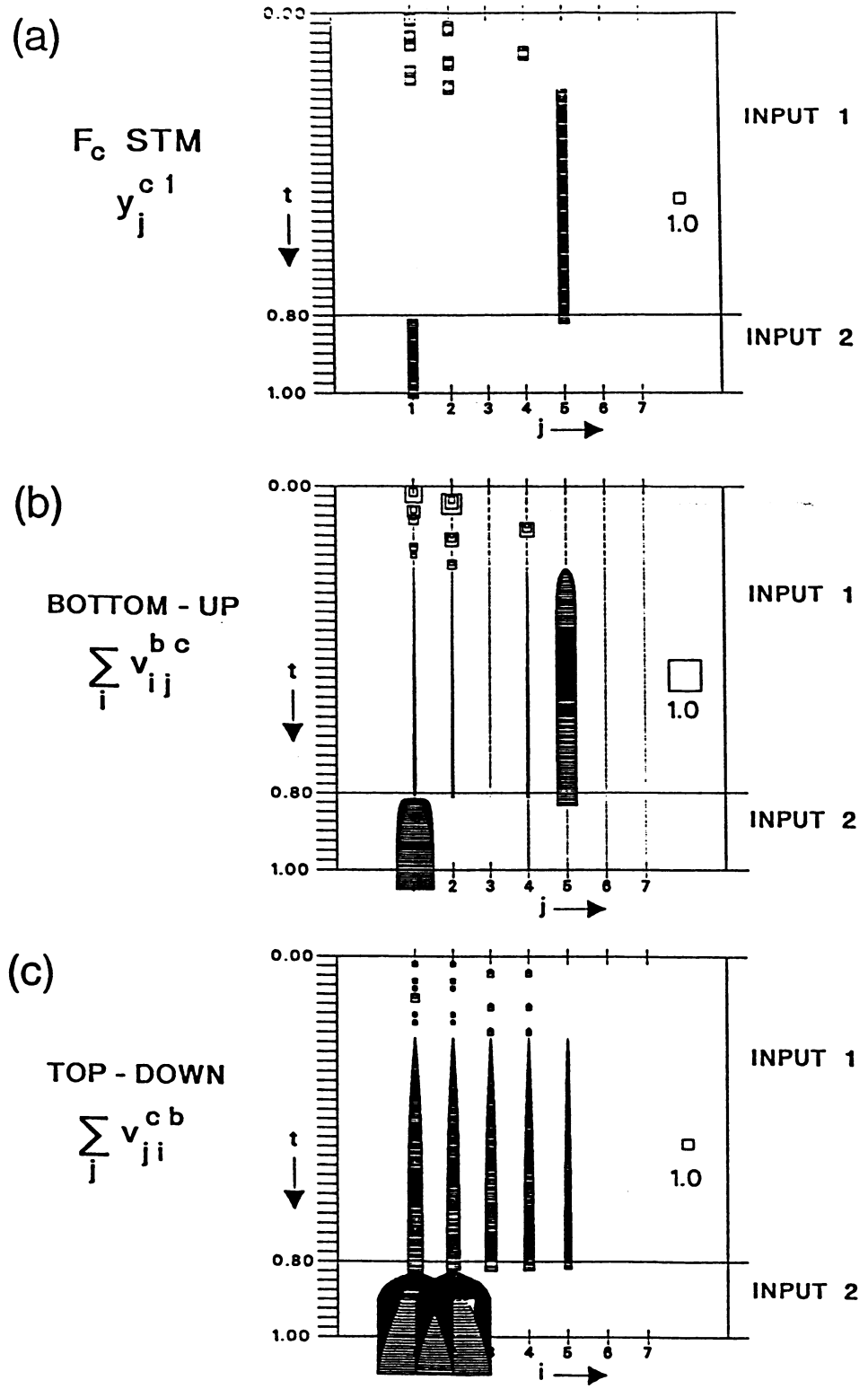


Figure 1. ART 3 simulation with $\rho \equiv .98$. A series of 9 mismatch resets lead to activation of the matched category ($j = 5$) at $t = .215$. Input 1 switches to Input 2 at $t = .8$ causing an input reset and activation of a new category representation ($j = 1$).

SELF-ORGANIZING ANALOG FIELDS (SOAF)

by Fred S. Weingard, Booz·Allen & Hamilton, Inc.
Crystal Square 2, Suite 1100, 1725 Jefferson Davis Hwy.
Arlington VA. 22202-4158

ABSTRACT

In this paper we describe a fundamentally new neural network paradigm called Self-Organizing Analog Fields (SOAF) capable of self-organizing analog-to-analog mappings. Analog-to-analog mappings, in contrast to winner-take-all (WTA) mappings, are essential for developing hierarchical systems capable of extraction and abstraction of spatiotemporal features. SOAF's functionality can be likened to that of an unsupervised backpropagation network (with entirely different system dynamics). We have found that "meaningful" unsupervised analog-to-analog mapping requires the use of modulating (MOD) system dynamics, as well as long-term memory (LTM) and short-term memory (STM) system dynamics. The MOD dynamics are used to drive the system to homeostasis where analog activations (STM dynamics) and learning (LTM dynamics) become "meaningful." Because we have found that unsupervised analog-to-analog mapping requires topological learning, as well as feature-space learning, the synapses in this paradigm contain two LTMs. One LTM regulates topology (e.g., cross-sectional area) and the other, feature-space (e.g., efficacy per cross-sectional area). The criteria for homeostasis allows one to solve, mathematically, the credit assignment problem between self-organizing topology and unsupervised learning of spatiotemporal features. SOAF design is entirely modular and readily accommodates hierarchical organization of SOAFs to build powerful neural network-based subsystems.

Characteristics of Neural Population Codes in Hierarchical, Self-Organizing Vision Machines

Kenneth Johnson
Hughes Aircraft Company, Missile Systems Group
8433 Fallbrook Avenue 262/C64
Canoga Park, CA 91304

This work seeks to understand how the characteristics of the distributed population codes at different levels in the Neocognitron affect the recognition performance of the model. The population codes have characteristics that are controlled by the parameters used to set weight masks at each level. In this paper we analyze these codes and provide interpretations that indicate what constitutes a good population code, and how the codes are related to the features in the input pattern. Analysis of the neural codes is carried out using standard pattern recognition techniques. Results from the studies point to several general principles which are characteristic of hierarchical population coding vision systems.

Measurement of Distributed Population Code Characteristics

The Neocognitron architecture is illustrated in Figure 1. Each layer is comprised of planes of neurons that perform computations using the data on the previous layers as input. The neurons form a massive distributed code that represents characteristics of the input pattern. Associated with each layer are a number of parameters that control the shape and magnitude of convolution weight masks that define the computation. The masks of primary interest are called the ***c*** and ***d*** masks. The location of each of these masks in the hierarchical structure is shown in Figure 1. Details of the computations can be found in the original Neocognitron papers [Fukushima, 1982].

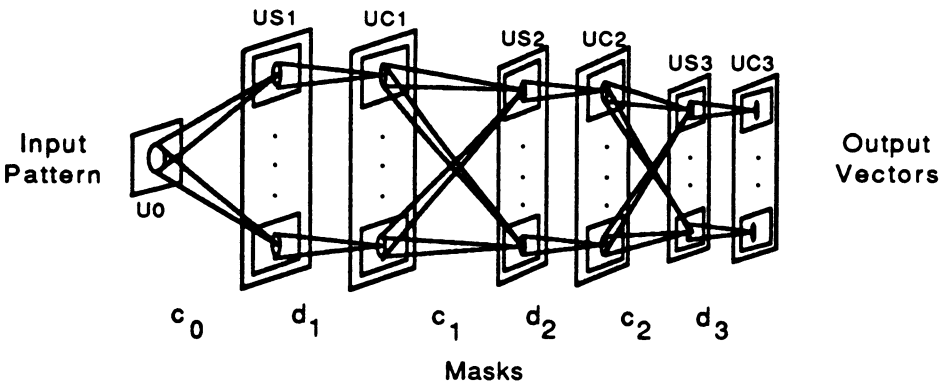


Figure 1: The Neocognitron architecture is hierarchical and massively parallel. Between each level (or representation) is a set of masks that defines the transformation. The location of these masks within the architecture are shown on the bottom.

Figure 2 shows that when the planes on any given layer are stacked on top of each other the neurons at position (x,y) form a code vector representing the input pattern in the neurons' receptive field. When viewing the large population of neurons as sets of code vectors that represent the pattern contained in the input receptive field, one can measure the code vectors' length, orientation in vector space, relationship to other code vectors, and a host of other characteristics using standard pattern recognition concepts. Knowledge of the code characteristics can then be related to the parameters used during the self-organizing process and to the system's classification performance.

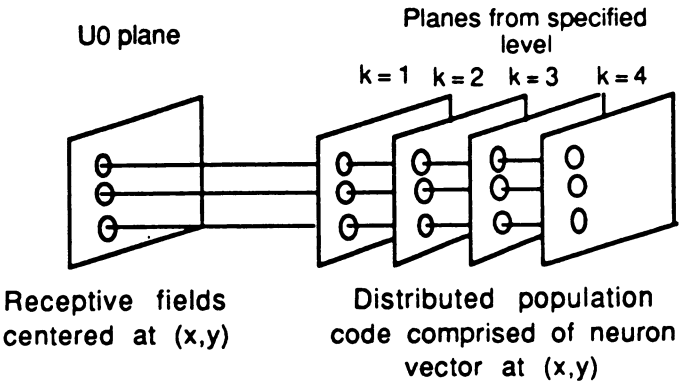


Figure 2: The planes of the Neocognitron can be viewed as sets of code vectors representing the 2D pattern contained in a small receptive field at each (x,y) position in the input.

The Neocognitron simulation used to generate the distributed representations in this paper was written in FORTRAN and run on an Alliant FX4. The input pattern consisted of 99 segmented IR airfield images. The original images were 512x512, the output of the segmentor was 50x50. The simulation was trained using a selected subset of the 99 images and various width masks. The masks were 7x7 pixel, normalized 2D Gaussians with the width specified as the variance of the Gaussian in pixels. The model had 3 sets of US and UC layers. There were 12 planes in layer 1, 24 in layer 2, and 12 in layer 3. The UC3 output planes were 3x3 pixels. The output of the Neocognitron was classified using a backpropagation network. Ten classes were defined according to range from target.

Work presented in previous papers [Johnson et. al., 1988a, 1988b, 1988c] defined optimal mask widths and parameter settings. Using these parameters we achieved 93% classification on the test set. The internal data from each of the US1, UC1, US2, and UC2 planes were stored for each image in the test set. Each image had approximately 120,000 neurons in the population codes of the first 2 layers.

Neural code vectors were analyzed in the following manner. The 7x7 pixel features in the 99 images were sorted and counted. This resulted in 13,219 different features. There were 2718 features that occurred more than 10 times. The (x,y) position of each of the 2718 features was noted in each of the stored population codes. The length and variance of the neural code vectors was computed for each feature type across the samples. Thus, if there were 45 occurrences of a feature in 13 different input patterns, the average length of the code vectors as well as the spherical variance of the code vector cluster in hyperspace was computed. The length and variance were averaged across all codes to generate an average length and variance measure for the entire population code. This experiment was carried out for all combinations of c0, d1, c1, and d2. The data generated for each experiment were compiled into a tree structure shown in Figure 3. Codes at each layer are levels in the tree and each node represents a specific parameter setting.

Code Characteristics

The data contained in the tree has embedded in it many important results which, when analyzed in terms of the models' functional operation and architecture, produce greater insight into the limitations of hierarchical population codes and into the nature of machine perception and recognition.

Of primary interest are the nodes of the tree that are circled. These nodes correspond to what we know to be optimal parameter settings. At these nodes the average code vector lengths are generally larger than any other node at the same level. This larger code length increases the

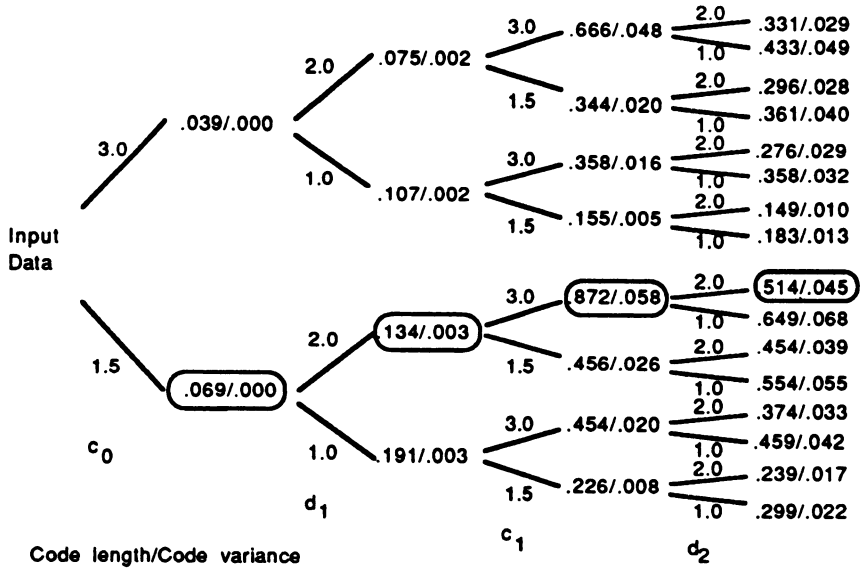


Figure 3: The data in this tree represents the code characteristics for 190,000,000 neurons involved in the coding of 99 airfields. The average code length and variance are specified for each parameter setting at each level. The circles indicate those parameter settings which we know provide the best performance.

volume within which codes can form and therefore increases the number of codes that can be unambiguously defined in the volume. Another observation is that trends in code lengths and variances are related to choices of mask width. For example, the US representation at level 2 has significantly shorter codes with reduced variance if the ***c*** mask is smaller. This is different from UC representations which have shorter codes with reduced variance if the ***d*** masks are larger. Another important point is that when progressing from a US representation to a UC representation the code vectors are generally compressed in length and variance. This is particularly true if the ***c*** mask in the preceeding layer was large. We have correlated performance measures against code lengths and found that those parameter settings that result in shorter code lengths generally result in poor performance.

Principles of Hierarchical Population Coding Machines

One of the purposes of the hierarchical structure of the Neocognitron is to allow the higher levels of the system to have larger receptive fields in the input pattern layer. The larger receptive fields of the higher layers enable the population codes in these higher layers to represent more global characteristics of the input pattern. This is contrasted with the lower level receptive fields which have only narrow views of the world and can therefore represent smaller localized features of the input pattern. Clearly then, if a mask at a higher layer is smaller, the mask is restricting the receptive fields' width. This results in incomplete and ambiguous codes at the higher levels.

A further complication with hierarchical population codes is associated with the dimensionality of the code. As data progresses to the higher layers, the system attempts to squeeze information from thousands of neurons into smaller and smaller populations of neurons. Given that recognition codes are only useful if they can be discriminated, this introduces an incomplete coverage of the code space or large ambiguities between codes. How does one trade code discrimination capability against the desire to squeeze many codes into a given code

volume? Our experience with this issue indicates that this type of architecture functions very well in limited problem domains. However, as pattern complexity and the number of classes increases the codes at the higher levels become very ambiguous. This inability to distinguish between small differences in the codes results in poor recognition performance. From a statistical point of view, this makes sense because feedforward hierarchical systems are essentially projection machines that are limited by the projection capacity of the weights in the system. These observations tend to indicate that a truly versatile vision system may have a different architecture that is free from the capacity limitations imposed by population codes.

Characteristics of a Generic Vision Architecture

The coding limitations imposed by compressive population coding hierarchical architectures can be overcome in several ways. If one assumes that generic object recognition codes are contained at the higher levels, code capacity could be increased by increasing the dimensionality of the codes. However, given the unlimited number of visual patterns in the world, there would never be enough code capacity to unambiguously represent the entire world. A second option would be to use the entire population of neurons at all levels to represent recognition codes for objects in the world. This would allow large scale structural codes to be represented at the higher levels and small scale details to be represented at the lower levels. However, the problem of integrating the multilevel codes into another memory structure that would allow objects to be tagged with the concept of small objects 'contained-within' large objects arises in this scheme. Furthermore, there would still be a finite representation capability due to the fact that neural activity can only represent what the receptive field weights allow them to. This limitation results in a monomodal projection of patterns onto weights. A third option would be to use feedback to adjust the population codes and thereby remove monomodal projection constraints. This option could be implemented in one of two ways. First, recognition codes could be the state of the neural population after feedback had equilibrated. However, the system would still have a limited discrimination and representation capability due to code capacity. A more plausible second alternative would be to use the trajectory of temporal feedback distributed across the entire population of neurons as the recognition code. That is, the object codes are the state transitions of the entire system, not the equilibrium state of the neural population.

The concept of temporal neural codes is not new [Optican and Richmond, 1987; Richmond and Optican, 1987]. From a pattern processing and recognition view a temporal recognition code would have a significantly greater representation bandwidth in the same neural population volume. The representation would not be limited to monomodal projections of input patterns onto the specified population weight spaces of individual receptive fields. Instead, recognition codes and neural processing would be populations of temporal messages which represent the input data and how it was being processed by the system. This is very different from equilibrium population codes which attempt to represent the input data by projecting it onto the weight spaces. This concept indicates that one cannot look at a neural signal and assume that because the neuron is active a specific pattern is contained in the input. Instead, the temporal codes must be viewed as longer messages which are being generated by the system's response to the input patterns. A further characteristic of this type of coding is that the neural codes are not only coding local features present in their receptive fields, but they are coding those features in terms of more global codes being fed back from the higher levels. Therefore, the system is simultaneously segmenting and representing input patterns as the temporal messages evolve in time.

Conclusions

In this paper we present numerical results that quantify the characteristics of neural population codes in the Neocognitron model. We have shown that neural code characteristics are directly related to recognition performance. Furthermore, model performance is directly related to the ability to unambiguously define recognition and representations codes in limited code volumes. Finally, we hypothesize that construction of more powerful vision architectures will require using the temporal aspects of neural responses to increase processing and representation bandwidths.

References

[Fukushima, K.; 1982] Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position, Pattern Recognition, Vol. 15, No. 6, pp. 445-469, 1982.

[Johnson, K., Daniell, C., and Burman, J.; 1988a] Feature extraction in the Neocognitron, 1988 IEEE International Conference on Neural Networks, Vol. 2, pp. 117-127, 1988.

[Johnson, K., Daniell, C., and Burman, J.; 1988b] Hierarchical feature extraction and representation in the Neocognitron, 1988 INNS Conference on Neural Networks, Vol. 1, p. 503, 1988.

[Johnson, K., Daniell, C., and Burman, J.; 1988c] Analysis of the Neocognitron's outputs and classification performance using backpropagation, 1988 INNS Conference on Neural Networks, Vol. 1, p. 504, 1988.

[Optican, L., and Richmond, B.; 1987] Temporal encoding of two-dimensional patterns by single units in primate inferior temporal cortex. III. Information theoretic analysis, J. Neurophysiology, Vol. 57, No. 1, pp. 162-178, 1987.

[Richmond, B., and Optican, L.; 1987] Temporal encoding of two-dimensional patterns by single units in primate inferior temporal cortex. II. Quantification of response waveform, J. Neurophysiology, Vol. 57, No. 1, pp. 147-161, 1987.

Spatiotemporal Pattern Segmentation by Expectation Feedback

Robert Hecht-Nielsen

HNC, Inc.
5501 Oberlin Drive
San Diego, CA 92121
619-546-8877
FAX:619-452-6524
and
Department of Electrical and Computer Engineering
University of California at San Diego
La Jolla, CA 92139

Abstract

The basic mechanism of spatial pattern segmentation for patterns such as images is an operation analogous to cutting out an object of interest in a photograph from its surround with scissors (finding the correct part of the image to cut out is the difficult part). A corresponding basic mechanism for segmenting spatiotemporal patterns (such as the time-varying power spectrum of a sound stream containing multiple overlapping sounds of interest) is not so obvious. This talk discusses a spatiotemporal pattern segmentation mechanism based upon a combination of the expectation feedback model of Grossberg and his colleagues [3,1,4,5] and the activation source backtracking spatial pattern attention mechanism of Fukushima [2]. The talk introduces a neural network architecture for implementing this spatiotemporal segmentation mechanism.

References

[1] Carpenter, Gail A., and Grossberg, Stephen, "ART 2: self-organization of stable category recognition codes for analog input patterns", *Applied Optics*, 26, 4919-4930, 1 December 1987.

[2] Fukushima, Kunihiko, "A Hierarchical Neural Network Model for Selective Attention," in: Eckmiller, Rolf and von der Malsberg, Christoph [Eds.], **Neural Computers**, Springer-Verlag, Berlin, 80-90, 1988.

[3] Grossberg, Stephen [Ed.], **Neural Networks and Natural Intelligence**, MIT Press, Cambridge, 1988.

[4] Grossberg, Stephen, **Studies of Mind and Brain**, Reidel, Boston, 1982.

[5] Grossberg, Stephen, "A Theory of Human Memory: Self-Organization and Performance of Sensory Motor Codes, Maps and Plans," in: Rosen, R. and Snell, S., [Eds] **Progress in Theoretical Biology, Vol. 5**, Academic Press, New York, 233-374, 1978.

Application Systems and Network Implementations

A Transputer Implementation of Toroidal Lattice Architecture for Parallel Neurocomputing

Naoyuki FUKUDA, Yoshiji FUJIMOTO, and Toshio AKABANE
Central Research Laboratories
Corporate Research and Development Group
SHARP Corporation
2613-1, Ichinomoto-cho, Tenri-shi, Nara 632, Japan

ABSTRACT

In a previous paper [1], we proposed a parallel Toroidal Lattice Architecture (TLA) neurocomputer to simulate large scale neural networks. In this paper, we describe implementation of the TLA with Transputers including a parallel processor configuration, load balance algorithm and evaluation of its performance. This TLA neurocomputer has achieved 2 MCPS in a feedforward network using 16 Transputers. Actual proof is given that its performance increases in proportion to the number of parallel processors. We have implemented the Hopfield neural network and applied it to the traveling salesman problem (TSP).

1 Introduction

Neurocomputer applications to speech recognition, image processing, natural language processing, etc. need considerable computing power. Therefore, a parallel neurocomputer which simulates a large scale network is being studied. Virtual implemented neurocomputers, in which a large number of artificial neurons are mapped onto an actual number of physical processors, have been developed. Electrical Virtual Neurocomputers using a parallel architecture such as CM [2], Warp [3], AAP-2 [4] or neuro-turbo [5], etc. are reported. However, these neurocomputers have not effectively solved the connectivity problem for simulation of large scale neural networks thus preventing their expandability. In the previous paper, TLA provided a solution to the connectivity problem between parallel processors.

In this paper, the TLA algorithm is first introduced briefly. Next, the load balancing algorithm crucial for effective parallel processing is indicated. Finally, implementation of a TLA neurocomputer using Transputers T800-20 and evaluation of the performance of the TLA neurocomputer are discussed.

2 Toroidal Lattice Architecture

The TLA virtual processor network as shown in Figure 1 has been derived from a general artificial neuron model. Feedforward processing along the rows and back-propagation learning processing along the columns of a multi layer perceptron are executed efficiently on the TLA processors. A large number of virtual processors (VP) are mapped onto an actual number of physical node processors (NP) by horizontal and vertical partitions. Consequently, the NPs also have a TLA structure. Communication processes between node processors and calculation processes are executed simultaneously, so that the overhead time for communication can be neglected.

The next important matter in efficient parallel processing is load balancing between NPs which is also important for TLA. We solved this problem by permutation of the VP matrix.

3 Load Balancing Algorithm

As shown in Figure 2, when the TLA neurocomputer assigns the VP networks for two neural network models such as the Hopfield network and the multi layer Perceptron, the load of each kind of VP is different. For example, the CP load is about ten times that of the SP with connection. Therefore, if the VP matrix is partitioned simply, load balance would be in adequate. To partition the VP matrix equally, permutations should be done before partitioning. The basic idea of row and column permutation is to distribute to the NPs the rectangular subregions obtained by

dividing a homogeneous or methodically arranged rectangular region. Using this method, each processor has almost the same number of SPs and CPs, that is, the load of each NP is nearly equal. A typical matrix which was partitioned almost uniformly into submatrices is shown in Figure 3.

As shown in Figure 4, the load balance algorithm is given as row and column mapping by permutations and partitions. Even though this algorithm is based on the existence of a homogeneous or methodically arranged subregion in a given matrix, it is applicable to nearly all neural network models.

4 Implementation

The TLA neurocomputer system is implemented by 16 INMOS T800 processors in a 4x4 NP configuration with TLA, a root Transputer, and an IBM-PC host. The 17 processors are assigned as one root processor (RP) and 4 trunk processors (TP), the NPs on the top row, and the 12 other NPs shown in Figure 5. The host computer executes the load balancing algorithm and communicates the network architecture, initial weight matrix, and network active potentials with the RP. It also distributes the submatrices to the NPs via the RP and TPs. The RP is used mainly for communication. The TP has three tasks: computing the weighted accumulation of an active potential and an error, communicating the division of virtual submatrix, partial active potentials and partial errors with neighboring NPs, and up-loading the status of the NPs. Except for up-loading the status of the NPs, the NP plays the same role as a TP.

First, the host computer partitions the VP matrix equally and sends the VP submatrices map to the NPs via the RP. Next, the RP delivers the assigned VP submatrices to each NP via the TPs. Then, each NP starts neurocomputing on its assigned VP submatrix and communicates an active potential with its neighboring NPs. Finally, the computing result is collected and sent to the host computer via the RP and TPs.

We have implemented the TSP with the Hopfield network on the TLA neurocomputer. The network was able to solve for 20 cities. It is important to note that the potential number of simulated artificial neurons and connections are not limited by the number of NPs but only by the total memory capacity. In the current implementation, the simulation rate is approximately 2 million connections per second (MCPS) for the feedforward propagation. Table 1 shows comparative feedforward propagation processing times for several NP configurations. The performance is approximately proportional to the number of NPs. This indicates the TLA architecture can be expanded for large scale neural networks.

5 Conclusion

The practical implementation of a neurocomputer using Toroidal Lattice Architecture has been discussed. The generalized load balance algorithm for TLA is presented. The simulation results showed that the speed is approximately accelerated in proportion to the number of processors.

We are now simulating other neural network models on a TLA neurocomputer and planning a more powerful neurocomputer using DSP chips.

6 References

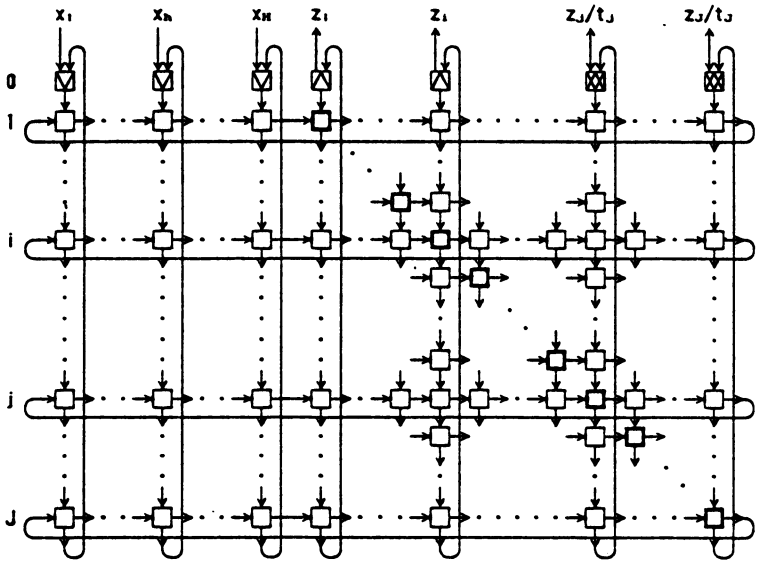
[1] Y. Fujimoto, N. Fukuda, "An Enhanced Parallel Toroidal Lattice Architecture for Large Scale Neural Network", IJCNN, June, 1989.

[2] G. Blelloch, C. Rosenberg, "Network Learning on the Connection Machine", IJCAI, Milano, Italy, 1987.

[3] D. Pomerleau, G. Gusciora, D. Touretzky and H. Kung "Neural Network Simulation at Warp Speed", ICNN, July, 1988.

[4] T. Watanabe, Y. Sugiyama, T. Kondo, Y. Kitayama, "Neural Network Simulation on a Massively Parallel Cellular Array Processor:AAP-2", IJCNN, June, 1989.

[5] A. Iwata, Y. Yoshida, S. Matsuda, et al., "An Artificial Neural Network Accelerator using General Purpose 24 Bits Floating Point Digital Signal Processors", IJCNN, June, 1989.



□ : Synapse Processor
□ : Cell Processor
▽ : Input Processor
△ : Output Processor
⊠ : Input/Output Processor

Figure 1 :
Virtual Processors
with the Toroidal
Lattice Architecture

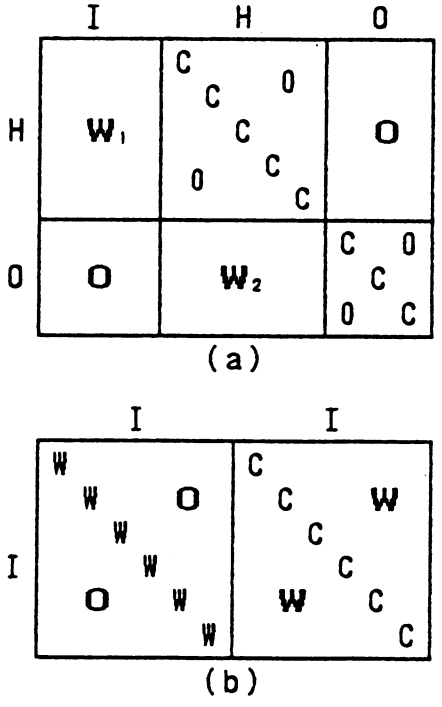


Figure 2 : VP networks for (a) the multi layer Perceptron and (b) the Hopfield network. Zero processor means the synapse processor whose weight is 0.

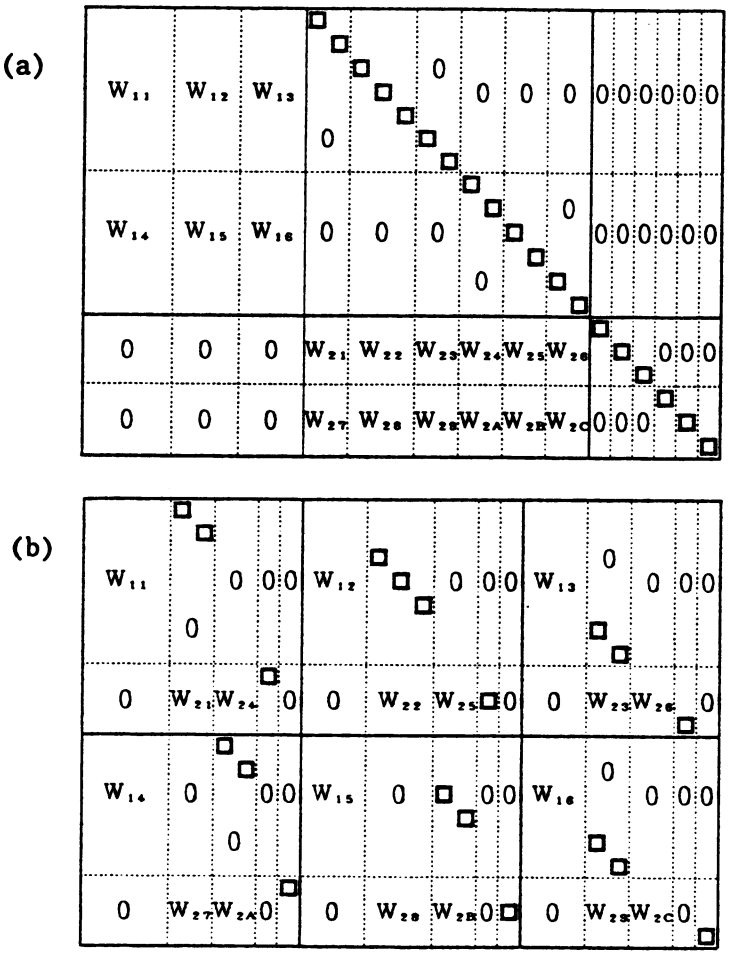


Figure 3 : A typical matrix which was partitioned almost uniformly into submatrices : (a) the subdivision of VPs for permutations. (b) the result of row and column permutations for load balancing.

```
v.sup=0;     h.sup=0
for( h=0; h<b; h++ )
    v[0,h]=L0/Q+1
for( h=b; h<Q; h++ )
    v[0,h]=L0/Q
h.sup=h.sup+b
for( k=1; k≦M; k++ )
    b=mod(Lk,P)
    for( v=v.sup; v<v.sup+b; v++ )
        V[k,v]=Lk/P+1
    for( v=v.sup+b; v<v.sup+P; v++ )
        V[k,v]=Lk/P
    v.sup=mod(v.sup+b, P)
    for( u=0; u<P; u++ )
        c=mod(Vk[v],Q)
        for( h=h.sup; h<h.sup+b; h++ )
            v[k,u*Q+mod(h,Q)]=V[k,u]/Q+1
        for( h=h.sup+b; h<h.sup+Q; h++ )
            v[k,u*Q+mod(h,Q)]=V[k,u]/Q
        h.sup=mod(h.sup+c, Q)
```

(I)Row mapping by permutation and partition

v[k,n] : the number of VPs in the n-th subregion of the k-th layer.

vS[k,n], vE[k,n] : the start and end address of the n-th subregion.

$$vS[k,n]=\sum_{i=1}^{k-1}Li+\sum_{j=0}^{n-1}v[k,j]$$

$$vE[k,n]=vS[k,n]+v[k,n]-1$$

Assume that the y-th VP row is assigned to the y'-th VP row in the p-th NP row.

IF $vS[k,(p-1)Q]\leqq y\leqq vE[k,pQ-1]$, (k=1,2..M)

THEN $y'=y-vS[k,(p-1)Q]+\sum_{i=1}^{k-1}\sum_{m=(p-1)Q}^{pQ-1}v[i,m]$

(II)Column mapping by permutation and partition

Assume that the x-th VP column is assigned to the x'-th VP column in the q-th NP column.

(a)IF $vS[0,q]\leqq x\leqq vE[0,q]$

THEN $x'=x-\sum_{m=1}^{q-1}v[0,m]$

(b)IF $vS[k,q+jQ-1]\leqq x\leqq vE[k,q+jQ-1]$,
(j is one of {0,1,..P-1}, k=1,2,...M)

THEN $x'=x-vS[k,q+jQ-1]+v[0,q]+\sum_{i=1}^{k-1}\sum_{m=0}^{P-1}v[i,q+mQ-1]+\sum_{m=0}^{j-1}v[k,q+mQ-1]$

Figure 4 : Load balance algorithm. Where VP network has M layers, L0 units in the input layer, and Lk units in the k-th layer. NP matrix has P rows and Q columns.

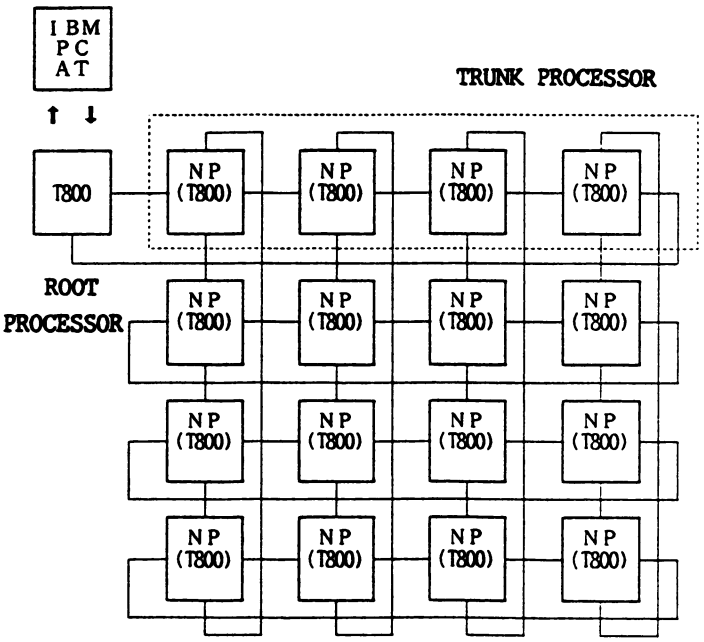


Figure 5 : TLA neurocomputer implementation using 17 Transputers.

Table 1 : Performance in proportion to the number of Transputers.

Number of Transputers (column×row)	Number of Cities kCPS/Performance ratio		
	10	16	20
1	124/0.2	114/0.2	115/0.2
4 (2×2)	505/1.0	522/1.0	522/1.0
8 (4×2)	950/1.9	1023/2.0	1037/2.0
16 (4×4)	1707/3.4	1997/3.8	2042/3.9

A Parallel Neurocomputer Architecture towards Billion Connection Updates Per Second

Hideki Kato, Hideki Yoshizawa, Hiroki Iciki and Kazuo Asakawa
Computer-based Systems Laboratory
FUJITSU LABORATORIES LTD., KAWASAKI
1015 Kamikodanaka, Nakahara-ku, Kawasaki 211, JAPAN

Abstract: *This paper describes a parallel neurocomputer architecture with an error back-propagation learning algorithm. The prototype system consists of 256 digital signal processors and is estimated to run back-propagation at over 500 million connection updates per second. After describing the architecture and algorithm, some improvements and research plans will be mentioned.*

1. Introduction

There have been developed and used many digital electronic neurocomputers or artificial neural network (ANN) simulators including just software ones for general purpose processors or vector processors. These simulators should be very important because they make it easier to use ANNs and help the exploration for the mechanism of the nerve system, where simulation is a virtually effective way to approach because mathematical analysis of large ANNs is very difficult due to their non-linearity. In addition, as research tools, they have certain advantages over optical or analog neurocomputers. These advantages include flexibility and programmability of learning algorithms as well as network configurations, stability and repeatability among several simulations.

Several fast neurocomputers have been reported on with speeds of over a million connection updates per second (MCUPS) such as ANZAplus[1], NeuroTurbo[2], Delta[3], NeuMan[4], CM-1[5], Warp[6] and SX-2[7]. Although the fastest simulator, SX-2 NETtalk, runs at 72 MCUPS, even this is not fast enough for simulating large networks. We have developed a new multiprocessor architecture for digital neurocomputers which is faster and have been developing a 256 processing element (PE) prototype, Sandy/8, as our research tool.

2. Basic Architecture

The architecture developed is shown in Figure 1 and consists of multiple "trays" and PEs. Each tray functions as a container and a router that is connected to its two neighbors. A ring or one dimensional (1-D) torus network is formed which functions as a cyclic shift register. Some of the trays have associated PEs. Each PE has a floating point multiplier, an adder, and some local memory in which weight vectors and program code are stored.

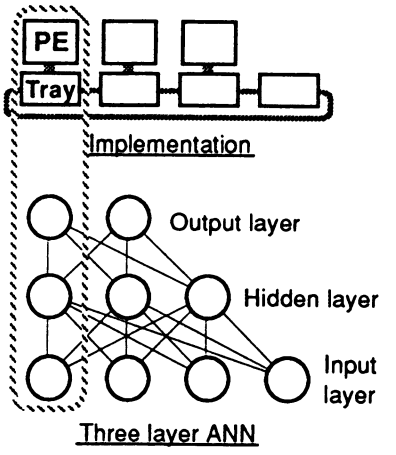


Figure 1. The Basic Architecture

The way to divide and assign tasks to PEs, or to map neurons or synapses to PEs, is a problem commonly found in multiprocessors. Several ideas have been considered on how to solve this for multiprocessor neurocomputers[4; 5; 6]. The mapping developed is shown in Figure 1. There is a three layer perfect connection ANN at the bottom and its implementation is at the top. The ANN has four neurons in the input layer, three in the hidden layer, and two in the output layer. The implementation has four trays, three PEs, and no physical layer structure. The neurons at the same column in all layers are mapped to the same tray and simulated successively by the associated PE (enclosed in dotted lines). It provides us the flexibility for changing the number of layers simulated that this architecture has no physical layer structure. The layers are implemented by software instead. The number of trays required is the maximum number of neurons in all but the output layer. Similarly, the number of PEs required is the maximum number of neurons in all but the input layer.† Note that it is not necessary to assign PEs to the neurons in the input layer.

This architecture can be explained by giving two examples. These are time-consuming operations found in error back-propagation (BP) algorithm[8]. Figure 2 shows a time-chart computing Wx where W is a four by three matrix

† Each PE can simulate any number of neurons by the code in real implementations so that it is possible to simulate large ANNs that have more neurons in a layer than PEs.

that represents the weight vectors of three neurons in the hidden layer and x is a four element vector that represents the outputs of the neurons in the input layer. Similarly, Figure 3 shows a time-chart of W^Ty, where W is the same as above and y is a three element vector of generalized errors, back-propagated from the output layer. The weight matrix W is distributively stored in the local memories connected to the PEs, i.e., a sub-vector of W, $wi*$, is stored in each local memory with skew as shown in the figure.

Each element of x is initially stored in a tray and circulated in the ring during Wx computation. Each partial sum, $\Sigma\ wij\ xj$, is stored in an accumulator on a PE, whose value is initially set to zero as shown at the top of Figure 2. At the next time, T1, PEi fetches and multiplies xi and wii and adds the product to acc$i$. The ring is then rotated counter-clockwise one tray, which is done in the same machine cycle. At T2, PEi fetches and multiplies $xi+1$ and $wii+1$ and adds the product to acc$i$, and so on. Finally, after T4, acc$i$ contains $\Sigma wij\ xj$. This computation takes only four machine cycles.

Computing W^Ty (see Figure 3) is similar except that the roles of trays and accumulators are swapped. Each partial sum is circulated in the ring and yi is held in PEi. The way in which the sub-vectors of W are stored in the local memories is the same as for Wx so that their transfers across PEs are not necessary. This allows for fast successive computation of Wx and W^Ty, which appears in BP.

There is a problem in applying this architecture. Assuming that the simulator has 256 trays and PEs, it requires 256 machine cycles to rotate the ring even if the ANN simulated has much fewer neurons. The time required is independent of the number of active PEs or the number of neurons in a layer simulated. Simulating multilayered ANNs that have different numbers of neurons in different layers cause similar problems. It is thus necessary to have some mechanisms changing the ring size dynamically across layers. This also lets some PEs eventually idle. In other words, the fewer the number of neurons in the output layer the worse the performance, even though the time elapsed does not increase.

This architecture might be seen as a kind of SIMD as each PE executes the same instruction at a time. However, the data-flow for a matrix-vector multiplication is similar to that of 1-D systolic machines[9]. S. Y. Kung at Princeton University independently developed a systolic architecture for neurocomputers, "ring systolic[10]," which is similar to ours for single layer ANNs but different for multilayered ones because the layer structure is physically implemented in his architecture.

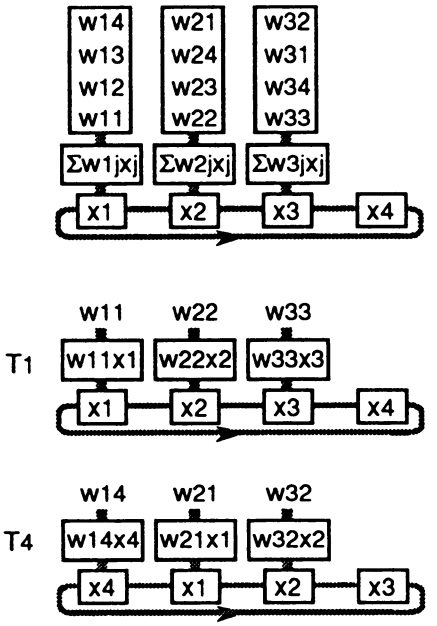


Figure 2. Time-chart computing Wx

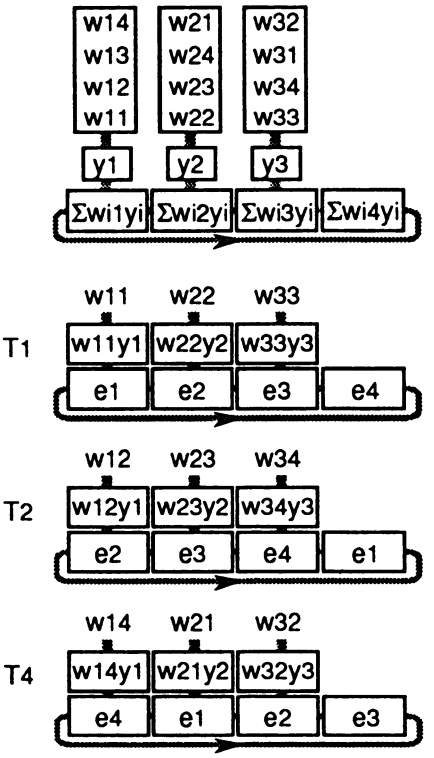


Figure 3. Time-chart computing W^Tx

3. Back-propagation algorithm

The all over time-space chart for the algorithm developed, shown in Figure 4, for three layer ANNs BP learning, which can be extended for almost any number of layers ANNs easily. The horizontal axis shows the space, i.e., PEs used, and the vertical axis shows the estimated time consumed. I denotes the number of neurons in the input layer, H in the hidden layer, and O in the output layer. τ is the time required for a multiply and add operation. The steps expressed at the right half are the operations performed by a PE, PEk, which repeats these steps until some convergence condition is satisfied.

The steps (2) and (3) are the combination of Wx, described above, and the calculation of some sigmoid function that can be done element-wise. Steps (5) through (8) are similar to (2) and (3), and require W^Tx and element-wise multiply-add-store operations in addition. The others are also element-wise operations. All of the steps can thus be

implemented efficiently on this architecture.

4. Extensions

There are two extensions for this architecture; one is for partially connected networks and the other is mainly for 2-D image processing. The basic architecture and algorithm described above are for completely connected ANNs. Partial connections are also efficiently implemented on this architecture. The index of an element of vector x is the same as that of PE in initial state of Wx computation and increases cyclically through the rotation for perfect connection. This is similar to a turntable with some trays on it, such as those commonly found in Chinese restaurants. A shuttle movement instead of full rotation of the ring results in partial connection. The index first increases by half width then decreases by width, which takes one and a half width machine cycles. Note that this requires bidirectional connection.

The network topology is not necessarily one-dimensional but can be extended to an arbitrary number of dimensions. Two dimensional networks are useful for image processing in which perfect connection is not realistic. A spiral movement can be used instead of a shuttle. The required time is thus directly be proportional, rather than proportional to the square, of the number of neurons simulated.

The total time required is roughly $(4H+3I)\tau$ plus twice the time for sigmoid functions.

5. Prototype system and performance

Sandy/8,† a 256 PE prototype system of this architecture, shown in Figure 5, is under development. A floating point digital signal processor (DSP), TMS320C30, is used as the PEs for Sandy/8.‡ Each DSP has 2K words internal RAM and 32K words high speed static RAM external. The word size is 32 bits. These areas can be used as both program and data storage. The machine cycle of the DSP is 60 ns in which multiply and add operation of two 32-bit floating point numbers can be done. The shift operation of the ring can also be done within the cycle, immediately after DSP operations. A custom gate-array of about 2,000 gates is used for the tray including some scratchpad registers. The additions to the basic architecture described above are: (1) a variable size FIFO memory connected to a host machine, Sun-3, (2) short bridges linking over the ring network; (1) and (2) are the mechanisms for changing ring size, and (3) a host common bus through which all of the external local memories associated with the PEs can be accessed by the host. The width of the ring network is 32 bits and its cycle is 60 ns. The band width is thus 67 MB/s, which is much faster than the VME bus connected to the host. The training sample vectors must thus be stored in the FIFO or local

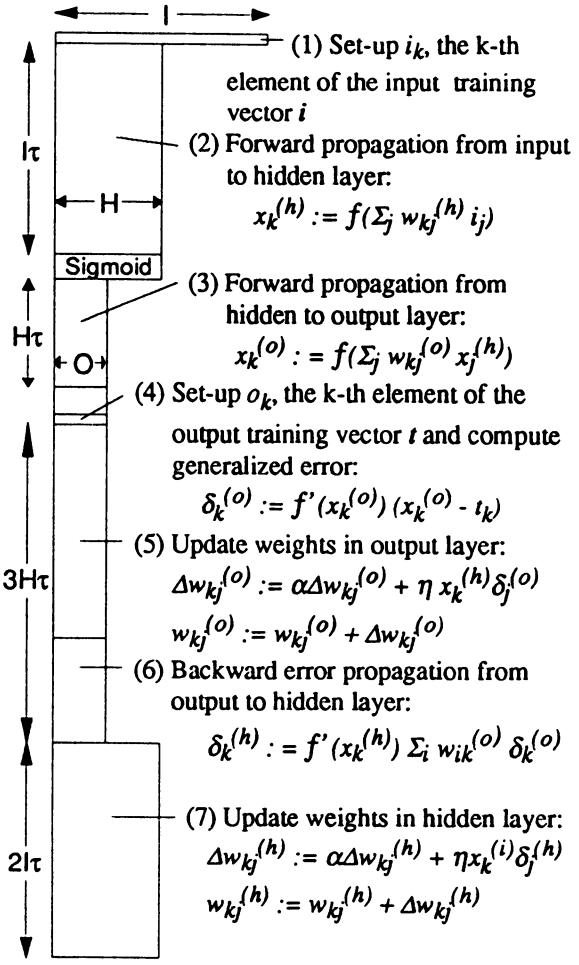


Figure 4. Time-space chart for three layer BP

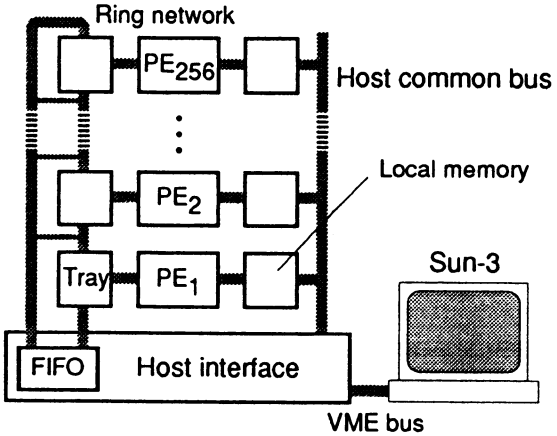


Figure 5. Prototype system

† The number following slash indicates the number of PEs included as in powers of two.

‡ The reason DSPs are used instead of custom LSIs is to speed up development of the prototype. The use of custom LSIs could result in a more compact and faster machine.

memories and used repeatedly in learning cycles to get the best performance.

Table 1 lists the estimated performance for both the 64 PE and 256 PE Sandy and other reported neurocomputers of over one million CUPS. NETtalk[11] which is a famous ANN application widely used for measuring the performance of neurocomputers, was used in the benchmark test. The network has 203 neurons in the input layer, 60 in the hidden layer, and 26 in the output layer. Up to 60 PEs can thus be used in this architecture. This is the reason that the estimated performances of Sandy/6 and of Sandy/8 for NETtalk are the same. However, larger ANNs including more than 256 neurons in a layer show better performance when Sandy/8 is used.

Table 1. Performance of fast neurocomputers

Machine	NETtalk	Max
ANZAplus	-	1.5
NeuroTurbo	-	2
Delta	-	2.7
NeuMan	-	8.5
CM-1	2.8	13
Warp	17	17
Sandy/6	118	141
SX-2	72	180
Sandy/8	118	567

6. Future work

We have some research plans on this architecture.

- Hardware: 2-D version of Sandy with custom VLSI PEs.
- Software: Developmental environment including language and compilers for describing ANNs, and firmware packages for several versions of the learning algorithm.
- Applications: Other than neural ones, conventional image processing and vector processing because this architecture is expected to be effective for these applications.

7. Conclusion

A new multiprocessor architecture for neural network simulators with an error back-propagation learning algorithm has been described. The estimated performance of a 256 PE prototype system, under development, is also reported. The estimation indicates that the prototype runs NETtalk at over 100 and larger tasks at over 500 million connection updates per second for BP learning. Some extensions including dynamic ring size change for flexibility, shuttle movement for one-dimensional partially connected networks, and spiral movement for two-dimensional ones are proposed. These may make it possible to run two-dimensional neural network learning in real time.

Acknowledgement

The authors thank Messrs. J. Tanahashi and H. Hayashi for their encouragement.

References

[1] Hecht-Nielsen, R. (1988). Neurocomputing: picking the human brain. *IEEE SPECTRUM*, **25**(3), 36-41.

[2] Sato, Y., Iwata, A., Suzumura, N., Matsuda, S., & Yoshida, Y. (1989). A Neural Network Accelerator Using General Purpose Floating Point Digital Signal Processors. *IEICE SIG MBE 88-134*, 83-88 (in Japanese).

[3] Works, G. A. (1988). THE CREATION OF DELTA: A NEW CONCEPT IN ANS PROCESSING. *Proceedings of IEEE ICNN88*, **II**, 159-164.

[4] Kajihara, N., Matsushita, S., Nakata, T., & Koike, N. (1988). Parallel Neural Network Simulation Machine: NeuMan. *Abstract of the first annual INNS meeting*, 544.

[5] Blelloch, G. & Rosenberg, C. R. (1987). Network Learning on the Connection Machine. *Proceedings of IJCAI87*, 323-326.

[6] Pomerleau, D. A., Guscoira, G. L., Touretzky, D. S., & Kung, H. T. (1988). Neural Network Simulation at Warp Speed: How We Got 17 Million Connections Per Second. *Proceedings of IEEE ICNN88*, **II**, 143-150.

[7] Asogawa, M., Nishi, N., & Seo, Y. (1988). Network Learning on the Super Computer. *Proceedings of the 36th IPCJ meeting*, 2321-2322 (in Japanese).

[8] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In *Parallel Distributed Processing* (Vol. 1, pp. 318-362). Cambridge, MA: MIT Press.

[9] Kung, H. T. (1982). Why Systolic Architectures? *IEEE Computer*, **15**(1), 37-42.

[10] Kung, S. Y. & Hwang, J. N. (1988). Parallel Architectures for Artificial Neural Nets. *Proceedings of IEEE ICNN88*, **II**, 165-172.

[11] Sejnowski, T. J. & Rosenberg, C. R. (1986). NETtalk: A Parallel Network that Learns to Read Aloud. *JHU/EECS-86/01*, The Johns Hopkins University.

Concurrent ANS Architectures using Communicating Concurrent Processes

Timothy T. Kraft and Stephen A. Frostrom
Science Applications International Corporation
10260 Campus Point Drive
San Diego, California 92121

1. Introduction

From the beginning, Artificial Neural Systems (ANS) have been associated with massively parallel computation[1]. The fundamental element of all neural systems is the processing element which is replicated thousands or millions of times and interconnected with other processing elements to form a concurrent distributed processing network[2]. In practice, the implementation of an ANS network in a multiprocessing architecture involves more than the mapping of processing elements to processors and the interconnection of processing elements. The load leveling of parallel computations, the minimizing of interprocessor communications and the coordination of processor activations must also be considered. For example, synchronous toroidal multiprocessor architectures have been used to implement the Backpropagation Network paradigm with nearly linear increase in performance with number of processors[3 4]. These architectures achieve high performance by exploiting the synchronous nature of gradient decent learning but do not directly map processing elements to processors. Recently Occam has been proposed as an alternative development environment for distributing ANS networks over multiple physical processors such as Transputers[5]. However, the application of communicating sequential processes[6] (CSP) can limit the inherit concurrency of the ANS network.

In this paper, we extend some of the conclusions and explore some of the implications of a concurrent architecture for the implementation the Backpropagation Network proposed in an earlier paper[7]. There we used an actor based language called ANSpec to specify a concurrent architecture based upon the principles of communicating concurrent processes (CCP) . ANSpec uses the actor model[8] developed by Hewitt[9] for the specification of ANS networks consisting of communicating concurrent processing elements called actors. An actor can operate in parallel with other actors to achieve system wide objectives. To prevent the actors or processing elements from competing for processing resources and developing inconsistent views of the system, they must communicate with each other within a distributed processing network[10]. An attractive feature of the actor model is that concurrency is considered the norm and sequential processing a special case such that it is possible to describe ultra-fine concurrent processing models[11]. The actor model also assumes no specific interprocessing protocol other than guaranteed mail delivery. Guaranteed mail delivery ensures that communications sent from one actor to another will be delivered but does not guarantee time or order that the communications will be processed by the receiving actor. This condition is so general that the actor model covers a wide range of physical processing architectures from loosely coupled asynchronous distributed processing to tightly coupled synchronous pipeline processing architectures[12]. Here we discuss the differences between CSP and CCP and their implications in modeling ANS networks and implementing them in multiprocessor VLSI architectures.

2. Backpropagation Network Paradigm

The Backpropagation Network[13] seems to violate the condition of local computation at the processing element during training. In the forward propagation direction, the recall activation function for a processing element is :

$$o_{pj} = \sigma\left(\sum_i w_{jp} o_{pi} + \theta_j\right)$$

For the backward propagation direction during training, the error function is :

$$\delta_{pj} = \begin{cases} (t_{pj} - o_{pj}) o_{pj} (1 - o_{pj}) & \text{output layer} \\ o_{pj} (1 - o_{pj}) \sum_k \delta_{pk} w_{kj} & \text{hidden layer} \end{cases}$$

To implement both the recall activation function and the error function within the local processing element j, the processing element must have access to the connection weights for both the forward (w_{kj}) and backward (w_{ij}) processing elements connected to it. This would imply that the weights are external and 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 learn rate and α is the momentum. The localizing of computations to processing elements is typically a property of ANS networks and a necessary property of concurrent processing networks. The laws of concurrent processing state that there can not be shared access to data among concurrent processing agents.

There is a further problem characteristic of back propagating or feed back networks of both forward and backward propagating signals asynchronously arriving at a processing element. There is the very real potential that an inconsistent implementation will occur if the two types of signals are not coordinated. The processing element performing the error function must match the error (δ_{pj}) with the input activation (o_{pi}) and output (o_{pj}) generated in the earlier forward pass for the pattern p. If the processing element is implemented using communicating sequential processes, the processing element holds these activations as internal state parameters, and blocks any further forward signals until the backpropagated error signal activates the processing element. Once the error and weight changes have been computed the processing element can then change state again to receive the next input signal. This approach will essentially force the network to pass training patterns through the network sequentially. Typically this is not a concern for single processor architectures or tightly coupled synchronous architectures. It does restrict the usefulness of the paradigm in multiprocessor concurrent architectures.

3. Concurrent Implementation of Backpropagation Networks

The concurrent implementation of the Backpropagation Network addresses the computational or algorithmic aspects of the paradigm within a concurrent architecture using communicating concurrent processes. This implementation directly addresses the synchronization problem in the BP network between forward and backpropagating signals and the local computation of weight updates. In the CSP case, the process element has a serialized behavior. That is to say, its state changes with each activation and the forward and backward propagated errors must be synchronized for the processing of the network to be consistent. This is precisely the situation one encounters when attempting to use sequential process techniques such as channels in Occam to implement ANS networks.

The implementation for the concurrent version of the Backpropagation Network requires that the processing elements be unserialized throughout the processing of a set of training patterns. At the end of a training cycle, the weights for each processing element are updated. The synchronization problem arises because the processing elements are serialized during the forward propagation with its internal state defined by the input and output activations. This situation can be removed if another actor is created to receive the back propagated signal and has an initial state given by the input activation and output signal (figure 1). This

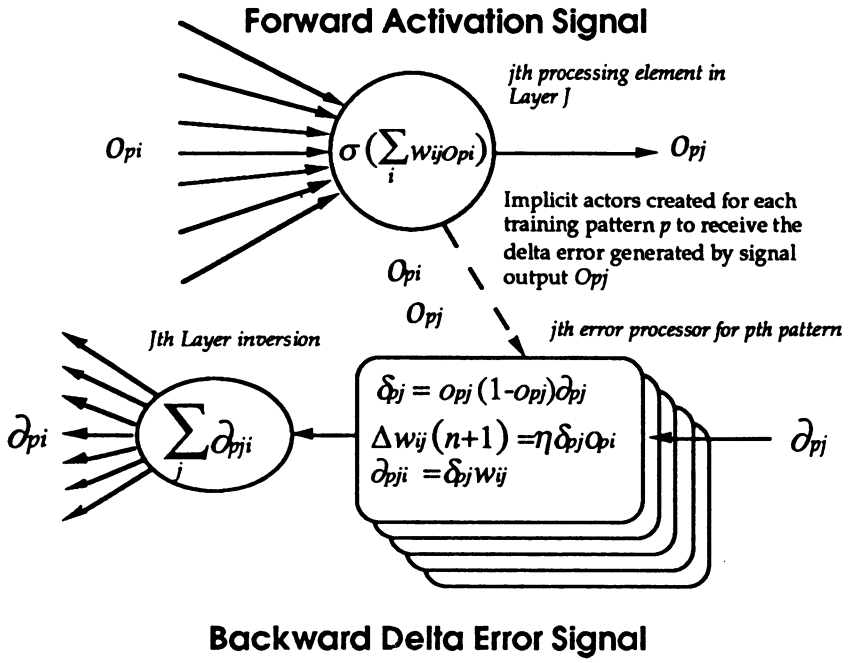


Figure 1 Use of implicit actors created during forward propagation to receive the delta error during backpropagation to resolve forward and backward signal propagation conflicts at the processing element. This will create an implicit actor for each training pattern pair during training. The delta weights are collected but not applied until the end of the training cycle.

uncouples the forward and backward propagation and allows both signals to propagate independently from each other. This new actor handles the error processing for each fire activation and is the customer of the error signal for the particular training pattern p. Meanwhile the original processing element is free to process the next training pattern.

The creation of new actors during processing frequently occurs in actor models and is analogous to spawning parallel processes. Often when implementing processes within a behavior, actors are created implicitly to provide the appropriate sequence of actions. The implicit actor created here inherits the weights and activations of the parent processing element and receives the delta error replies from processes the parent initiated . This eliminates the need for the processing element to save its forward activation state making it an unserialized process capable of immediately processing the next input activation. The error process actor will be the customer of the delta error signal from the forward layer. The error process actor is created implicitly and disposed implicitly after completing the error function process.

To localize the error function, the error process actor pj computes a vector of delta errors (δ_{pji}) using the weights local to the parent processing element (w_{ji}). The layer performs the inversion to compute the delta error (δ_{pi}) to be distributed to the processing elements of the next backward layer. The layer is an unserialized actor which essentially forwards or propagates signals among the processing elements. In the actor model unserialized forwarder actors become part of the guaranteed mail delivery system. The resulting network with forwarder layer behaviors is isomorphic to the direct connection of the processing elements themselves. For the computation of the weight changes, the order in which the weight changes are accumulated during a training cycle is not important and can be performed asynchronously with the input processing. This means that the processing of input signals and the accumulation of weight changes at a processing element can proceed concurrently.

In Kraft et. al[6] this strategy is implemented in detail except that the new actor is created at the layer level. An implicit layer actor is created each time the layer receives a forward activation. The implicit actor sends

the appropriate activations along with the delta error to the process elements during error processing. To ensure that the layer actor is always active, the layer behavior is unserialized. This allows for a potential static implementation of the layer behavior while implicit actors are created dynamically as required. Therefore, all intermediate error computations are managed by the processing elements to eliminate the need to save these as part of the layer task processing. This division of labor among the layer, process elements, and implicit actors will allow for a concurrent implementation of the network while at the same time allow for a more efficient static implementation of the original architecture.

4. Implications of the Concurrent Architecture.

This architecture was derived from the direct application of communicating concurrent processing to the computation of the Backpropagation paradigm. It attempts to maximize the concurrency of the given paradigm by removing sequential processing requirements. In this case the forward and backward signal propagation processes are uncoupled and can proceed concurrently over a training set to implement optimal updating of the weights. This was accomplished because the Actor Model which implements CCP allows for the spawning of parallel processes. The implication is that processors which would have remained idle are free to perform computations increasing the overall throughput of the system. Further the architecture is sufficiently general to support a number of implementation approaches from loosely coupled processing networks to tightly coupled MIMD processors implemented in VLSI chips.

1J.L. McClelland, D.E. Rumelhart, and G.E. Hinton. "The Appeal of Parallel Distributed Processing", Parallel Distributed Processing, Vol. 1, pp3-44, MIT Press 1986.

2D.E. Rumelhart, G.E. Hinton, and J.L. McClelland. "A General Framework for Parallel Distributed Processing", *Parallel Distributed Processing*, Vol. 1, pp45-76, MIT Press 1986.

3Fujimoto, Y. and Fukuda, N. An Enhanced Parallel Toroidal Lattice Architecture for Large Scale Neural Networks, IJCNN 89, also Fukuda, N, Fujimoto, Y, and Akabane, T. A Tranputer Implementation of Toroidal Lattice Architecture for Parallel Neurocomputing, IJCNN 90.

4Kato, H., Yoshizawa, H., Iciki, H. and Asakawa, K. A Parallel Neurocomputer Architecture towards Billion Connection Updates per Second, IJCNN 90.

5P. Koikkalain and E.Oja. "Specification and Implementation Environments for Neural Networks using Communicating Sequential Processes", Proceedings of the 2nd International Conference on Neural Networks, Vol. 1, pp533-540, 1988.

6Hoare, C.A.R. Communicating Sequential Processes, Communications of the ACM Vol 21 No. 8 (August 1978), pp. 666-677.

7Kraft, T., Frostrom, S., MacRitchie, B., and Rogers, A. The Specification of a Concurrent Backpropagation Network using Actors. NeuroNiemes 89.

8G. Agha. *Actors, A Model of Concurrent Computation in Distributed Systems*, The MIT Press 1986.

9C. Hewitt, "Concurrency in Intelligent Systems", AI Expert, Premier 1986.

10C. Hewitt, "Viewing Control Structures of Patterns of Passing Messages", Journal of Artificial Intelligence, pp323-365, June 1977.

11G. Agha and C. Hewitt, Actors: A Conceptual Foundation for Concurrent Object-Oriented Programming", Research Directions in Object-Oriented Programming, ed. Shiver and Wegner, pp49-74, MIT Press 1988.

12T.Kraft and S. Deiss, "ANSpec : A Specification Language for Massively Parallel Distributed Systems", unpublished.

13D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation", *Parallel Distributed Processing*, Vol. 1, pp318-364, MIT Press 1986.

FUZZY KNOWLEDGE MODEL OF NEURAL NETWORK TYPE
— A model which can be refined by learning —

Atsushi MORITA*, Yoshihito IMAI*, Akio NODA*, and Morikazu TAKEGAKI**

* Industrial Electronics & Systems Development Lab., Mitsubishi Electric Corp.
** Central Research Lab., Mitsubishi Electric Corp.
8-1-1 Tsukaguchi-honmachi, Amagasaki, Hyogo 661, Japan

1. INTRODUCTION

Inaccuracy in the knowledge model often degrades the performance of an expert system, but it is not easy to extract accurate knowledge from a human operator. One of the solutions is to refine a model by learning, and a few learning algorithms for a fuzzy model have been proposed [1,2]. They change an original fuzzy model by shifting membership functions, or by changing expression in THEN clauses so that the error included in the model is decreased. But these methods do not cover the problem when IF clauses are not properly described.

This paper proposes a fuzzy knowledge model, where each fuzzy rule has a weight which changes the importance of the corresponding rule. In the learning process each weight is changed to reduce the error included in the model. As a result those rules which are not quite correct are excluded from the model. Since the proposed model is already close to a skilled operator at the beginning, learning speed is faster than that by a neural network, where it starts from almost nothing.

2. STRUCTURE OF THE PROPOSED FUZZY KNOWLEDGE MODEL

Figure 1 shows a block diagram of the proposed fuzzy knowledge model of a neural network type. The model describes the knowledge of a skilled operator using fuzzy rules, and infers an output value from input values. It is different from a conventional fuzzy knowledge model in that the output of each fuzzy rule is multiplied by a weighting factor, ω_i. By varying the weighting factors the input-output characteristics of the model is changed. In the learning process the weighting factors are changed to decrease the error included in the model.

Since the weights of the proposed model correspond to synaptic weights of a neural network, we call the proposed model a "fuzzy knowledge model of a neural network type".

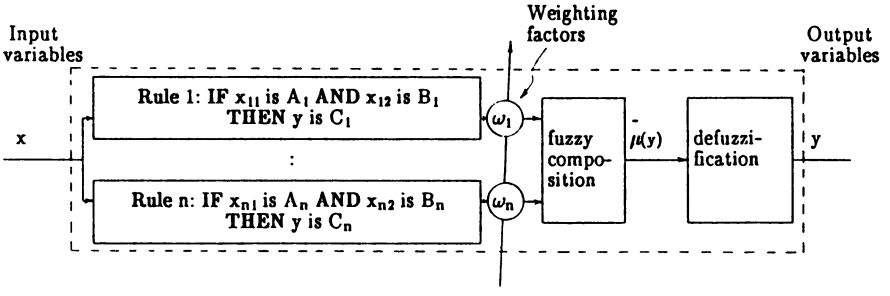


Figure 1 Structure of the proposed fuzzy knowledge model

The output value of the model, $\bar{y}$, is calculated in a similar manner as a conventional fuzzy knowledge model using Equations (1) and (2).

$$\tilde{\mu}(y) = \max_i [\omega_i * \mu_{C_i}(y) * \min \{\mu_{A_i}(x_{1i}), \mu_{B_i}(x_{2i})\}] \quad \cdots (1)$$

$$\bar{y} = \frac{\int \tilde{\mu}(y)\, y\, dy}{\int \tilde{\mu}(y)\, dy} \quad \cdots (2)$$

where $\mu_{A_i}(\cdot)$, $\mu_{B_i}(\cdot)$ and $\mu_{C_i}(\cdot)$ denote membership functions for fuzzy sets A_i, B_i, and C_i, respectively.

3. LEARNING PROCEDURE

In the first stage a knowledge base is created by describing skilled operator's know-how in the form of fuzzy rules and membership functions. This procedure is the same as a conventional one, which uses this knowledge base without modification. In the proposed model the model is refined by modifying the weighting factors so that the difference between the output value of a nominal model and that of the proposed model becomes smaller than that in the previous iteration.

In the following three learning methods are shown, one of which models a skilled operator directly, and the others make the proposed model approach an inverse model of a plant.

3.1 The model of an operator

The first method uses a skilled operator as a nominal model in the learning procedure. Figure 2 shows a block diagram of the system. With the proposed fuzzy model it is refined by adjusting the weighting factors so that the difference between the output value of the proposed model and that of a skilled operator is decreased.

Let $J(k)$ be a performance index of an error in k^{th} iteration as shown in Equation (3). The weighting factors, $\omega_i(k)$, is modified in the next iteration so that the performance index of an error in $(k+1)^{th}$ iteration, $J(k+1)$, is slightly less than $J(k)$. Then the proposed model is adjusted to approach the skilled operator as the number of iterations increases.

$$J(k) = \frac{1}{2} \|y(k) - y_d(k)\|^2 \quad \cdots (3)$$

$$\Delta\omega_i(k) = -\varepsilon_i \frac{\partial J(k)}{\partial \omega_i(k)} = -\varepsilon_i\, y(k) \frac{\partial y(k)}{\partial \omega_i(k)} \quad \cdots (4)$$

$$\omega_i(k+1) = \omega_i(k) + \Delta\omega_i(k) \quad \cdots (5)$$

where ε_i is a positive small number.

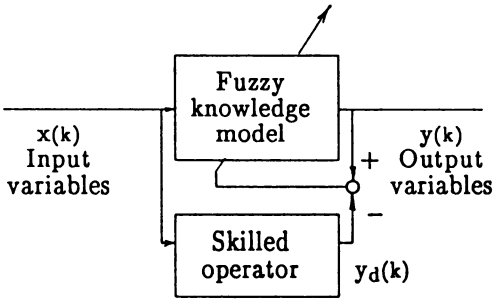


Figure 2 The model of an operator

3.2 Inverse model of a plant

The goal of the previous method is to model operations done by a skilled operator, who does not guarantee the best operation. A better method is to build an inverse model of a plant. Figure 3 shows a configuration of this method, where the output of the plant is compared with the command. The weighting factors in the fuzzy knowledge model are modified so that the difference between the output of the plant and the command is decreased.

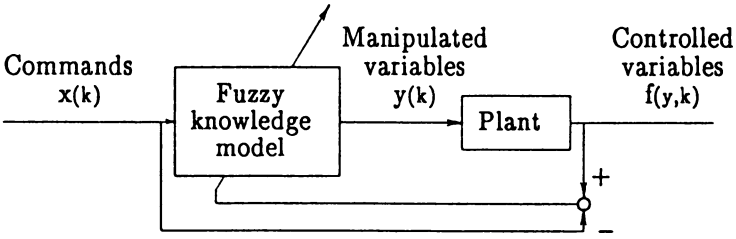


Figure 3 Inverse model of a plant

Let $J(k)$ be a performance index of an error in k^{th} iteration as shown in Equation (6). The weighting factors, $\omega_i(k)$, are modified after each iteration as follows:

$$J(k) = \frac{1}{2} \|f(y,k) - x(k)\|^2 \quad \cdots (6)$$

$$\Delta\omega_i(k) = -\varepsilon_i \frac{\partial J(k)}{\partial \omega_i(k)} = -\varepsilon_i\, f(y,k) \frac{\partial f(y,k)}{\partial y(k)} \frac{\partial y(k)}{\partial \omega_i(k)} \quad \cdots (7)$$

$$\omega_i(k+1) = \omega_i(k) + \Delta\omega_i(k) \quad \cdots (8)$$

Then the proposed model is modified to approach an inverse model of the plant as the number of iterations increases.

3.2 Another method to build an inverse model of a plant

To calculate Equation (7) partial derivative of f(y,k) with respect to y(k), (a sign of it at least,) is required. But it is not always available, especially when the characteristics of the plant is not monotonous. To overcome this problem learning steps are divided into a coarse and fine steps which are used in CMAC [3]. In the coarse step the model is roughly tuned in the configuration shown in Figure 3. Then the difference is compared not at the output of the plant but at the output of the model in the fine learning step, which is described by exchanging the plant with the proposed model as the one shown in Figure 4. The weighting factors, $\omega_i(k)$, are modified using Equations (3)-(5), where partial derivative of f(y,k) with respect to y(k) is not required.

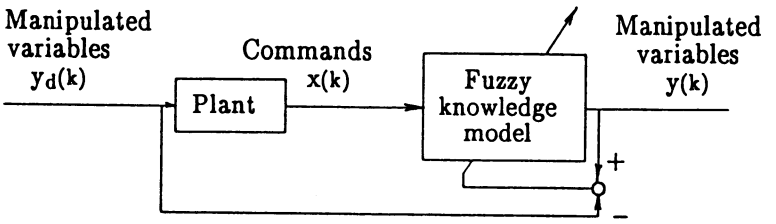


Figure 4 Another method to build an inverse model of a plant

4. SIMULATION RESULTS

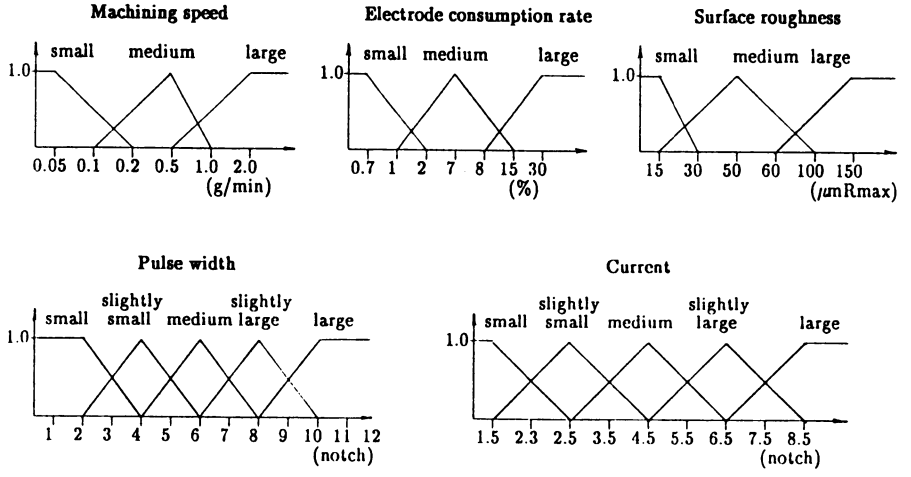
To verify the validity of the proposed fuzzy model, simulations are performed. The control used in the simulation is machining condition setting for an electric discharge machine, and the learning procedure used in the simulation is the one described in 3.3. The commands are machining speed, electrode consumption rate and surface roughness, and the manipulated variables are current and pulse width. The characteristics of the machine is given in [5].

The fuzzy rules and their membership functions are shown in Figures 5 and 6, respectively. There are 17 fuzzy rules and their corresponding membership functions.

Rule #	IF clause			THEN clause	
	Machining Speed	Electrode Consumption	Surface Roughness	Pulse Width	Current
1	Large	Large		Medium	Large
2	Large	Medium		Large	Large
3	Large	Small		Large	Large
4	Large		Large	Slightly Large	Large
5	Large		Medium	Slightly Small	Large
6	Medium	Small		Large	Slightly Large
7	Medium		Small	Small	Medium
8	Small	Small		Medium	Small
9	Small		Small	Slightly Small	Small
10		Large	Large	Medium	Large
11		Large	Medium	Medium	Medium
12		Large	Small	Small	Slightly Small
13		Medium	Large	Slightly Large	Large
14		Medium	Medium	Medium	Slightly Small
15		Medium	Small	Small	Small
16		Small	Large	Large	Large
17		Small	Medium	Slightly Large	Medium

Figure 5 Fuzzy rules used in the simulation

Figure 6 Membership functions used in the simulation



The results of the simulation for the pulse width are shown in Figures 7 and 9, those for the current are shown in Figures 8 and 10. The vertical axes in Figures 7 and 8 show the performance index given in Equation (3), and those in Figures 9 and 10 show the weighting factors given in Equation (5). The horizontal axes show a number of iterations.

In the simulation the weighting factors at the beginning, $\omega_i(0)$, are set to 0.5, and the input values to the machine are randomly changed. It can be seen that the errors included in the model are decreased as the number of iterations is increased. Figures 9 and 10 show that the weighting factors are adjusted according to the importance of the rules.

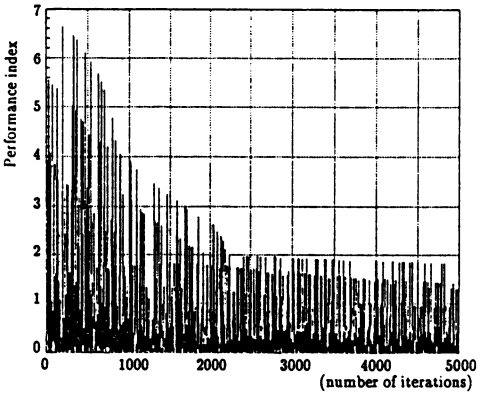


Figure 7 Change in performance index (pulse width)

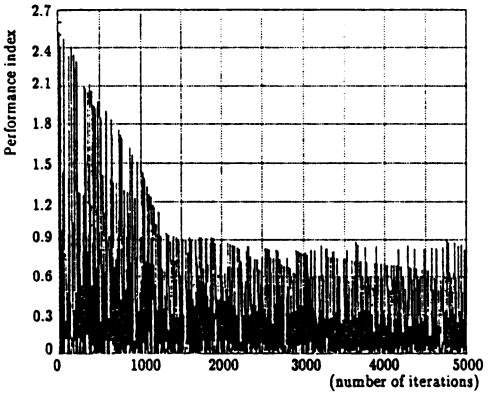


Figure 8 Change in performance index (current)

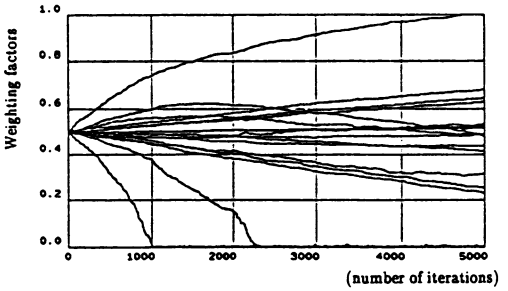


Figure 9 Change in weighting factors (pulse width)

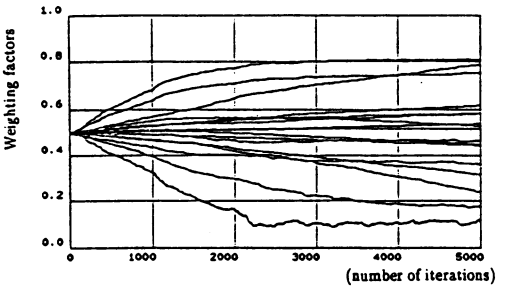


Figure 10 Change in weighting factors (current)

5. CONCLUSION

In this paper we presented a fuzzy knowledge model of a neural network type. The output membership function of each fuzzy rule in the model is weighted by a factor, which is adjusted to minimize the error included in the model. Three learning methods are presented. One of them models a skilled operator directly, and the others make the proposed model approach an inverse model of a plant. Simulation results for an electric discharge machine show that the error decreases as the number of learning iterations is increased.

References

[1] T.Yamazaki and M.Sugeno: Self-Organizing Fuzzy Controller, Trans. of SICE of Japan, Vol.20, No.8, 720/726 (1984, in Japanese)

[2] M.Maeda and S.Murakami: Self-Tuning Fuzzy controller, Trans. of SICE of Japan, Vol.24, No.2, 191/197 (1988, in Japanese)

[3] L.A.Zadeh: Fuzzy Sets, 338/353, Information and Control, Vol.8 (1965).

[4] J.L.Albus: Data Storage in Cerebellar Model Articulation Controller (CMAC), Trans. of ASME, J. of Dynamic Systems, Measurement, and Control, 228/233 (1975)

[5] NC-EDM Systems K Series, Machining Characteristics, Data Book 1 & 2, Mitsubishi Electric Corporation (1988)

ARCHITECTURE OF A SYSTOLIC NEURO-EMULATOR

U. Ramacher, J. Beichter
SIEMENS AG, R&D CENTRE MUNICH, W. GERMANY

ABSTRACT

The response and the characteristics of present models of artificial neural nets are primarily investigated by simulation on vector computers, workstations, special coprocessors or transputer arrays. The fundamental drawback of such simulators is that the spatio-temporal parallelism in the processing of information that is inherent to the neural net is lost entirely or partly and that the computing time of the simulated net especially for large associations of neurons (tailored to application-relevant tasks) grows to such orders of magnitude that a speedy acquisition of "neural" know-how is hindered or made impossible. This paper discusses systolic architectures that support emulation of neural networks composed of MLP-modules, and presents the layout of a chip specially designed to accelerate the weighting.

1. INTRODUCTION

An appreciable reduction in computing time for the simulation of neural nets and thus the handling of largish tasks or those that are to be executed in realtime become possible with neural hardware that contains an artificial neural net of finite size. Apart from the shortest possible computing time, neural hardware offers a very much smaller structural volume than can be implemented with hardware simulators for the same task. This aspect is especially important when neural hardware is to be incorporated in terminals for man/machine communication or mobile robotics.

In the development of digital neurocomputers it is consequently a matter of implementing the compute-bound learning algorithms in hardware and designing a system and circuit architecture that supports in optimal fashion the massive parallel networking of the neural net and produces a sufficient measure of flexibility and expansion capacity for coping with a domain of applications (e.g. vision, speech, signal processing, robotics) or with general-purpose action.
Emulation pursues a strategy of restricting the massive parallelism of a neural net as little as possible. The basic idea is to generate a large net by means of a systolic array of small (chip-integrable) nets. An implementation of this kind has the spatio-temporal parallelism for the modules of the array and still allows the spatially parallel, time-sequential processing of the neural input. Systolic emulation with small nets seems to be the best possible approximation to neural parallelism for rapid execution of the learning and recall phase respectively of large neural nets.

2. NEURAL SIGNAL PROCESSING

A neuron's elementary recall function is of simple algorithmic structure:

$$y_i = f(z_i) = f\left(\sum_{k=0}^{N} (W_{ij} a_j + w_i b_i + \theta_i) \cdot \lambda \right) \qquad , i = 1 \ldots M \qquad (1)$$

N is the number of inputs common to a layer of M neurons. The first summand represents the weighting of the common inputs a_j, the second one the weighting of the neuron's individual input b_i, and the last one a threshold value θ_i. f is the discriminator function (step, ramp, sigmoidal, tanh or a table) and y_i the output of the i-th neuron. The parameter λ controls the slope of the discriminator function or the temperature in case that a Boltzmann net is to be generated. Formula 1 displays an elementary neural processing step which is common to all known neural paradigmas [1,2].

For large neural nets with several thousands of neurons, 1-chip implementation becomes impossible, even with a future 0.3 µm CMOS technology [3] at wafer level. In order to preserve as much of neural parallelity as possible, we suggest to decompose formula (1) in the following way:

$$y_i = f\left(\left| \sum_{k=0}^{K-1} \sum_{j=kn+1}^{(k+1)n} W_{ij} a_j \right| + w_i b_i + \theta_i \cdot \lambda \right) \qquad , i = 1 \,..\, L \cdot \; m = M \; . \qquad (2)$$

n counts the number of parallel multipliers of a neuron, m the number of neurons to be implemented at a chip. It follows immediately that the whole weighting takes N/n·M/m steps. Therefore, the number n·m of synapses to be implemented in parallel on a chip should be chosen as high as possible. Figure 1 shows a 4-neuron unit with 16 synapses.

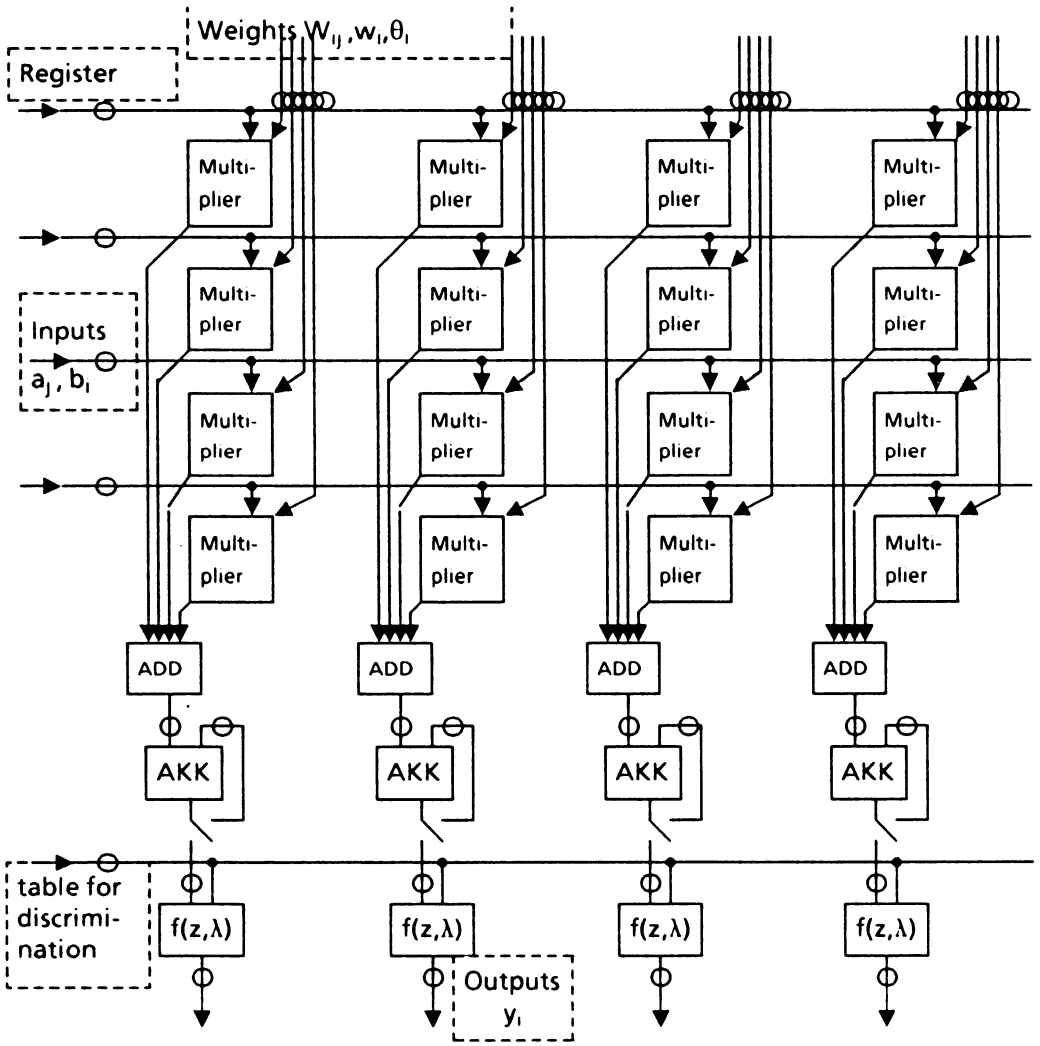


Figure 1 Schema of a 4-neuron unit with 16 multipliers.

It becomes clear from figure 1 that the number of pads needed to transfer the weights to the chip determines the number of neurons to be implemented. If 8 bit weights were to be used, a total of 128 pads were necessary for parallel transfer of the weights to the chip. Apparently, it makes no sense for the fast emulation of large nets to implement more neurons on a chip than can be supplied with parallel weights. The use of pad demultiplexers will make little change to this either. Chip area which is not used by multipliers can be used to advantage for implementing those parts of the circuitry that support universal emulation (for example, block floating point format). See, however, [4].

A second constraint which influences the chip architecture is posed by the clock cycle. If the weights are to be stored in a DRAM bank, reading the memory in the ripple mode results in ca. 30 ns clock cycle. Thus, trimming the multiplier array to highest throughput is not an urgent requirement and word-level multipliers and accumulators suffice.

The m-neuron unit may be embedded into a systolic chain (figure 2). A chain of L m-neuron units makes multiple use of the input data $a_1..a_N$. The input vectors need to be repeated M/(Lm)-times

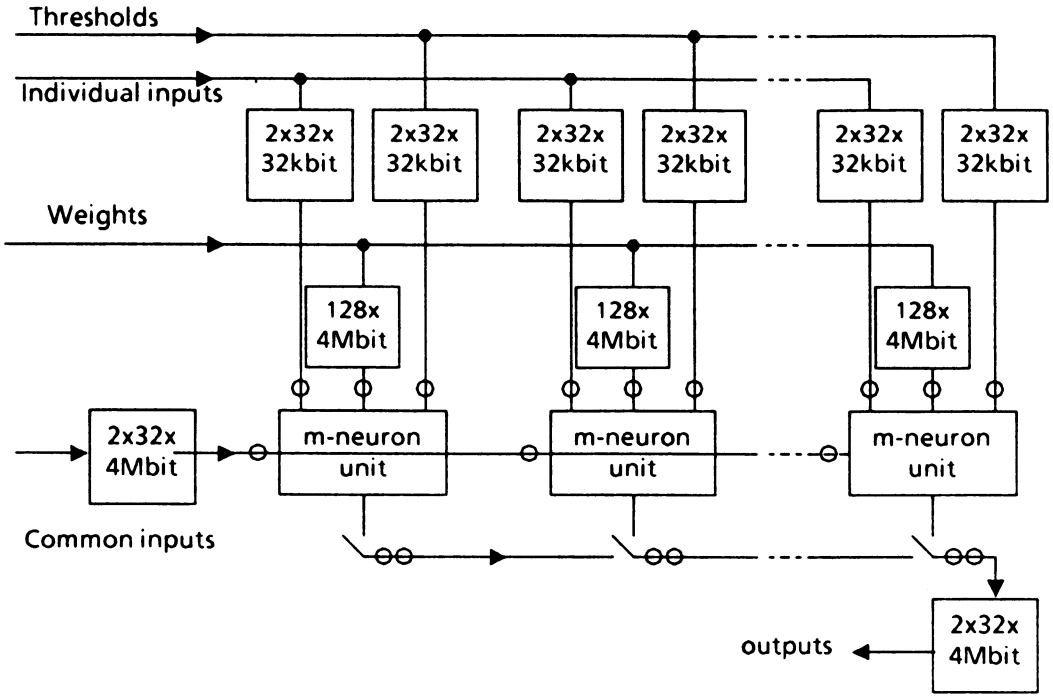


Figure 2 Systolic neuro-emulator with peripheral memory banks.

only, i.e the systolic chain computes L-times faster than the simple m-neuron unit.

The computation time is given by the expression

$$Computation\ time = \left| \frac{N \cdot M}{m^2} \cdot \frac{1}{L} + 2L - 1 \right| \cdot \Delta \quad , \quad \Delta = clock\ period$$

With Δ = 50 ns, L = 10 and m = 4, a number $6.4 \cdot 10^8$ of connections may be computed in 0.2 s. This size of local memory is sufficient to run a Hopfield net of 25000 neurons. Larger nets, however, require additional time to be added for loading new weights into the local memory (a DRAM bank of $6.4 \cdot 8 \cdot 10^8$ bit would take approximately 160 s to be rewritten by means of a 8bit bus, with a transfer rate of 400kB/s).

Figure 3 displays a systolic chain for multiple use of weights. For instance, a set of pictures could be scanned by the same set of weights.

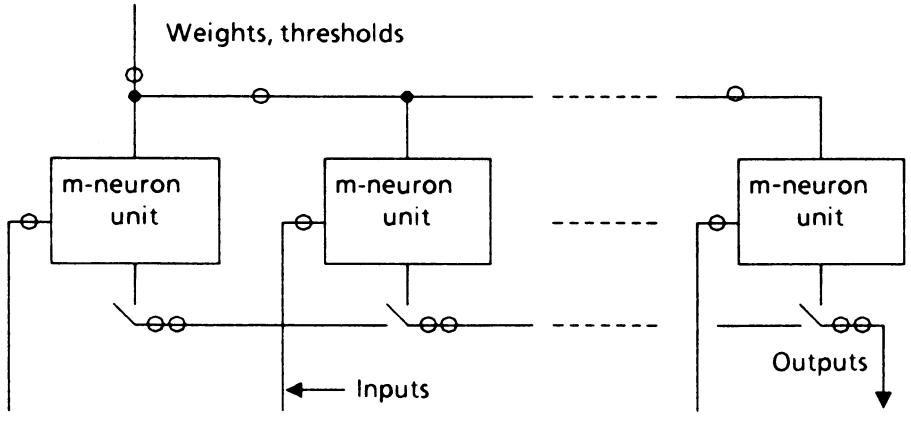


Figure 3 Systolic emulator array with independent input ports.

Figure 4 shows the layout of a WSI chip with 512 data pads which can perform the simultaneous

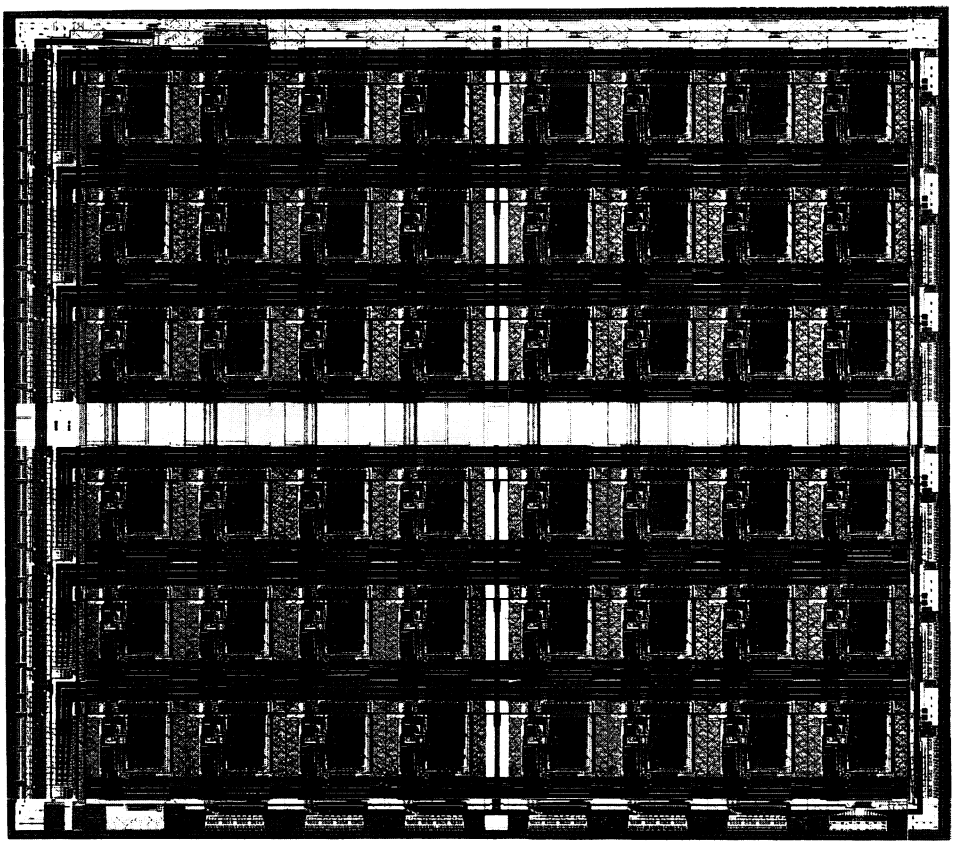


Figure 4
Layout of a systolic array of multipliers & accumulators (chip size = 6x6.8cm^2) .

weighting of 6 independent input lines. The chip measures 6.8x6 cm^2 and comprises more than 900 000 transistors. It consists of an 6x8 array of 16 bit multipliers and accumulators. Every row is provided 2 spare modules which are controlled by soft configuration. The total power consumption of the chip is estimated to be less than 10 Watts when run at 16 MHz (see [5] for details concerning the worst case clock frequency). A full account of the chip architecture and its redundant configuration is found in [6] .

4. LEARNING ARCHITECTURE

In the search mode, (i.e. when the network responds to given weights and input patterns), the data flow has a relatively simple structure, for all a controller has to manage is start and stop the weighting for a layer, initialize subsequent discrimination, transfer the results and proceed for the next layer. In case of the systolic chain, the control would be systolically transmitted along the chain. In the learning mode, however, the data flow program is more involved, and support for a wide variety of neural paradigmas is harder to achieve. Figure 5 shows an architecture common to the learning rules according to Hebb, Widrow-Hoff and error back propagation. Here, the indices p numerate the patterns to be learned, k_i is an index running through the neurons of the i-th layer, $t_{k_3}(p)$ and $y_{k_i}(p)$ indicate the reference output at the last layer and the actual output at the i-th layer respectively. Having computed the $\delta_{k_i}(p)$ in the recursion unit (necessary for MLPs), an additional block of 4x4 multipliers suffice to compute the corrections $\delta W_{k_i k_{i-1}}$. The learning module shown is designed to fit the m-neuron unit, also in the systolic mode. 1- and 2-dimensional systolic arrays composed of these basic modules may be arranged, either for fast learning and/or recall. Similiar architectural schemes have been proposed in [7] .

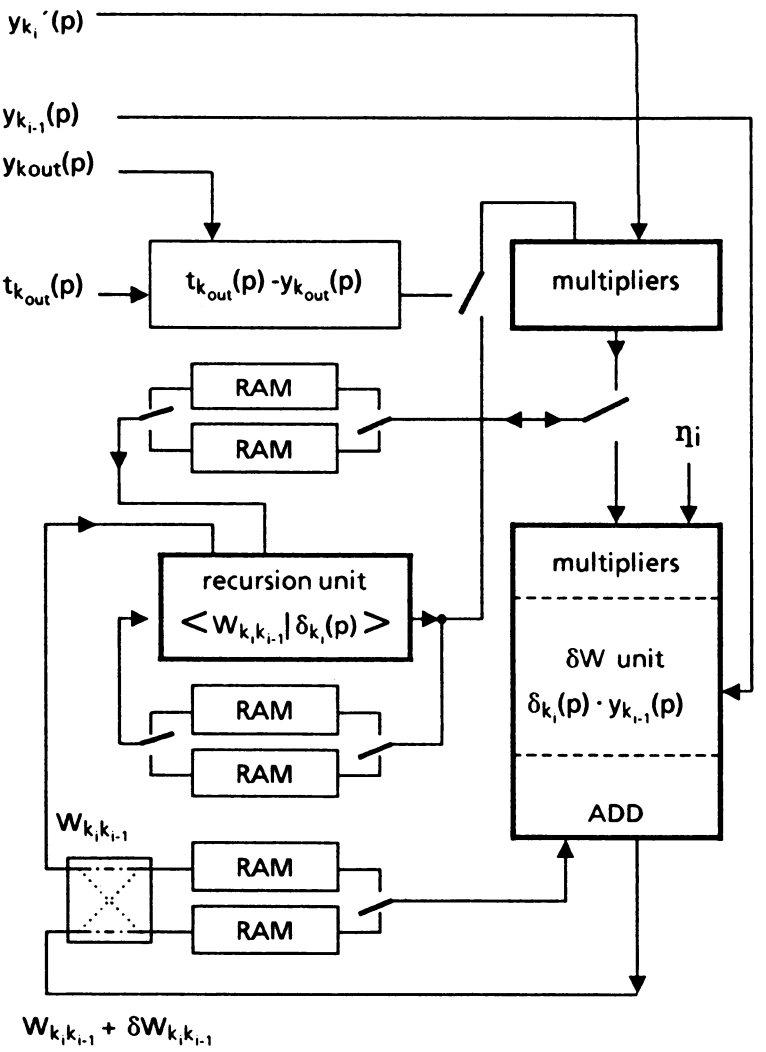


Figure 5 Learning module corresponding to a m- neuron unit

REFERENCES

[1] R. P. Lippmann, "An introduction to computing with neural nets", IEEE ASSP Magazine, pp. 4-22, April 1987

[2] R. Hecht-Nielsen, "Neurocomputing: picking the human brain", IEEE Spectrum, vol. 25, no. 3, pp. 36-41, 1988

[3] "Microelectronics for Artificial Neural Nets", editors H. Klar, U. Ramacher, VDI-Verlag, Düsseldorf, 1989

[4] D. Hammerstrom, J. Bailey, "Why VLSI Implementations of Associative VLCNs require Connection Multiplexing",vol.2, Int. Conf. on Neural Networks, pp. II-173, San Diego, 1988

[5] U. Ramacher, J. Beichter, "A selftesting WSI matrix-matrix multiplier", Proc. of I.F.I.P. workshop on Wafer Scale Integration, Brunel University, Sept. 1987

[6] U. Ramacher, J. Beichter, W. Kamp, "On the design of a selftesting WSI multiplier array", IEEE Proc. of the Int. Conf. on Wafer Scale Integration, San Francisco, Jan. 1989

[7] S. Y. Kung, J. N. Hwang, "Parallel architectures for artificial neural nets", vol.2, IEEE Int. Conf. on Neural Networks, San Diego, July 1988

Optically Configured Phototransistor Neural Networks

Charles F. Neugebauer, Aharon Agranat, and Amnon Yariv
California Institute of Technology
Mail Code 128-95, Pasadena, CA 91125

Introduction:

Modeling neural networks on conventional digital hardware is inherently slower than special purpose hardware simulations. Single processor and coarse grain multiprocessor simulators with high numerical precision are not particularly suited to neural network processing -- a fine grain, low precision task. The highly parallel task of computing synaptic weighting for neural networks can be most efficiently addressed by a hybrid optoelectronic technology[1]. The systems which we develop take advantage of the fact that signal processing in silicon is a mature technology and incorporate optics where silicon fails -- namely the interconnectivity problem.

Network Description:

The basic synaptic accumulation calculation

$$I_i = \sum_j W_{ij} * V_j \qquad \text{Eqn. (1)}$$

(where V_j is the pre-synaptic input, W_{ij} is the connection strength, and I_i is the accumulated input) has been implemented with a variety of technologies, the most common being silicon. VLSI hardware simulators with computation rates on the order of 10^8 - 10^{10} connections per second have been proposed, demonstrating the gains to be had with hardware implementations. The computation rate increase comes at the expense of flexibility, however. Most hardware solutions of Eqn. (1) rely on some form of VLSI memory to store the synaptic connection matrix, W_{ij}. To achieve the high computation rates, the synaptic weight memory access time must be minimized, resulting in the local storage of the synaptic weights within the neural processor IC. In order to change the weights completely to simulate another network, new connection values are time multiplexed into the neural processors -- a time consuming process for large networks. Thus quotes of computation rates of a billion connections per second per chip must be qualified by the fact that they can only simulate small ($<10^3$ neurons, $<10^6$ connections) networks very quickly and larger networks (involving changing the weights) prohibitively slowly. Thus for interesting networks containing biologically relevant numbers of connections ($>10^7$) these implementations must either contain prohibitively large numbers of chips or be constrained by the classic von Neumann bottleneck.

Volume holographic crystals are currently under development that permit the storage and readout of multiple 2-D images with a simple beam indexing scheme. Images are read by the application of a single laser beam, the specific pattern recalled being indexed to the direction of the laser beam. Theoretical limits on the storage capacity of these crystals are about 10^9 bits/cm^3 with readout access times limited by the switching of the single indexing beam. The architecture we propose uses these materials to break the von Neumann bottleneck associated with large networks.

The basic features of the architecture are shown in Fig. 1. The system consists of two main subassemblies: a 2-D spatial light modulator (SLM) using a holographic crystal for storage and an integrated circuit which performs the synaptic accumulation and nonlinear processing. The connection matrix Wij is stored within the SLM which projects an image of the weight matrix onto the IC. The neural processing IC contains a matrix of detectors which convert the intensities (proportional to the weights) into an electrical signal to be used for the synaptic weighting and accumulation. Thus the connection strengths can be loaded into the IC in parallel from the SLM. Many independent sets of connection strengths can be stored within the holographic crystal SLM which is accessed by the direction of a single laser beam. Thus by changing the direction of this single beam, an entire new set of weights (e.g. a different section of a large network) can be loaded into the IC. There is no penalty associated with loading the entire weight matrix, making very large ($>10^5$ neurons, $>10^8$ connections) and very fast (limited by silicon detector response) networks possible. By using parallel optical storage/transfer to solve the memory bottleneck and fast VLSI detectors, a hybrid optoelectronic approach makes very large networks possible.

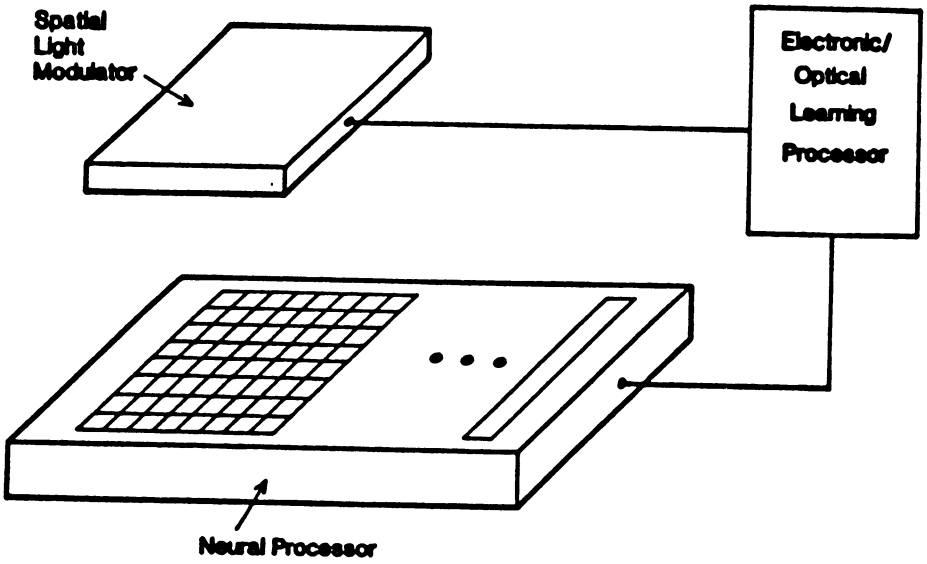


Fig. 1 *Basic Architecture*

The IC is accessed by a conventional digital computer which stores the input and output of the network. The computer deals only with the neurons, not the connections, reducing the computation and storage needs tremendously. In addition, the computer allows conventional access to the neural network -- namely digital I/O (no special interface needed).

Circuit Description:

A typical pwell CMOS process has a parasitic vertical bipolar transistor that makes an excellent photosensor. The standard MOSIS CMOS process produces a phototransistor with a typical current gain of over 200. This NPN bipolar device has its collector (the substrate) tied to V_{dd}. The base region (the pwell) is left floating -- photocurrent generated in the base will be multiplied by the current gain factor. The emitter is formed by a heavy N type diffusion at the surface of the substrate.

An IC implementing an optoelectronic neural processor using phototransistors has been built. A single synapse performs a multiplication and sum operation. The phototransistor synapse contains two devices -- a phototransistor and a p-channel FET, as shown in Fig. 2.

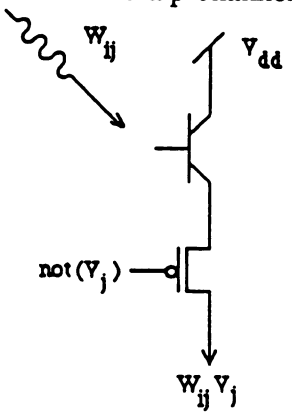


Fig. 2 *Binary Phototransistor Synapse*

The photocurrent generated at each detector is proportional to the respective W_{ij}'s. In this initial implementation, the neurons are binary -- thus a multiplier is simply a switch which can be implemented with a single FET. The gate of the multiplier FET is connected to the column input line. The potential of the j^{th} column line reflects the state of the respective input (V_j). The multiplying FET's at each synapse produce currents proportional to $W_{ij} * V_j$ which are

subsequently summed along a row output line. Thus the accumulation according to Eqn. 1 is accomplished in parallel, providing the synaptic accumulation, I_i, in the form of a current. Each row line is then connected to a decision function which provides binary outputs, as shown in Fig.3. In addition, the IC contains peripheral circuitry for subtracting a weight offset to allow for inhibitory connections.

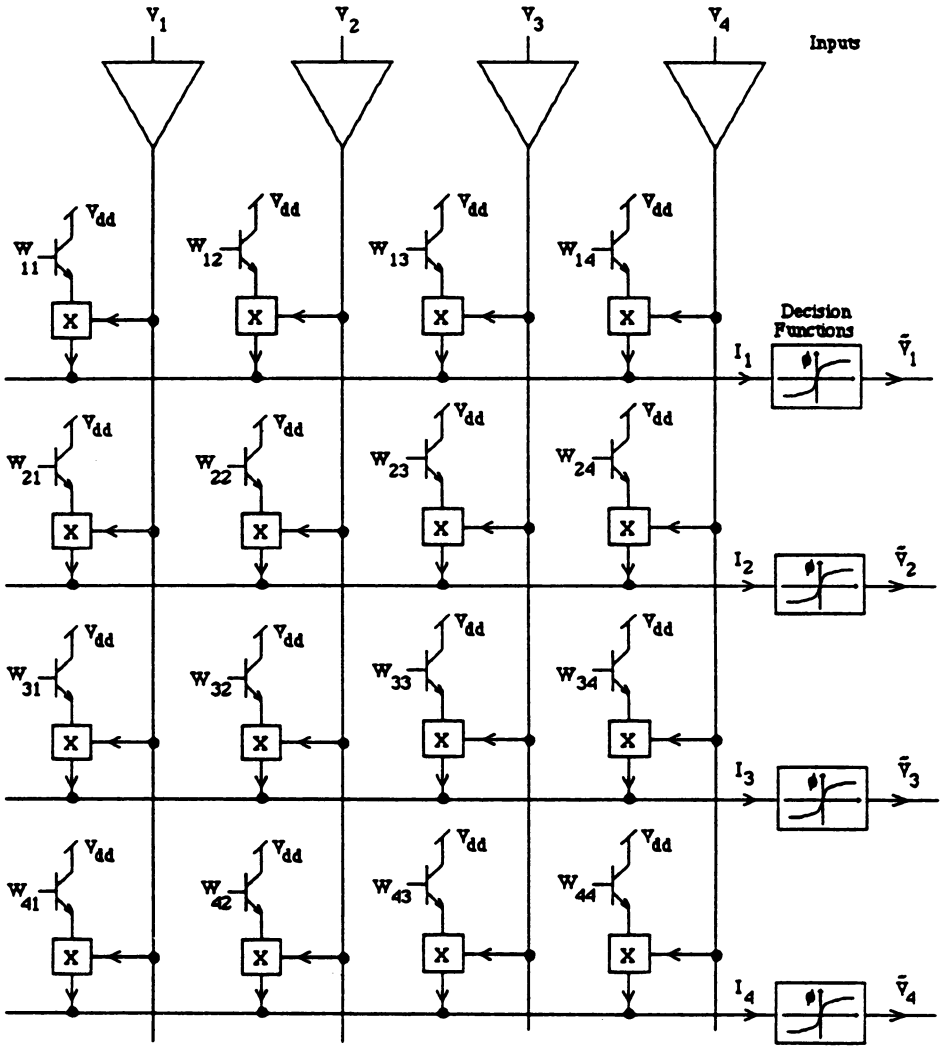


Fig. 3 *Phototransistor Network*

Fabrication Results:

The IC, containing a 32x32 array of synapses and 32 decision functions, was fabricated in MOSIS's 3μm pwell process. The synapse size was approximately 50x50 μm^2. The binary decision function circuits were found to have thresholds within 7-10% uniformity across the chip. The phototransistors themselves could resolve the minimum SLM changes in input intensity, putting their sensitivity at >45dB. The sensitivity of the entire neuron (phototransistors and decision functions) was measured at 35dB which translates into 5-6 bit accuracy. The settling time of the network depends on the illumination strength as shown in Fig.4. A simple, two layer inverting XOR was implemented using a low power CRT as the SLM. Six weights and three neurons were used to demostrate operation, shown in Fig. 5.

Conclusion:

We have demostrated an optoelectronic network that computes 10^3 connections with 5-6 bit accuracy in less than 10μsec in low illumination, giving a computation rate of $>10^8$ connections per second. The architecture can be scaled (this chip is only 3mm on a side) and

illumination increased, resulting in greater performance. More importantly, very large networks can be efficiently computed using holographic crystals as connection storage.

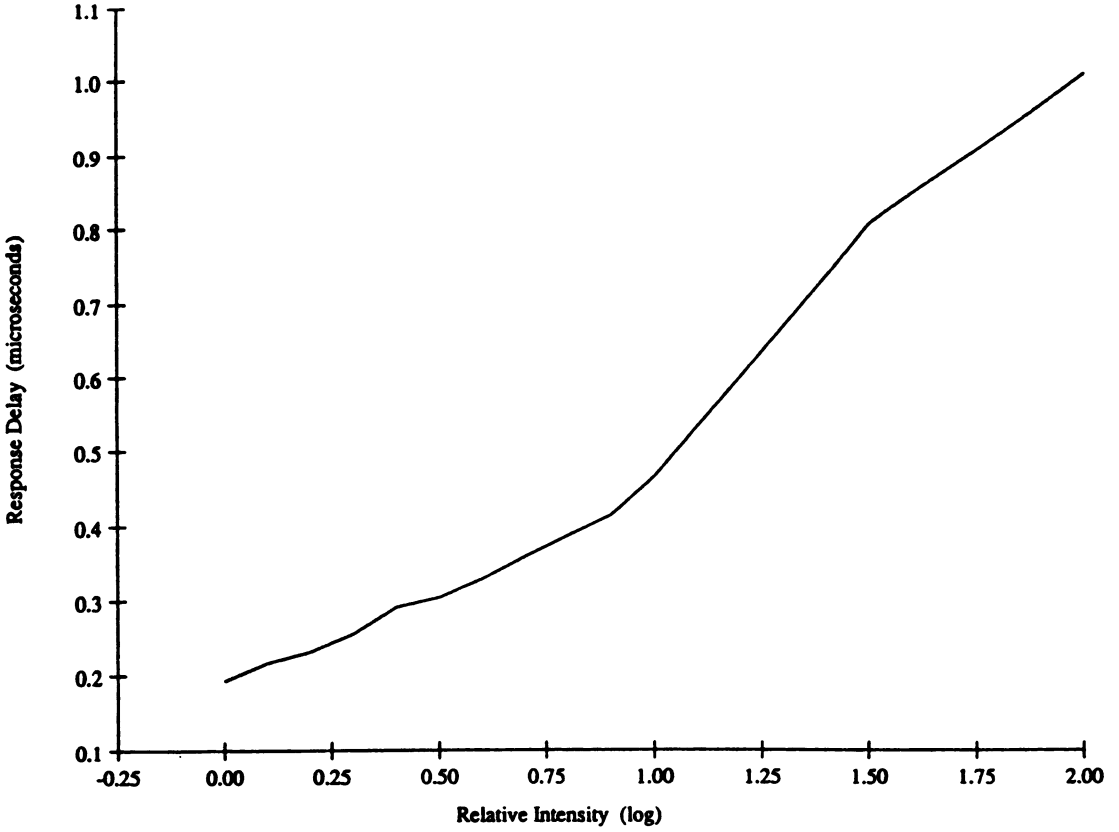


Fig.4 *Neuron Response Time*

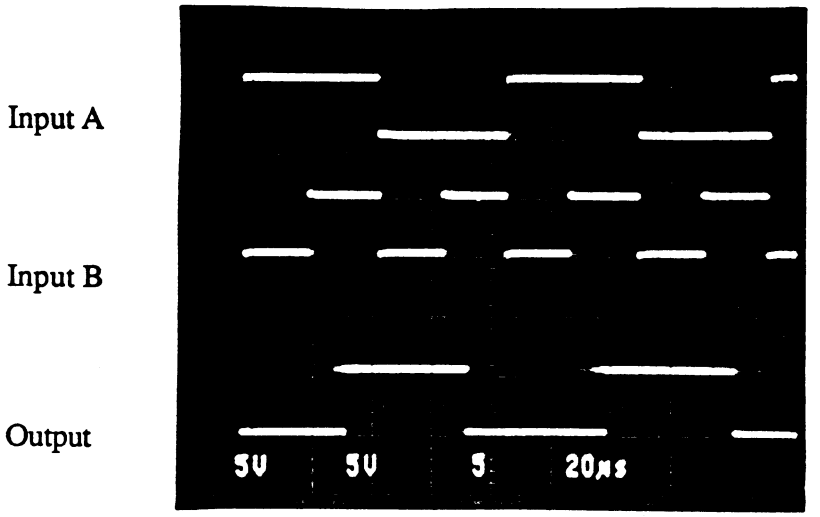


Fig. 5 *Inverse XOR Operation*

References:

1. A. Agranat and A. Yariv, "A New Architecture for a Microelectronic Implementation of Neural Network Models," in *Proceedings, IEEE First Annual International Conference on Neural Networks*, June 1987, Vol. 3, pp 403-409.

Optically implemented Hopfield associative memory using two-dimensional incoherent optical array devices

Kazuhiro NOGUCHI and Toshikazu SAKANO
NTT Transmission Systems Laboratories,
1-2356 Take, Yokosuka-shi Kanagawa-ken, 238-03, JAPAN

Optically implemented two-dimensional(2-D) Hopfield associative memory using a hologram array for interconnection between optical source and detector arrays has been proposed[1], [2]. However, it is difficult to construct such large element number systems because extremely large size hologram elements are necessary. This communication describes a simplified optical Hopfield associative memory using 2-D arranged incoherent source and detector array devices. Polarization encoding and shadow encoding methods for installing the optical interconnection for the Hopfield memory are proposed.

In Hopfield associative memories, the connection matrix $T=\{T_{ij}\}$ is written as

$$T_{ij}=\sum_{k=1}^{M}(2V^k{}_i-1)(2V^k{}_j-1)-M\delta_{ij} \qquad (1),$$

where $V^k(k=1, 2, \cdots, M)$ is a set of stored binary vectors[3]. In networks which have N 2-D arranged elements, the output vector V^{out} is determined by the following operation,

$$V^{out}{}_i=th\left\{\sum_{j=1}^{N}T_{ij}V^{in}{}_j\right\} \qquad (2),$$

$$th(x)=\begin{cases}1 & \text{if } x>0\\ 0 & \text{otherwise}\end{cases} \qquad (3),$$

where V^{in} is the input binary vector. The vector V^{out} is fed back to the input, and the resulting vector converges to the stored vector which is most similar to the input.

In this operation, a matrix represented as

$$T^*{}_{ij}=\sum_{k=1}^{M}(2V^k{}_i-1)(2V^k{}_j-1)$$

$$=\sum_{k=1}^{M}\{EQV(V^k{}_i, V^k{}_j)-XOR(V^k{}_i, V^k{}_j)\} \qquad (4)$$

$$EQV(A, B)=\begin{cases}1 & \text{if } A=B\\ 0 & \text{otherwise}\end{cases} \quad XOR(A, B)=\begin{cases}1 & \text{if } A\neq B\\ 0 & \text{otherwise}\end{cases} \qquad (5)$$

is used as an alternative for the T_{ij} matrix[4]. Thus, the Hopfield associative memory can be optically implemented using the optical EQV and XOR interconnections.

Here, we propose the optical Hopfield associative memory using polarization encoding as shown Fig. 1. This system consists of source and detector arrays which are covered with polarizer arrays, and a spatial filter. The source array contains N×M incoherent optical sources, and the detector array contains N×M detector pairs each of which consists of two adjacent detectors. The source and detector arrays are divided into N segments which contain M sources or M detector pairs. In Fig. 1, arrays are divided into 12 segments containing 4 sources or detector pairs. Each segment corresponds to an element of V^{in} or V^{out}.

Here, we classify all the polarizer states to be "0" if they pass vertically polarized light, and to be "1" if they pass horizontally polarized light. Obviously, light passes through two polarizers from source to detector only when the two polarizer states are equal to each other.

The optical interconnection scheme linking a source segment to a detector segment through a spatial filter is shown in Fig. 2. In this example, each segment contains 4 sources or detector pairs. The spatial filter connects k-th source in a segment only to k-th detector pair in all detector segments in the detector array. The state of the polarizer that covers the k-th source element in the i-th segment agrees with the i-th element of the stored binary vector V^k $(1 \leqq k \leqq M)$. Also, the state of the polarizer that covers one detector element (named "+" detector) in k-th detector pair in the i-th segment agrees with V^k_i, and that with the another element (named "-" detector) agrees with the complement of V^k_i. In the example shown in Fig. 2, the "+" detectors are put at the upper side of the detector pair, and the "-" detectors at the lower side. Then, the light currents $I^{(k)+}_{ij}$ and $I^{(k)-}_{ij}$, which are generated at the k-th "+" and "-" detectors in the j-th segment by the light emitted from the k-th source in the i-th segment, is written as

$$I^{(k)+}_{ij} = I_0 \cdot \mathrm{EQV}(V^k_i, V^k_j) \tag{6}$$

and

$$I^{(k)-}_{ij} = I_0 \cdot \mathrm{XOR}(V^k_i, V^k_j) \tag{7},$$

where I_0 is the generating light current when $V^k_i = V^k_j$. These equations show that optical EQV and XOR interconnections can be formed using polarization encoding.

In the operation of this system, all sources belonging to the i-th segment are simultaneously turned on or off according to the i-th element value of the input binary vector V^{in}. Provided that the I_0 for all optical connections are equal, the photocurrent difference in the i-th detector segment I_i between the total photocurrents generated in all "+" detectors and that generated in all "-" detectors is written as

$$I_i = I_0 \sum_{k=1}^{M} \sum_{j=1}^{N} V^{in}_j \{EQV(V^k_i, V^k_j) - XOR(V^k_i, V^k_j)\} \qquad (8).$$

Therefore, V^{out}_i is determined from the photocurrent difference I_i.

We have constructed a 5×5-element Hopfield memory with 2 stored patterns as shown in Fig. 3(a). The sources are 880-nm LED's, and the detectors are Si-PIN photodiodes. Two patterns, which represent 5×5-dot characters of "A" and "C", are stored in the memory. Figure 3(b) and (c) shows the the demonstration of the Hopfield memory operation. In both cases, one of perfect stored patterns is reconstructed from the part of the stored patterns.

The optical EQV and XOR interconnection can be constructed by shadow encoding without using polarizer arrays. Similarly to polarization-encoding, the Hopfield memory using shadow encoding is assembled from source and detector arrays, which contain N×M sources or detector pairs and are divided into N segments. The interconnection scheme between one source segment and detector segment is shown in Fig. 4. The states of source elements, which agree with corresponding stored vector element, are determined by the source position. In the example in Fig. 4, the upper side is "0" and the lower is "1". The light from a source connects to only one detector in detector pairs through the spatial filter, and another detector is covered with shadow of the spatial filter. The state of the detector pairs is "0" when the "+" detector connects with the state "0" sources, and is "1" when the "-" detector connects with "0" sources. In this condition, the EQV interconnection is constructed between the source and "+" detectors, and the XOR between source and "-" detectors. Therefore, Eqs. (6)~(8) also hold in the shadow encoding model.

The Hopfield memory using the polarization encoding or the shadow encoding described above is constructed very easily if integrated 2-D optical source and detector array devices are used. Therefore, we believe that these encoding methods are suitable for constructing a Hopfield memory with a large number of elements.

To summarize, optically implemented 2-D Hopfield associative memory using incoherent optical array devices was proposed. Optical EQV and XOR interconnections were implemented between source and detector arrays using polarization or shadow encoding. The structure of the system is suitable for constructing the Hopfield memory with a large number element because it is very simple and easy to assemble.

The authors would like to thank Hidetoshi Kimura and Takao Matsumoto for their continuous encouragement.

References

(1). H. J. White, et al., Appl. Opt., 27, 331, (1988).
(2). Ju-Seong Jang, et al., Opt. Lett., 13, 248, (1988).
(3). D. Psaltis, et al., Opt. Lett., 10, 98, (1985).
(4). G. R. Gindi, et al., Appl. Opt., 27, 129, (1988).

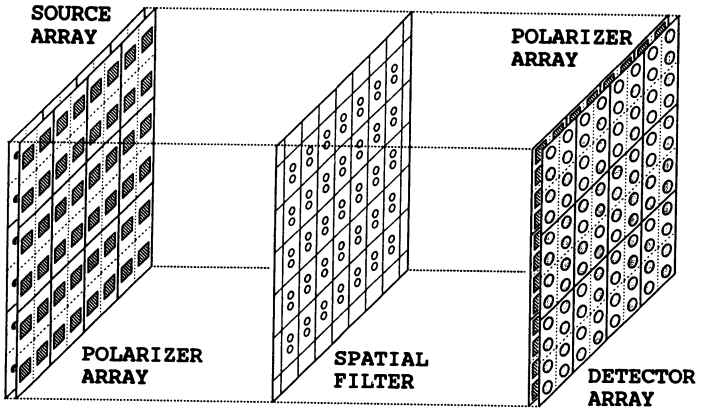


FIG. 1.

Optical Hopfield memory using polarization encoding method.

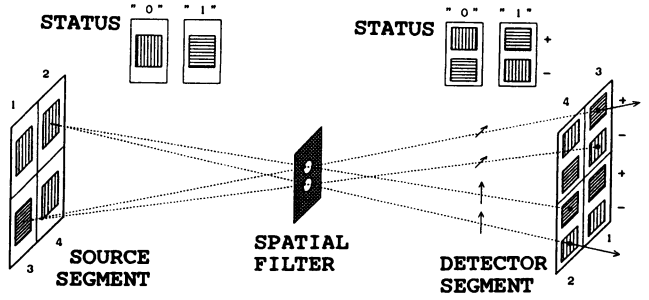


FIG. 2.

Interconnection scheme for the Hopfield memory using polarization encoding.

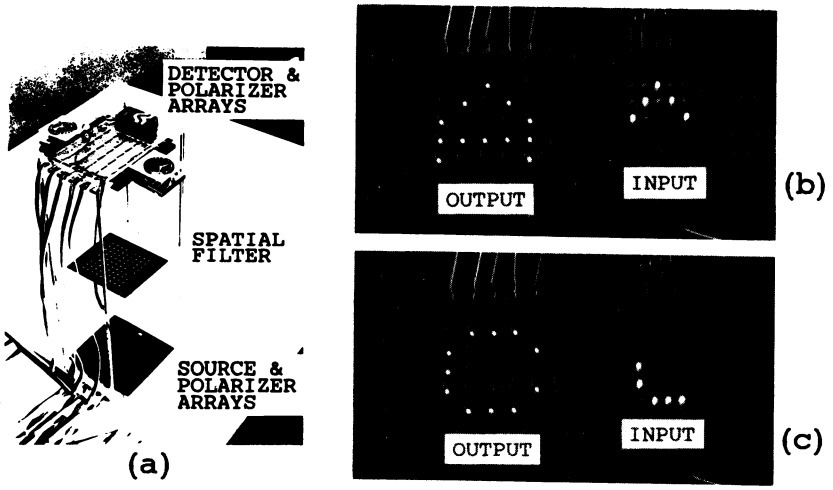


FIG. 3.

Setup and demonstration for 5×5 Hopfield memory.

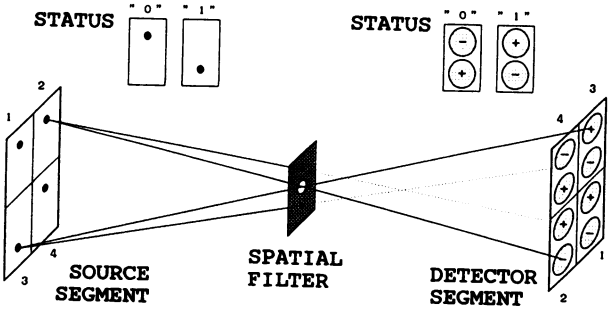


FIG. 4.

Interconnection scheme for the Hopfield memory using shadow encoding.

LEARNING IN OPTICAL NEURAL COMPUTERS

Demetri Psaltis, David Brady, and Ken Hsu

California Institute of Technology, Pasadena, California 91125

Optical holography is useful in artificial neural hardware because of the relatively large size of the networks which can be implemented and because it provides a potentially simple method for dynamically controlling the free parameters of the network. The simplicity of learning in holographic systems is based on the analogy between Hebbian learning and hologram formation: *A holographic connection between two optical modes forms in linear proportion to the product of the activities of the modes.* When this principle is applied to construct large-scale adaptive networks, the hologram must integrate contributions from many modes over many learning cycles. The use of multiple learning cycles (holographic exposures) necessarily restricts the dynamic range of the recorded holograms. In this paper, we describe how these restrictions arise and how they may be partially overcome by periodic copying between short term and long term components of holographic memories.

A basic module for a volume holographic neural system is sketched in Fig. 1. The activity of the i^{th} neuron in this system is represented by an optical signal on the i^{th} pixel at the input plane. A connection is made from the i^{th} neuron to the j^{th} neuron via a holographic grating coupling the mode excited by the i^{th} input with the mode incident on the j^{th} output pixel. The signals diffracted from all the input modes onto the j^{th} output mode are summed and thresholded at the j^{th} output pixel. The strength of the connection is proportional to the the product of the activity of the i^{th} pixel on the input plane and the activity of the j^{th} pixel at the training plane. Light from the j^{th} pixel on the training plane excites the mode which couples into the j^{th} pixel on the output plane. The strength of the connection between the i^{th} and j^{th} neurons is increased if the interference pattern between the corresponding modes is in phase with the hologram which stores the connection and is decreased if the interference pattern and the hologram are out of phase.

In addition to connecting the modes used to record it, a grating coupling a pair of modes results in undesired connections between other pairs of modes. To prevent this degeneracy from constraining the nature of the transformations which can be implemented using a hologram, the modes which are coupled by the hologram must be restricted. Since each mode corresponds to a unique pixel on the input or output plane, only a subset of the available pixels can be used in the interconnection system. Suppose that the input and output planes each consist of N^2 pixels. If the same number of pixels are to be used on the input and output plane, only $N^{\frac{3}{2}}$ of the pixels on each plane may be used in an unconstrained interconnection system. A grid wich samples pixels on the input and output planes such that each pair of input-output pixels may be independently interconnected is shown in Fig. 2. The input pixels are sampled densely as shown at the top of the figure. The output pixels are arranged in lines separated by the width of the input grid. In this figure, $N = 100$.

The most promising technology for recording adaptive volume holograms is based on photorefractive materials. The key advantages of these materials are that they offer relatively high dynamic range, that their response to new recording beams does not degrade under multiple exposure, and that they respond in real-time with no development steps. A photorefractive hologram is formed by the redistribution of photogenerated charge among local traps. While a recorded hologram remains stable in the dark, the excited charge created by writing a new hologram increases the conductivity of

the material and causes the charge patterns corresponding to previous holograms to decay exponentially in time. The time constant of the decay is inversely proportional to the intensity of the writing beams. The decay of previously recorded holograms limits the number of exposures which can be usefully recorded in a photorefractive crystal. We have described previously, [1], an exposure schedule by which an arbitrary number of holograms, M, each with identical diffraction efficiency, may be recorded in a photorefractive material. Unfortunately, recording each hologram with the same strength necessitates a fall off in the efficiency with each successive exposure. The end result is that the diffraction efficiency of each hologram falls off in proportion to $\frac{1}{M^2}$.

Outer-product based storage of associated patterns can be implemented using photorefractive holograms in the system of Fig. 1 by exposing the hologram with a sequence of appropriately sampled patterns on the input and training planes. As an example, we have stored up to twenty associations between random image-name pairs of the sort shown in Fig. 3 using this approach. The sampling grids of Fig. 2 where used in these experiments. The reconstruction fidelity when only a few patterns are recorded was good and we were able to verify the fall off in diffraction efficiency with the square of the number of associations recorded.

By an adaptive system, we mean a system which responds in real time to some function of its input and control signals. In general, the input and control signals will be generated externally and will not drive the system to its "optimal" state in a minimum number of steps. For a system implemented using photorefractive crystals, it is desirable to minimize the number of learning steps in order to preserve the diffraction efficiency, and thus the dynamic range, of the recorded hologram. One method by which the effective number of recording steps can be minimized is to record the hologram in a two step process using a "short term memory" to condense the information stored in each short series of exposures and a "long term memory" to store the connections which emerge over the length of the learning process. This approach is particularly applicable to systems which adapt continuously, in which case an indefinitely long sequence of exposures controls a finite number of independent connections in the hologram.

In this paper we describe and present experimental results from a preliminary implementation of short term-long term storage. The basic idea is to use two holographic media to periodically refresh each other. The architecture of our system is shown in Fig. 4. A series of holograms between a reference plane wave and a set of signal beams is recorded in a cesium doped strontium barium niobate crystal (SBN:Ce). Shutters $S4$ and $S5$ are closed during this operation. In our experiments we used plane waves generated by rotation of the mirror RM as signal beams. The diffraction efficiency of the recorded holograms is monitored continuously using the phase conjugate of the reference wave. The path of the diffracted conjugate to an output CCD is shown as a dashed line in the figure. A self-pumped $BaTiO_3$ phase conjugate mirror is used to generate the conjugate wave. We use a PCM to compensate for the generally poor optical quality of the SBN. (This problem is unique to the particular crystal we used.) When the diffraction efficiency of the photorefractive holograms begins to be unacceptably low, the recorded holograms are copied from the SBN to a second holographic medium, which in our experiments was a thermoplastic plate. The thermoplastic hologram is formed using the diffracted phase conjugate reference and a back-traveling reference wave. Shutters $S2$ and $S4$ are closed. The hologram written on the plate is copied back to the SBN with the original intensities in the signal and reference beams. The original reference beam and the conjugate to the thermoplastic reference are used to create this hologram. Shutters $S1$ and $S5$ are closed during this step. The result is a rejuvenated hologram of each of the signal beams in the SBN. The diffraction efficiency of each hologram is now proportional to $\frac{1}{M}$ as opposed to the previous $\frac{1}{M^2}$. Since the diffraction efficiency per hologram for M superposed patterns is at best $\frac{1}{M}$, this copying scheme allows us to implement adaptation under

multiple exposures with no cost compared to recording the same information in a single exposure.

The temporal behavior of photorefractive holograms may be described by growth in the amplitude of the space charge density proportional to $(1 - e^{-\alpha I t})$ during recording and decay proportional to $e^{-\alpha I t}$ during the recording of successive holograms. I is the recording intensity. The amplitude of the space charge corresponding to the m^{th} hologram when M holograms are recorded is

$$A_m = A_o(1 - e^{-\alpha I t_m}) \exp(-\sum_{m'=m+1}^{M} \alpha I t_{m'}), \tag{1}$$

where A_o is the saturation diffraction efficiency and t_m is the recording time of the m^{th} hologram. A_m is a constant for all m if

$$t_m = (\alpha I)^{-1} \log\left(\frac{1 + (m-1)\chi}{1 + (m-2)\chi}\right), \tag{2}$$

where $\chi = \frac{A_1}{A_o}$.

To begin recording a series of holograms in the system of Fig. 4, we record m_1 holograms on the SBN following the schedule of eqn. (2) for $\chi = 1$. The amplitude of the space charge for each hologram is $\frac{A_o}{m_1}$ at this point. When only a few holograms are recorded, the diffraction efficiency of the optical field may be nonlinear in the space charge amplitude due to pump depletion. We assume, however, that m_1 is large enough that the diffraction has fallen to the linear regime. In this case the diffraction efficiency in intensity for each of the stored holograms is $\frac{\eta_o}{m_1^2}$, where η_o is the saturation diffraction efficiency for a single hologram. We now copy the summed holograms in the SBN onto the thermoplastic plate by using the phase conjugate of the reference beam to read out the crystal. Copying the hologram on the thermoplastic back onto the SBN with the original total intensities in the signal and reference beams results in a restoration of the photorefractive hologram with $\sqrt{m_1}$ times greater amplitude. The reduction by a factor of $(\sqrt{m_1})^{-1}$ in the amplitude of each hologram results from a reduction in the modulation depth with which each hologram is recorded due to the sharing by all m_1 signals of the intensity available in the signal beam. This factor is inherent in the simultaneous recording of m_1 signals. However, the total diffraction efficiency, summed over all m_1 holograms, is restored to its saturation value.

At this point we begin recording another series of new holograms on the SBN using the schedule of eqn. (2) with $\chi = \frac{1}{\sqrt{m_1}}$. We make m_2 exposures in this cycle. In order to maintain a constant diffraction efficiency from the thermoplastic, m_2 is selected such that the total diffraction efficiency of the summed hologram on the SBN falls back to its value after the first m_1 exposures, i. e. $\frac{\eta_o}{m_1}$. After m_2 exposures we copy back to the thermoplastic, back to the SBN and again make holograms until the total diffraction efficiency falls again to $\frac{\eta_o}{m_1}$. From here the process may proceed indefinitely. Each time $M = \sum m_i$ holograms are copied back and forth, the diffraction efficiency for each hologram is restored to $\frac{1}{M}$.

Fig. 5 is a log-log plot of experimental results for recording holograms in using the exposure schedule of Eq. 2 and using periodic copying. The diffraction efficiency of the recorded holograms was monitored in each case using the diffracted phase conjugate reference and the CCD shown in Fig. 4. The solid line in Fig. 5 corresponds to the theoretical M^{-2} decay in the diffraction efficiency per hologram with no copying between

short and long term storage. The *'s are experimental data points for the mean diffracted power of the stored holograms. The dashed line shows the theoretical decay in diffraction per hologram when periodic copying is used. The first 5 holograms are recorded exactly as in Fig. 2. These five are then copied to the thermoplastic and back to the SBN, restoring the diffraction efficiency per hologram to the M^{-1} line, which is the dotted line in the figure. Thermoplastic holograms are also made after 10 and 15 exposures. The #'s represent experimental data points.

In conclusion, photorefractive volume holography remains a very promising technology for implementation of adaptive neural networks. Of the difficulties which must be addressed before large scale networks can be implemented, the most troublesome is the need to find a way to control volume holograms over many exposures without sacrificing too much dynamic range. Significant progress can be made on this problem using the short term-long term memory approach sketched in rough form here.

This work is supported by the Defense Advanced Research Projects Agency. The authors thank Rockwell International and Ratnakar Neurgaonkar for the SBN used in these experiments.

[1] Demetri Psaltis, David Brady, and Kelvin Wagner, *Adaptive optical networks using photorefractive crystals*, Applied Optics, **27**, 1752-1759, 1 May 1988.

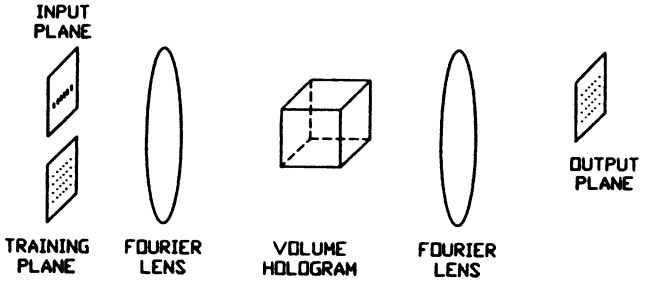


Figure 1. Volume holographic neural system.

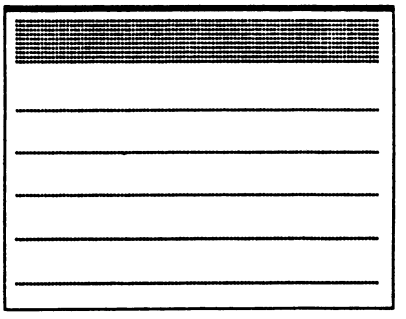


Figure 2. Sampling grids.

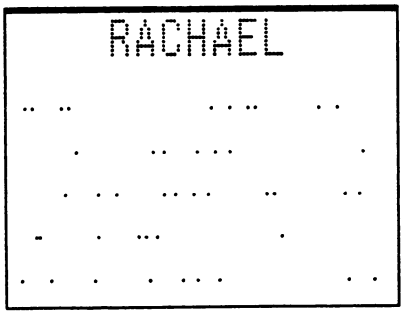


Figure 3. Code-name pair.

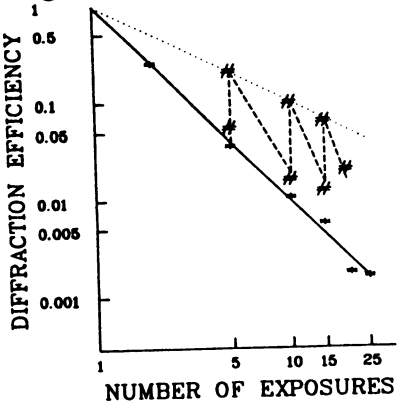


Figure 5. Log mean diffraction efficiency vs. log number of exposures.

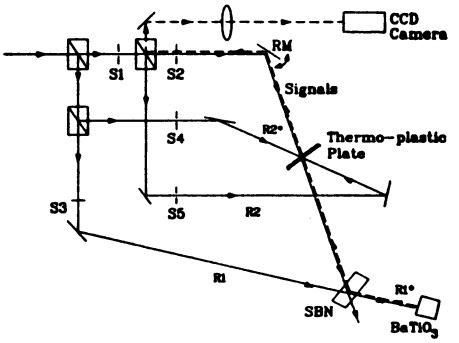


Figure 4. Experimental system layout.

Simulated Annealing Feature Extraction from Occluded and Cluttered Objects

Harold Szu, and Kim Scheff

Code 5756, Naval Research Laboratory, Washington D.C. 20375-5000

Abstract: Human Visual System (HVS) is investigated for solving the occlusion pattern recognition problem. A movie of partially occluded moving objects is simulated in a cluttered imagery sequence. In order to suppress the clutters and the occlusions, the HVS passes the salient features before lowering the image resolution by means of a layer-by-layer coarse graining of the input image sequence feeding through a Fukushima-like neural network, namely a feedforward neural network architecture. In a successive approximation manner, the last layer produces invariant object templates having almost orthogonal features. Such useful coarse-graining processing, based on multiple frames and successive layers, has efficiently achieved the mini-max optimization that is useful for intra-interclass clustering with respect to associative memory recall. Thus, we have adopted in this paper, an imagery sequence of different land vehicles of 3 x 3 pixels tracking over natural terrain within the attentive field of view of 9 x 9 pixels, scanned by a space-filling Peano curve [Space-Scanning Curves for Spatiotemporal Representations.., Szu, Foo, IJCNN-90, Wash. D.C. 1990]. Then the technique of Fast Simulated Annealing (FSA)[Szu, Hartley, Phys.Lett. A122, p.157; Proc. IEEE, V.75,p.1538] is employed for the first time to search for those orthogonal features constrained by object templates. The constraint is obtained by an imprecisely centroid-overlayed sequence due to various degrees of motion-occlusion by moving past trees. These template imperfections can be overcome by the fault tolerance nature, inheritant in associative memory, together with the orthogonality of those extracted features from templates [IJCNN-89, p. I- 547, Wash. D.C. 1989]. In this paper, pattern data have simulated the pointing-and-tracking gray-scaled land vehicles, and the FSA has sped up the automation of binary feature extraction and orthogonalization.

Keywords: Image Processing, Clutter, Occlusion, Simulated Annealing, Peano Curve, Neural Networks

1. Introduction: An animal visual system is known to be sensitive to a moving object, and pays attention to the object immediately with the second look pointing and tracking towards the motion-detected scenery. This kind of attentive image summation is believed to be important for better object template formation, and simultaneous feature extraction, as well as subsequent neural network pattern recognition. A video sequence of a submerged object has been taken downward through a wavy surface of swimming pool. Then, a shifted-and-added technique was used in the second look to produce a sharp template of the submerged object. The second look was a regional resummation that was re-done piecewise with respect to the identical set of imagery that has produced in the first pass a blurred template which had a correct statistics of image pieces through the straightforward pointing-and-tracking summation of many frames (about 16 distorted fields) according to the centroid of the whole frame [1](c.f **Fig. 1** Distorted Fields, Object, Long Term Average, Centroid Correction). This effect had demonstrated the need of a smart sensor concept such as the eye which can see a weak star during an "instance of good seeing"[2] through the turbulent sky. On the contrary, the undiscriminating and dumb telescope camera can only produce a blurred picture of the weak star in the over exposed picture by the whole frame summation based on the straightforward pointing-and-tracking gimbal without any adaptive phase for turbulence medium phase correction mechanism.

Recently, a sequence of distorted imagery that consists of a training set of 15 samples of hand-written characters (each has 4 by 4 pixels, only trained to recognize 3 classes) has demonstrated the ability of generalization: recognize a new class of letter[3]. This was done by means of critical feature extraction using the "mini-max concept" to discover by itself a new class of 5 more hand-written characters by analyzing the "intra-interclass clustering property" on the self-constructed feature space(c.f. **Fig. 2** for 20 samples 4 classes).This example used a table top computer, because the Gram Schmidt orthogonal feature extraction was based on the associative memory employing the Fixed-Point Cycle Two Theorem [4]. Such a procedure of parallel Gram-Schimdt constrained orthogonalization could be exceedingly usefully for a covert communication constrained by call signs and known scrambling instruction, because feature extraction by means of the straightforward projection is not permitted to obliterate critical portion of the signal. However, any practical construction of large set of orthogonal feature vectors could be subject to a realtime processing bottleneck. In this paper, the Fast Simulated Annealing (FSA) technique is adopted to alleviate the bottleneck problem.

Image processing by annealing techniques have been attempted by Geman and Geman [4], Barrett et al.[5], etc. mainly for noise/distortion reduction. Neural networks have been recently applied to pattern recognition by Kohonen, Fukushima, Grossberg, Hopfield, etc.. White noise annealing and neural networks are combined through the Boltzmann Machine by Hinton, Sejnowski, Ackley [6] of which colored noise variant has been referred to as Cauchy Machine[7,8,9].

2. Imagery Sequence: A useful clutter rejection hypothese is that man-made vehicles are designed to minimize the hydrodynamic drag via streamlined shapes and wheels while the natural environment of tree trunks is

mainly vertical against the gravity[unpublished work of J. Landa, H.Szu]. Thus, a sequence of imagery of land vehicles passing trees and bushes is considered, Fig. 3a. When a land vehicle moves by a tree, the partial occlusion of the vehicle by the tree trunk can be easily overcome by a properly pointing tracking, zooming, imaging on the moving vehicle. The image sequence can be averaged and threshold to get rid of the relative motion between the tree and the vehicle, Fig. 3b, together with the 3 by 3 scanning curve and the 9 by 9 scanning curve. The centroid pointing and tracking of the vehicle is assumed to produce the averaged gray-scaled image $< I_c(x,y) >$

$$< I_c(x,y) > = \Sigma_j \; I_{j(x+x_c, y+y_c)} / \text{ frames} \qquad (1)$$

where (x_c, y_c) is a vehicle local centroid coordinate. After a certain threshold, the obscuring effect of the tree and bush will be minimized. Fig.2 (describe the templates)

$$\underline{I}_c(x,y) = \text{Threshold}(< I_c(x,y) >) \qquad (2)$$

Let the critical feature of the template class-c be denoted as $f_c(x,y)$. Then, the performance criterion is the minimum distance between the template of the c-class=1,2 together with the direction cosine in the numerator, and the maximum difference between feature vectors in the denominator. Thus, the mini-max filter energy is

$$E(f_c) = a \Sigma_{c \neq c'} (< f_c | f_{c'} >) + b \Sigma_{c=1,2,...} | f_c - \underline{I}_c |^2 + \Sigma_{c \neq c'} \; d / | f_c - f_{c'} |^2 \qquad (3)$$

where the coefficient of the direction cosine via the inner product $< | >$ may be heavily weighted, eg. by setting a = 10 (relative to b = 1, and d=10). The change of energy is defined as $\Delta E = E_{new} - E_{old}$.

3. Cauchy Simulated Annealing: The physical space is 2-D; but the search space can be 1-D, provided that space-filling scanning technique is adopted here for mapping 2-D imagery space to 1-D search space and yet preserving the local neighborhood relationship[11]. In principle, the space-filling can be done to any desired degree of resolution that is meaningful by the original dynamic range and the image pixel resolution.

The periodic 1-D space is used for the 1-D infinity search space for the Cauchy probability:

(1) Generation of the new state x' from the previous old state x by the Cauchy random number X, and the 1-1 mapping back to 2-D image domain: Choose $f_1 = \underline{I}_{c1}$. Let f_2 be constructed from $\underline{I}_{c2}$.

$$G_T(x' | x' = x + X) = T(t) / \{ [T(t)^2 + X^2] \pi \}; \; T(t) = T_0 / (1 + t) \quad \text{where } T_0 = 100. \text{ in this paper.}$$

The random displacement X is equivalent to the following simple formula: using a random number n, normalized between [0,1], to generate a uniform angles: $(n - 0.5) \times \pi$, between $-\pi/2$ and $+\pi/2$.

$$X = T(t) \tan(\theta) \qquad (4)$$

(2) Canonical Acceptance Criterion

$\underline{I}_{c2}$ pixel toggling for $f_2(x')$ with $\Delta E < 0$ is accepted; the output state energy increase $\Delta E > 0$ is also accepted if the random number generated between [0,0.5] is less than the following acceptance function

$$P_T(\Delta E) = 1 / [1 + \exp(\Delta E / T(t))] \qquad (5)$$

Eq(5) is similar to the Cauchy acceptance criterion [10] when expressed in terms of the energy increment in a simulation by a serial process. To insure the mini-max property,Eq(3), if $f_2(x')$ happens to be togged to be *0*, we can reset $f_1(x')$ to $\underline{I}_1(x')$;otherwise, we chang $f_2(x')$ back to $\underline{I}_2(x')$ and set $f_1(x') = 0$. The final data of f_1 and f_2 are given in Fig.3d. The generating, the accepting, and the energy are plotted in Fig. 4 which shows three segments of the coordinate with respect to the abscissa of 2000 time points in five minute CPU of MAC II(top: searching 9 x 9 states , middle: accepted 9 x 9 states, and bottom: the energy of the visited state). Note that the scattering points about the accepted states is gradually narrowing down due to the Cauchy random walks but never completely because of the occasionally Cauchy random flights. Moreover, the energy occasionally goes up before it goes down, demonstrating the typical characteristics of simulated annealing.

Acknowledge: J. Landa, F. Polkinghorn and A. Tse have helped prepare the land vehicle imagery.

Reference:

[1] H. Szu, J. Blodgett, "Self-reference Spatiotemporal Image-Restoration Technique," J.Opt.Soc.Am., Vol.72, pp.1666-1669, 1982; Also, Szu, Messner, "Adaptive Invariant Novelty Filters," Proc. IEEE, V.74, p.519, 1986.

[2] H. Szu, J. Blodgett, L. Sica, "Local Instances of Good Seeing," Optical Comm., Vol. 35, pp. 317-322, 1980

[3] H. Szu, K. Scheff," Gram-Schmidt Orthogonalization Neural Nets for Optical Character Recognition," Int Joint Conference on Neural Networks, Vol. I, pp. 547-555, Washington D.C., June 18-22, 1989

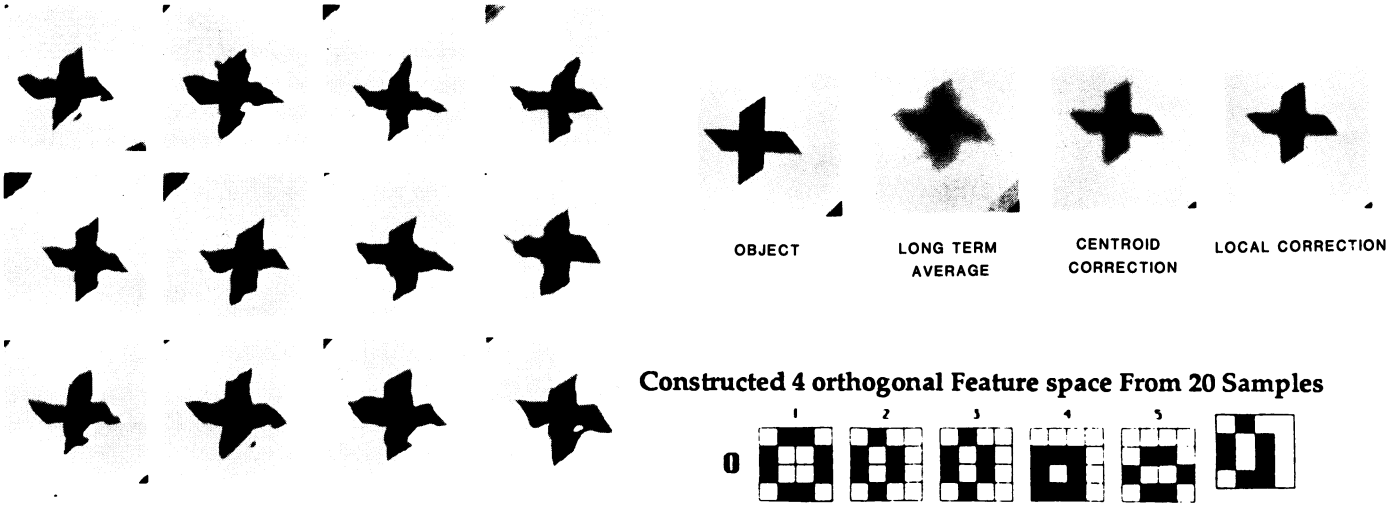
[4] H. Szu, J. Tan, "Can associative memory recognize characters?,"In: U.S. Postal Service Advanced Technology, Washington D.C., pp.1003-1017, May 3-5, 1988

[5] S. Geman, D. Geman, "Stochastic relaxation, Gibbs distribution and Baysian restoration in images," IEEE Trans Patt. Anal. Mach. Int. Vol. PAMI-6, pp.721-741, Nov. 1984

[6] W. E. Smith, H. H. Barrett, R. G. Paxman,"Reconstruction of objects from coded images by simulated annealing," Optics Letters, Vol.8, pp199-201, April 1983

[7] G. E. Hinton, T. J. Sejnowski, D.H. Ackley, "Boltzmann Machines: Constrained Satisfaction Networks that Learn," CMU-CS-84-119, Carnegie Mellon Univ. May, 1984. "Parallel Distributed Processing, Vol. I , Vol.II Edited by J. McCelland, D. Rumelhart, PDP Group, MIT Press, 1986
[8] H. Szu, "Fast Simulated Annealing,", In: "Neural Networks for Computing," AIP Conf. Vol. 15, pp. 420-425, Edited by J. Denker, Snow Bird U.T., 1987; Also, Phys. Letters A 122,p.157, Jun 8, 1987; Proc.IEEE, V. 75, p.1538.
[9] K. Scheff, "1-D Optical Cauchy Machine Infinite Film Spectrum Search, " Int. Conf.Neural Networks-87, P. III-673, San Diego 1987
[10] Y. Tarkefuji, H.Szu, "Parallel Distributed Cauchy Machine," Int.Joint Conf. Neural Networks-89, P. I-529, Washington D.C. June 18-22, 1989
[11] H. Szu, S. Foo,"Space-Scanning Curves for Spatiotemporal Representations, useful for large scale neural network computing," Int. Joint Conf.Neural Networks-90, Washington D.C. Jan.15-18, 1990

Fig. 1 Distorted Fields, Object, Long Term Average, Multiple-Frame Centroid-Corrected Template



Constructed 4 orthogonal Feature space From 20 Samples

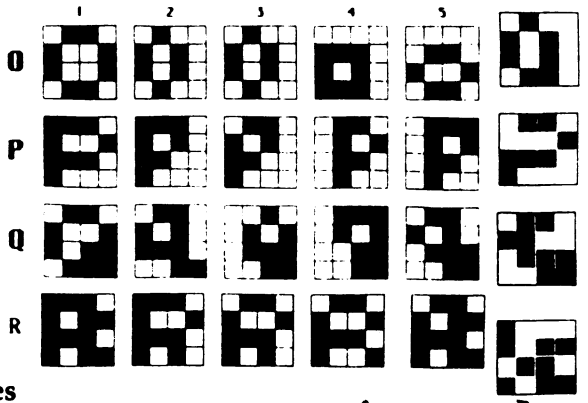


Fig. 3 a Occluded and Cluttered Imagery of Land Vehicles

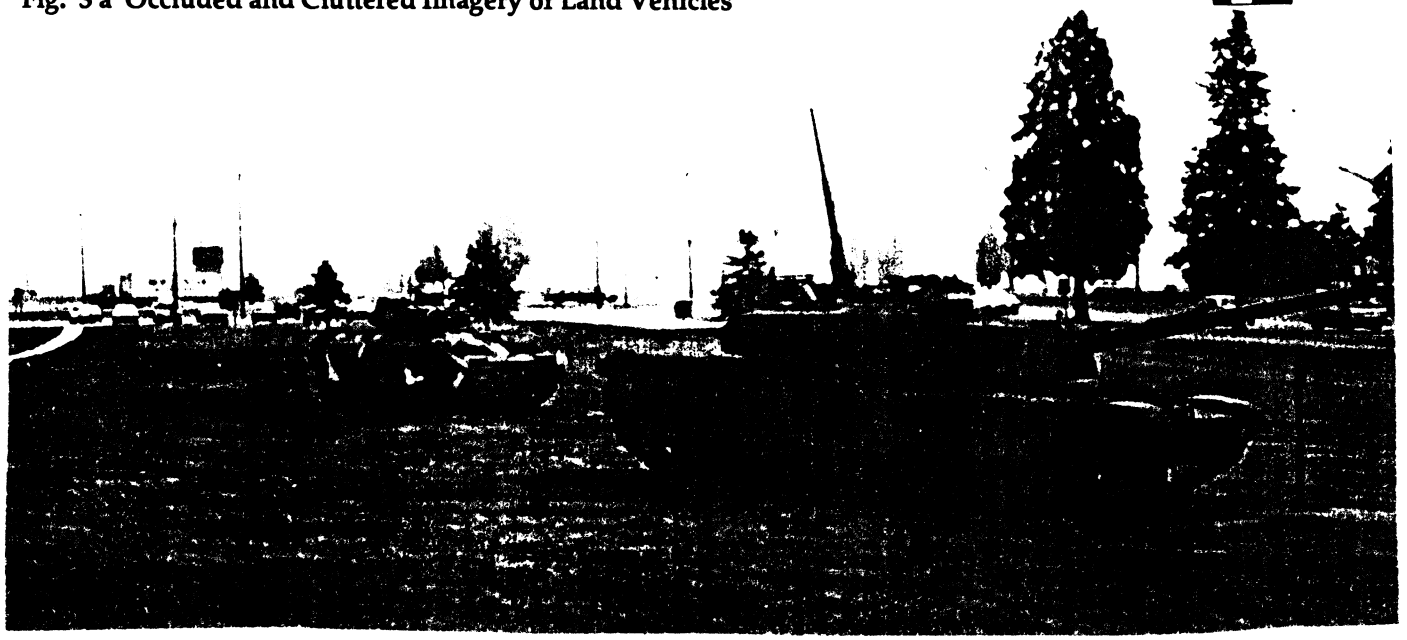


Fig. 3 b Gray-Scaled Templates in terms of 3 x3 and 9x9 Sanning Curves

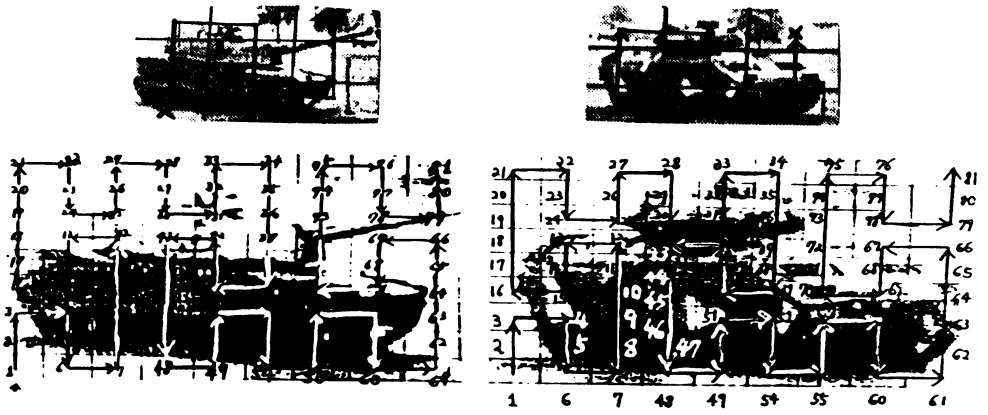


Fig. 3c Binary Features

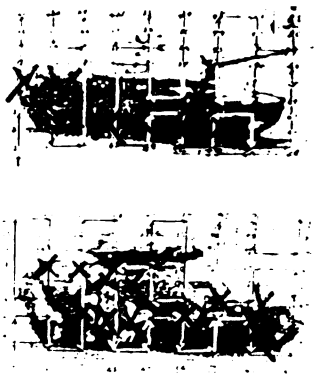
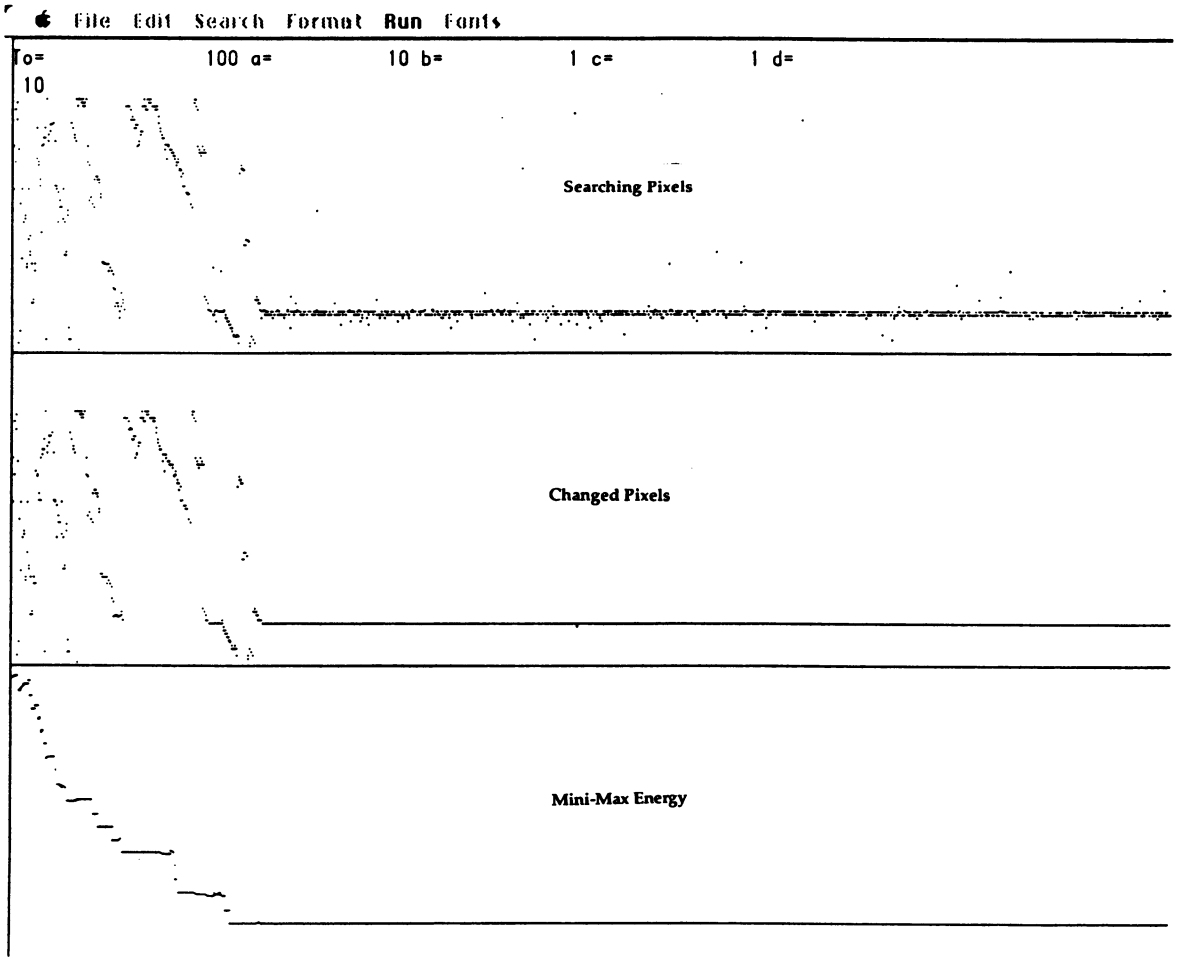


Fig. 3d Output Data of Feature Vector in terms of the 9x9 Scanning Coordinates, shown above

```
<f1|f2>=        0 energy=       31.4
|f1 - I1|=     15 |f2 - I2|=    16
To=           100 a=          10 b=          1 c=          1 d=
 10
feature vector of tank f1:
 0  0  0  1  1  0  0  1  0  0  0  0  0  1  1  1  1  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  0  0  1  0  0  1  0  1
 1  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0
 0  0  0
feature vector of carrier f2:
 0  0  0  0  0  0  0  0  1  0  0  1  1  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  1  0  0  1  0  0  0  0  1  0
 0  0  0  1  1  1  0  0  0  1  1  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0
 0  0  0
```

```
File  Edit  Search  Format  Run  Fonts
inner_product = 20
template vector of tank I1:
 0  0  0  1  1  0  0  1  1  0  1  0  0  1  1  1  1  1  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  1  0  0  1  0  0  1  0  1  1  0  0  1  1  1
 1  0  0  1  1  1  1  1  0  0  0  0  0  0  0  1  0  1  1  1  1  1  0  0  0  0  0  1
 1  0  0
template vector of carrier I2:
 0  0  0  1  1  0  0  1  1  0  0  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0
 0  0  0  1  1  0  0  0  0  0  0  1  0  0  0  0  1  1  0  0  1  1  0  0  1  1  1
 1  0  0  1  1  1  1  1  0  0  1  1  0  0  0  0  0  1  1  0  0  0  0  0  0  0  0
 0  0  0
```

Fig. 4 Fast Simulated Annealing searching 81 binary pixels (for 2 to 81 power possibility)



System Design for a Second Generation Neurocomputer*

Dan Hammerstrom
Eric Means
Department of Computer Science and Engineering
Oregon Graduate Center
19600 Von Neumann Drive
Beaverton, Oregon 97006-1999

1 Introduction

In their organization and operation neural networks differ radically from conventional computer systems. However, it is likely that the systems that may eventually be built around them will be remarkably similar to the traditional computers that these networks are designed to augment. This is because the business of the system architect is to build systems that solve problems at a reasonable cost: all else is secondary to this goal. And viewing neurocomputer architecture in terms of such traditional issues as communication bandwidth, processor utilization, and memory system organization lends a problem–solving perspective to the entire system design.

The bulk of research reported today treats neurocomputer design in a very narrow focus: in most cases the goal is simply to design an electronic analogue to a biological neuron. Despite the significant achievements in this regard, it can be argued that this approach sidesteps the fundamental goal of building practical problem–solving systems. This is largely due to the inordinate burdens that a faithful neuron design would place on the surrounding system. For example, pure analog processors often require real–valued inputs to each synapse. Few communication systems (other than direct optical input) could begin to deliver the required millions of analog signals that a complete system would require. As another example, many proposed designs push the burden of neural learning off–chip, slowing it to the point of eliminating the performance gains achieved by using a neurocomputer in the first place.

These common situations result from a premature emphasis on elegant circuit design that relegates architectural issues to secondary status. This is surprising, for it is the traditional architectural issues mentioned above that really determine the

*This work was supported in part by the Semiconductor Research Corporation contract no. 86–10–097, and by the Office of Naval Research, ONR contract no. N00014 88 K 0329.

viability of the system. Given models of neural operation and network structure, the primary task of the neurocomputer architect is to select the best set of implementation strategies to implement them. Analog versus digital computation is only one of many questions that must be addressed, and its determination must be balanced with the other requirements of the entire system.

We believe that the three key issues in neurocomputer architecture are (1) communication, (2) weight representation and learning, and (3) neural computation. These represent special cases of traditional system design considerations, and will be briefly discussed below. When analyzing the target neural network model, as much attention should be paid to the model's temporal characteristics as to its connectivity structure and learning behavior. Many significant economies are possible if concurrent utilization in any of these areas is sparse.

Looming above all implementation issues is the connectivity dilemma. Our research has shown that an attempt to fabricate a neural network consisting of one million neurons, each connected to its one thousand nearest neighbors in a two–dimensional grid, would result in a VLSI wafer approximately 85 square meters in area [1]. Most of this area is consumed in routing the one billion metal wires. Multiplexing is one technique available to reduce this area appreciably, but only if the target model can tolerate the reduced availability of interconnect. We are currently investigating architectures that have appropriate static characteristics such as sparse and spatially local interconnect, and appropriate dynamic characteristics such as sparse temporal behavior, allowing the shared utilization of valuable communication resources, a technique that biological systems seem to exploit.

Compromises that must be made to create a practical communication system have impact elsewhere. Neural computation has been the favorite topic of neurocomputer research, and most proposed systems feature the parallel nature of analog multiplication, essentially having a processor per synapse. These approaches usually require that all inputs to the neuron be available coincidently, a requirement unlikely to be satisfied by any communication system based on shared wires. Even if all inputs could be made available simultaneously, the attraction of parallel multiplication is dimmed if only a small subset of inputs actively take part in the computation. If input activation is low, which is common in many associative structures, it may be possible to share computation resources in the same manner as was interconnect, and to consider serial digital or hybrid analog/digital approaches for multiplication. Again, this kind of compromise depends on the temporal behavior of the particular neural model being implemented, and exemplifies the sort of questions that system architects must face before committing to alternate forms of computation.

If the area implications of direct interconnect do not seriously impact the pure ideal of distributed, electronic nervous systems, the difficulties posed in learning and weight representation might. Thousands of parallel multipliers require a correspondingly enormous memory system bandwidth to feed the inputs and store the outputs. It is difficult to conceive of a single data path off–chip to memory that could approach the required performance. Instead, local storage of synaptic weights in close proximity to the multiplication hardware is the preferred solution. The

challenge is to combine a memory system compatible with the local computation hardware. Even more troublesome is the learning task. The hardware required to enforce local learning rules can be prohibitively expensive unless ways can be found to share the area burden among multiple processors. For example, at OGC we have designed and built a hybrid analog digital implementation of the Klopf/Kosko differential Hebbian algorithm. This learning rule is not atypical of biological learning mechanisms. By the time we had integrated all the circuitry required to perform the various differentiations and temporal integration, the per synapse circuitry had grown to quite unreasonable proportions. As we try to implement such functionality we are increasingly impressed with Nature's sophisticated and compact mechanisms. In a manner analogous to using multiplexed interconnect and processors, such learning can only be contemplated if the target neural model limits coincident learning to small subsets of the total number of neurons. In the associative structures we are studying, learning occurs at a fraction of the total synapses at any point in time.

Neurocomputer design as expressed here embraces an enormous design space, a much broader one than the narrow focus on Σ–Π neuron design. This architectural perspective embodies a problem–solving approach consistent with traditional methods of system design. The key steps in neurocomputer design first require breaking the system into its three main architectural components, and then closely examining the target model's structure and temporal characteristics, as well as their complex interrelationships. The questions for the architect involve selecting the best set of implementation strategies consistent with the requirements of the model.

One theme mentioned repeatedly here is the focus on sparse activity in the model that allows us to cost–effectively leverage shared resources. Many currently popular neural models such as backpropogation are not compatible with this technique because of the simultaneous activity across multiple processors and communication lines. Some of the work involved in neurocomputer design, then, involves looking for those models that possess "sparse" characteristics, in both the temporal and spatial domains, so that architectural economies can be considered. Much of our research is typified by this type of effort, and interestingly enough some biological models appear to fit many of our requirements. One such example is the pyriform cortex model of Gary Lynch, Rick Granger and their colleagues at the University of California at Irvine [3].

It is our belief that these "second generation" neural network models will be more closely inspired by biology and will have the size and power for solving some of the more truly difficult problems that currently sit at the boundary between the messy, analog, real world and discrete world of the digital computer.

Although we have couched our discussion in the context of our own research in silicon cortex models [2], we feel that much of this is broadly applicable to general neurocomputer design. The take–home message of this paper can be summarized quite simply: beware of local optimizations, for neurocomputer design requires a systems perspective.

References

[1] Jim Bailey and Dan Hammerstrom." Why VLSI implementations of associative VLCNs require connection multiplexing."In *Proceedings of the 1988 International Conference on Neural Networks*, June 1988.

[2] Jim Bailey, Dan Hammerstrom, John Mates, and Mike Rudnick."Silicon Association Cortex. In S. F. Zornetzer", J. L. Davis, and C. Lau, editors, *An Introduction to Neural and Electronic Networks.* Academic Press, August 1989.

[3] Gary Lynch, Richard Granger, and Jose Ambros-Ingerson. "Derivation of encoding characteristics of layer II cerebral cortex." *Journal of Cognitive Neuroscience*, 1(1):61–87, 1989.

A Parallel Implementation of Kohonen Feature Maps on the Warp Systolic Computer

Richard Mann and Simon Haykin
Communications Research Laboratory
McMaster University
Hamilton, Ontario, Canada
L8S 4K1

Abstract

We describe a parallel implementation of Kohonen self-organizing feature maps based on an *epoch update*. That is, for a set, or epoch, of training patterns the weight changes are calculated independently for each pattern and added at the end of the epoch cycle. Use of the epoch update allows efficient parallel implementation, as demonstrated on a Warp systolic computer, with simulation speeds ranging from 6 to 12 million connections per second. Experimental results indicate, that for a small epoch size (10 to 20 patterns), convergence is similar to the original algorithm. It is observed, however, that the epoch update causes instability of the ordering process and "folded" maps sometimes result. To ensure topological ordering it is recommended to use a small number of iterations of the exact algorithm to initialize the map to an ordered state before using the parallel algorithm.

I. Introduction

The Kohonen self-organizing feature map [1] is a two dimensional mesh of neurons each with a weight vector, $\mathbf{w}_i$. During the *search phase*, each node, i, computes the Euclidean distance between its weight vector and the input vector, $\mathbf{x}$,

$$\eta_i = ||\mathbf{x} - \mathbf{w}_i|| \tag{1}$$

and chooses the closest neuron (minimum η_i) called the *winning cell*. During the *update phase*, a small number of neurons within a neighbourhood around (and including) the winning cell are updated,

$$\mathbf{w}_i = \mathbf{w}_i + \alpha(\mathbf{x} - \mathbf{w}_i) \tag{2}$$

where α is a small learning constant. The update neighbourhood usually begins large (to include about half the cells in the network), and decays slowly with time, until it includes only the winning cell.

A parallel implementation of the Kohonen algorithm is desired because of the large cost to compute (1) for every neuron in the network to find the winning cell, and the large number of iterations required for the update rule (2) to converge.

The Kohonen algorithm is difficult to implement on a parallel computer as *global communication* is required to broadcast the input vector $\mathbf{x}$ to all neurons in the network and to find the winning neuron in the network. Current attempts at a parallel implementation of the Kohonen algorithm depend on global communication in analog hardware [4] or a global broadcast facility if the network can be implemented on a single digital chip [5]. Although a fully parallel implementation will require the global communication available in analog or optical computers, it is possible to achieve limited parallelism using digital computers if the number of processors is small relative to the number of neurons in the network.

This paper presents a parallel implementation for the Warp systolic computer. The implementation based on the epoch update is similar to the implementation of the back propagation algorithm for the Warp [3].

II. Partitioning the Problem for Systolic Architectures

The Warp systolic computer [2] shown in figure 1 is a linear array of 10 computing cells, connected to an I/O processor at either end, and driven by a Sun "host". The elements are systolic in the sense that communication is local (left or right neighbour), and that each cell has a simple function (floating point add and multiply).

The simplest implementation called *network partitioning* [3], is to divide the neurons amoung the processors. The feed-forward operations of (1) can be calculated independently, but global communication will be required to find the global winning neuron. This approach is efficient as long as the number of processors is small relative to the number of neurons. For the Warp implementation, however, this approach was abandoned due to the programming complexity of dividing the neurons amoung the processors, and of implementing a variable sized update region (neighbourhood) which may span more than one processing cell.

For the Warp implementation, a second approach, called *data partitioning* [3], was used. In this approach, the input patterns, rather than the neurons are divided amoung the processors. A different pattern is given to each processing cell, and the network updates are calculated independently for each pattern. At the end of the cycle, all of the network changes are added to provide an updated network. Each set of patterns used for one cycle is called an epoch. Assuming a small epoch size and small weight changes for each pattern, this update is expected to give results similar to the exact algorithm.

III. Warp Implementation

The epoch update was implemented on the Warp architecture shown in figure 1. One training pattern is placed in each of the first 9 processors. During the search phase, the weights *flow* through the array from the left to the right, with each cell computing the activation of the neurons in the network given by (1) and choosing the winning neuron. During the update phase, the weights flow through again, with each cell outputting the weight changes given by (2). The weights flow along the upper (X) channel, while the weight changes are accumulated along the lower (Y) channel. The tenth cell inputs the weights and the weight changes, adds them, and outputs the updated weights.

Two changes were made to increase the speed on the Warp. The first change was to put two patterns in every Warp cell since the I/O speed of the Warp is limited to approximately on half of the computation speed. This results in an epoch size of 18 patterns. The second change is to calculate the similarity measure of (1) as a dot product. Since the squared Euclidean distance can be written,

$$\|\mathbf{x} - \mathbf{w}_i\|^2 = \|\mathbf{w}_i\|^2 + \|\mathbf{x}\|^2 - 2\mathbf{x}^T\mathbf{w}_i \tag{3}$$

it can be calculated using the dot product, $\mathbf{x}^T\mathbf{w}_i$, if the magnitudes of $\mathbf{w}_i$ and $\mathbf{x}$ are known. This requires that the final cell compute the magnitudes of the weight vectors, and that the magnitude of the input patterns be calculated as a pre-processing step.

The implementation of (3) requires 1 computation cycle per weight on the Warp, while the update rule given by (2) requires 2 cycles per weight. The compiled code for the Warp performs the search and adapt cycle in approximately 4 cycles per weight, corresponding to a maximum speed of 12.5 million connections per second. The actual simulation speed ranges from 6 to 12 million connections per second depending on the network size. This is due to a fixed communication overhead of 0.01 seconds each time an epoch of patterns is started on the Warp array. For example, a network of 1024 neurons (*32-by-32* mesh), with 16 inputs, runs at 29 epochs per second. This corresponds to a speed of 522 iterations per second or 8 million connections per second. The same network with 128 inputs will run at 5.3 epochs per second, a speed of 95 iterations per second or 12.5 million connections per second.

IV. Experimental Results

The effect of the epoch update is to average several individual updates. For a small number of patterns the epoch update appears to have similar performance to the exact algorithm, but suffers from two limitations. The first limitation is that for stability, the size of the learning constant, α is limited to $\alpha \leq 1/K$, where K is the size of the epoch. This provides a practical limit on the epoch size. The second limitation is that, because of the epoch update, the self-organization process occasionally fails, giving maps which are not topologically ordered.

Maps were trained with both 2 and 8 dimensional inputs to compare the epoch update to the exact algorithm. The two dimensional data shown in figure 2 consists of two Guassian peaks, the first with a variance of 1 centred at (0,0) and the second with a variance of 2 centred at (2.32,0). The 8 dimensional data had the same two peaks, with centres at (0,0,0,0,0,0,0,0) and (2.32,0,0,0,0,0,0,0) respectively.

For the 2-dimensional input, *16-by-16* maps were trained with the input of figure 2, a learning rate of .05 and a neighbourhood decaying linearly from 8 to 0 during the first 3600 iterations. Figure 3 shows the exact algorithm, while figure 4 shows the similar results of the epoch version. Figure 5 compares the convergence of the exact (solid line) and the epoch (dotted line) algorithm based on *mean squared error* calculated as the map evolves. The convergence of the exact and epoch maps was almost identical when the same learning rate (.05) was used.

In the second experiment a *32-by-32* map was trained with the 8-dimensional data set. The learning constant was set to .05 and the neighbourhood was decayed linearly from 16 to 0 during the first 7200 iterations. Figure 6 shows that the exact and epoch versions have similar convergence in terms of mean

squared error. Figure 7 shows the topological ordering of the exact algorithm (solid line) and the epoch update (dotted line) using a variant of a method proposed by Lampinen [6]. A line is drawn in the input space moving from neuron (1,1) to (2,2), (3,3), and so on up to neuron (32,32), while the path of the line is plotted by the position in the network. If the map is topologically ordered, the path should follow the diagonal as in figure 7. Figure 8 shows how the path deviates from the diagonal for a map which is not topologically ordered, as is occasionally found when the epoch update is used.

V. Summary and Conclusions

A parallel implementation of the Kohonen self-organizing feature map algorithm is proposed based upon an epoch update. An efficient implementation on the Warp systolic computer achieved a learning rate of between 6 and 12 million connections per second depending on the network size.

By using the measurements of mean squared error and topological ordering, it is possible to evaluate the convergence the algorithm for maps with any input dimension. Experiments have shown that for a small epoch size (10 to 20 patterns), most runs of the epoch and the exact algorithm provide similar performance. It has been observed, however, that as the epoch size increases, failure of the topological ordering becomes more likely.

Two alternatives are being considered to improve the reliablity of the topological ordering process. The first idea is to train the map with the exact algorithm for the first 100 to 1000 iterations, as the early iterations, called the *ordering phase*, are responsible for the topological ordering of the map [1]. This could be implemented on the Warp by disabling learning on all but the first systolic cell for the first 100 to 1000 epochs. For the longer *convergence phase*, the network could be trained with a full 18 patterns per epoch. The second idea is to start the network in an ordered state before learning begins by choosing an initial set of ordered weights.

VI. References

1. T. Kohonen, *Self-Organization and Associative Memory*, 2nd ed., Springer, Berlin, 1988.
2. H.T. Kung et al. "The Warp Computer: Architecture, Implementation and Performance",*IEEE Trans. Computers*, Dec. 1987.
3. D.A. Pomeerleau, G.L. Gusciora, D.S. Touretzky, H.T. Kung, "Neural Network Simulation at Warp Speed: How We Got 17 Million Connections Per Second", Proc. IEEE ICNN-88, San Diego, CA, July 24-27, 1988, II-143-150.
4. J.R. Mann, S. Gilbert, "An Analog Self-Organizing Neural Network Chip", David S. Touretzky (ed.), *Advances in Neural Information Processing Systems I*, Morgan Kaufmann Publishers, 1989.
5. D.E. Van den Bout, T.K. Miller III,"TinMANN: the Integer Markovian Artificial Neural Network", Proc. IJCNN-89, Washington, D.C., June 18-22, 1989, II-205.
6. J. Lampinen, E. Oja, "Fast Self-Organization by the Probing Algorithm", Proc. IJCNN-89, Washington, D.C., June 18-22, 1989, II-503.

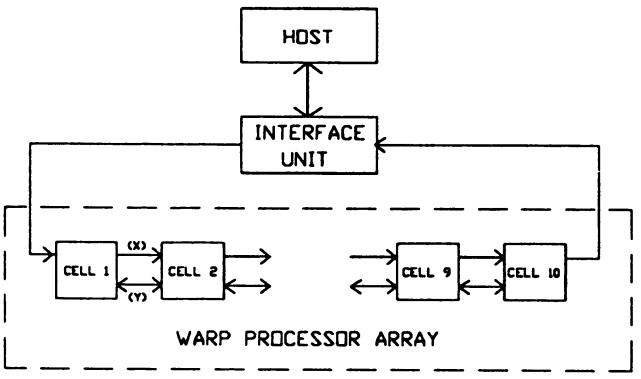


Fig 1: Warp Systolic Computer

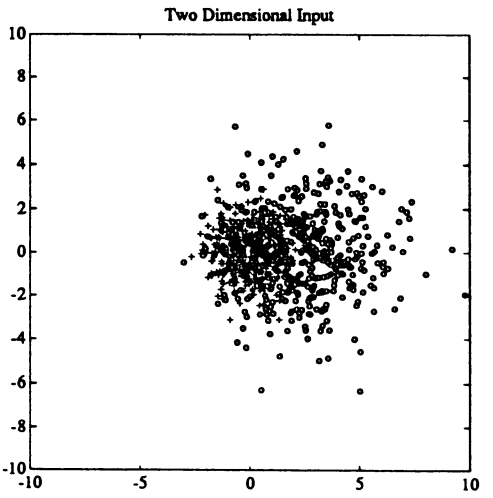
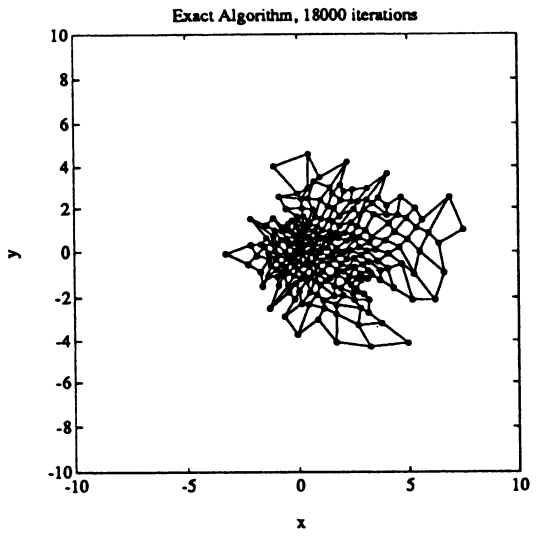


Fig 2: Training Input

Fig 3 (L)
Fig 4 (R)



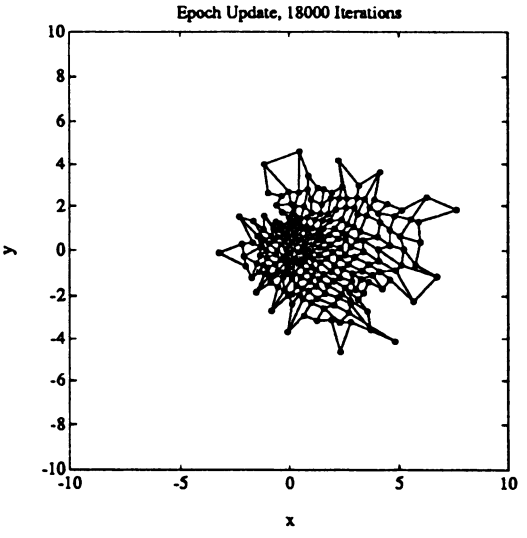
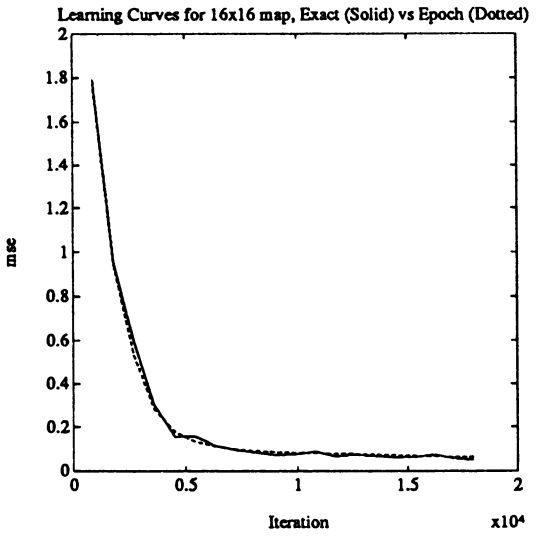


Fig 5 (L)
Fig 6 (R)



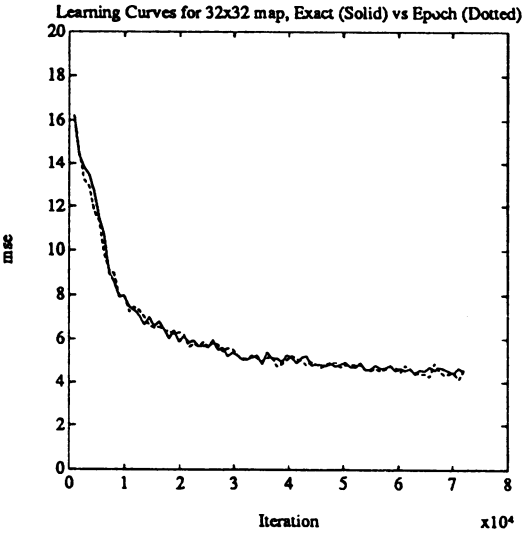
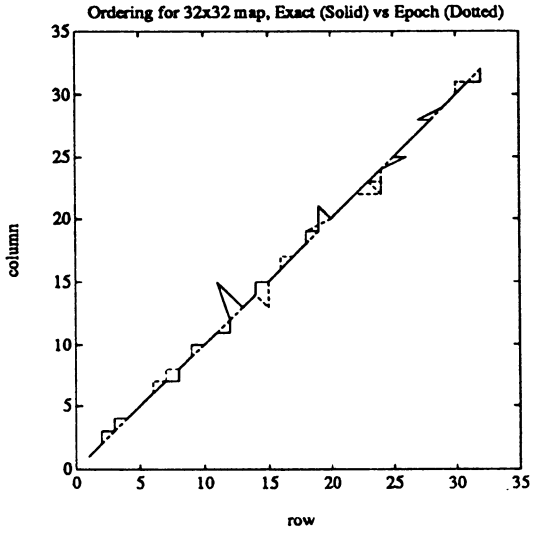
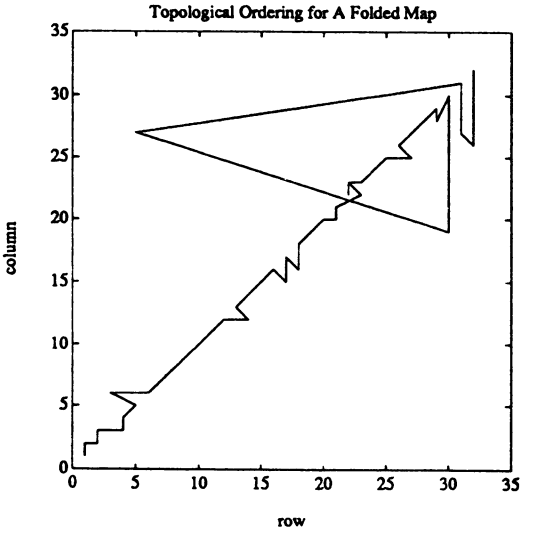


Fig 7 (L)
Fig 8 (R)





MULTIPLEXED, CHARGE-BASED CIRCUITS FOR ANALOG NEURAL SYSTEMS

L.W. Massengill
Space Electronics Group
Department of Electrical Engineering
Vanderbilt University
Nashville, TN 37235

Abstract

A charge-based analog multiplier circuit and a time-division multiplexed architecture for single-chip, multi-layer neural network implementations is presented. The neural cell supports continuous signals and weights; all computations are performed in the analog domain and multiplexing is accomplished by dynamic current injection. Test circuits fabricated with the MOSIS 2.0μm analog process operate as predicted by analysis and simulation, with a multiply cycle time of approximately 30 nsec.

INTRODUCTION:

If one uses the development of mature microelectronic technologies as a precursory model, the extension of artificial neural network models into integrated hardware at VLSI levels will require scalable, modular circuit primitives, high-bandwidth communications, and possibly unique circuit designs not yet seen in traditional digital or analog systems. Many engineering issues presently complicate chip-level integration of connectionist systems [1].

One direction of neural implementation efforts has been the construction of massively-parallel digital systems. These networks exhibit good global communication (interconnect) efficiency and VLSI potential, but are limited by the complexity of the individual computational (multiplier) elements and inherent quantization effects. In contrast, analog VLSI offer simplicity of the neural processing elements, good dynamic range, and continuous signals. However, analog systems are often plagued by interconnection limitations, a less than modular/scalable arrangement, and high power consumption.

The particular long-range implementation goals of our group (VLSI, radiation-tolerant, space systems) presents many system constraints. Massive interlayer connections on a planar chip require multiplexing [2], yet high throughput is essential. The analog synaptic weighting requires a multiplication circuit which is size and power efficient, accurately supports continuous weights and signals, maintains a high degree of noise and degradation immunity, and can be naturally and efficiently multiplexed. Traditional active multiplier circuits, such as Gilbert cells [3] or op-amp gain stages, which are common in analog neural implementations [4], are complex circuits requiring substantial space and power, thus limiting system complexity at the single-chip level. Passive weighting using resistors or single FET devices are usually not modifiable after construction or are highly nonlinear. Charge-based approaches, such as CCD [5] or switched-capacitor [6], hold a great deal of promise for power and size efficiency, but weighting by charge attenuation opens questions of reliability in noisy environments, such as space.

This paper presents a charge-based approach for analog, multi-layer neural systems using charge-injection multiplier circuits and analog multiplexing over distributed communication busses. Standard double-polysilicon, 2 um CMOS processing is utilized in the design.

TRANSDUCER CELL DESIGN:

The basic circuit module is a cell which produces an integral charge proportional to an input signal and a weight, both of which are analog (continuous) quantities. Figure 1 shows a schematic of the multiplier circuit.

This work is supported in part by an internal research grant from the Vanderbilt Univ. Research Council.

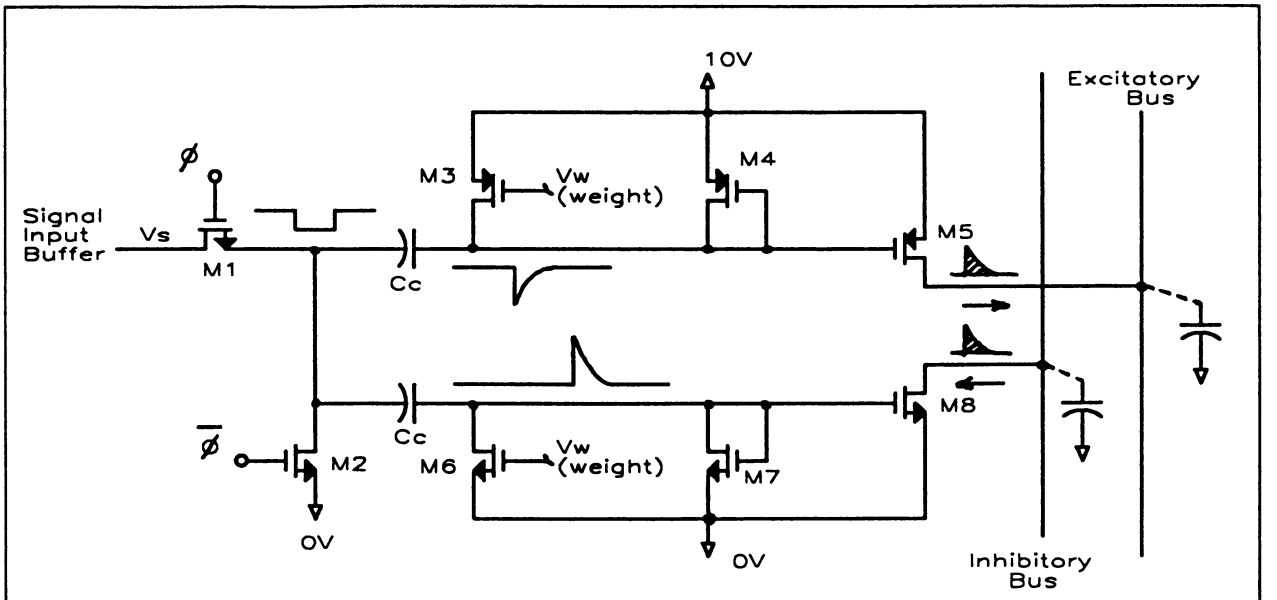


Fig. 1. Schematic diagram of the charge-based multiplier circuit.

During operation, the input signal to a group of these modules, termed a neural broadcast cell, is temporarily stored as a voltage, V_S, in a low-power buffer. Clock signal, ϕ, and devices M1 and M2 act as an input switched capacitor stage which cycles the cathode of coupling capacitors C_C between V_S and ground. The weight voltage appears on the gates of devices M3 and M6; this weight can be stored in a nonvolatile fashion using double-poly floating gate devices for M3 and M6, or can be dynamically refreshed from off-chip circuitry [7]. (We have fabricated both and are evaluating the tradeoffs of nonvolatility versus ease of weight programming.)

The remainder of the circuit generates excitatory and inhibitory exponential pulses controlled by the signal and weight, onto distributed charge accumulation busses shared by all cells. Multiplication is performed using the simple property that the integral charge contained in an exponential current pulse is given by the peak value multiplied by the decay time constant. This circuit is designed so that (to first order) the weight and the input signal independently determine the exponential decay time constant and the initial magnitude of the pulse, respectively. Sourcing and sinking output drivers connected to excitatory and inhibitory busses produce two-quadrant multiplication over the weight range 0 to 10 Volts. More rigorous circuit analysis [8] shows interesting second order nonlinearities, as is subsequently discussed.

By circuit design and constraints on the accumulation bus voltage, the output devices, M5 and M8, are always saturated; because of this, the charge delivered does not depend (to first order) on the changing voltage along the distribution bus. Thus, many of these cells can be active at any one time. Also, since analog switches onto the shared busses are not needed, clock feedthrough from switching transients are minimized. The circuit not only performs the multiply operation on weight and signal, it acts as a simple, pulsed amplifier for high signal-to-noise ratios. However, because this is a dynamic amplifier technique, power consumption is minimized.

ARCHITECTURE:

These basic multiplier circuits are grouped in neural broadcast cells, as shown in Fig. 2a, and time multiplexed over shared analog busses, as depicted in Fig. 2b. Space does not permit a complete discussion of the overall architecture of the analog multiplexed system, but the extension to more cells and layers is straightforward. In the figure, S denotes signals and W denotes interconnect weights. Local control signals generated by a globally-clocked shift register within each neural cell produce the ϕ timing signals of Fig. 1. Lockstep operation produces concurrent multiplication and charge injection by one output of each neural broadcast cell during every clock cycle.

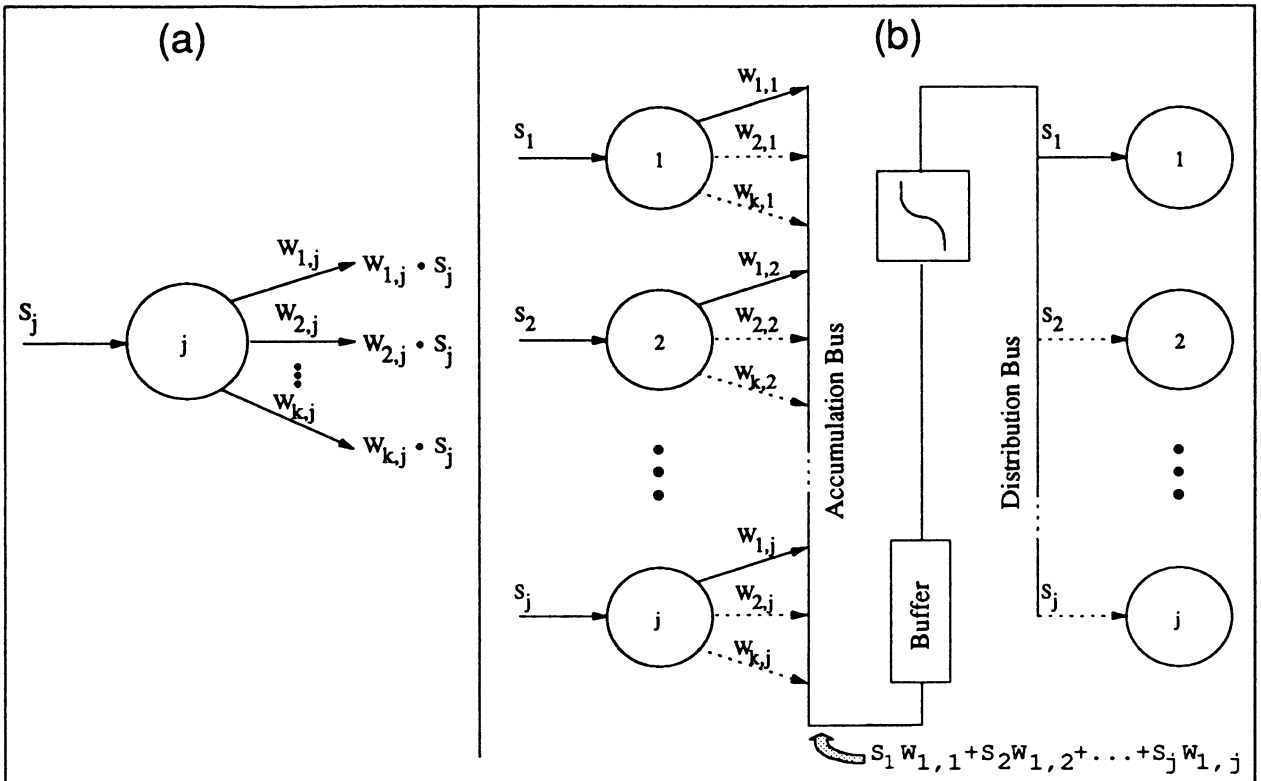


Fig. 2. (a) Operation of the basic neural broadcast cell. (b) Multiplexing and interconnect diagram; solid lines indicate active signal paths during one clock cycle.

During each clock cycle, the parasitic capacitance of the accumulation bus acts to integrate and sum the charge packets delivered by the neural cells. Shared buffer and activation circuits process the sum, perform appropriate offset and scaling, and produce a voltage corresponding to the particular activation function implemented (e.g. a sigmoid function). This output is then distributed to an analog latch for the first neural cell in the N+1 layer. The process is repeated (time multiplexed), loading each neural broadcast cell in the N+1 layer. Since only one set of processing (activation) circuits are used between each layer, considerable cost (size, complexity, power) can be expended where it is needed in these analog subcircuits without severe penalties in system size. In fact, the approach is generic in that any type of activation function can be implemented with minimal modifications.

EXPERIMENTAL RESULTS:

Test bars of several of the subcircuits described have been fabricated using the MOSIS 2μm analog (double polysilicon) process. The circuit of Fig. 1, which was not at all optimized for space in this phase of the work, occupied approximately 40x60 μm. Experimental results of the operation of this basic circuit are shown in Fig. 3. Here we see the injected charge as a function of the input signal for three weight values. Other similar data was acquired, but only three curves are shown for clarity. An interesting natural nonlinearity of the circuit is seen in Fig. 3. This nonlinearity was not unexpected; it is directly predicted by circuit analysis of Fig. 1. The curves were fit with a sigmoid function multiplied by weighting factors, shown by the solid lines. Because of this form, if backpropagation [9] is the learning scheme implemented, a simple modification of the activation temperature of the desired sigmoid function between layers can predistort the input signals so that very linear response is seen over the entire rail-to-rail dynamic range of the circuits [8].

The test circuit delivered the charge packet at the clock edge in approximately 30 nsec. A conservative clocking scheme of 100 nsec cycle time with 50% duty cycle corresponds to 10 MHz operation for the multiplications, summing, and loading of each neural broadcast cell in a layer. Scaling and speed-optimization could increase this figure, as well as pipelining information through the layers. Of course, many outputs are active during any one clock pulse, so throughput depends on the number of broadcast cells in each layer.

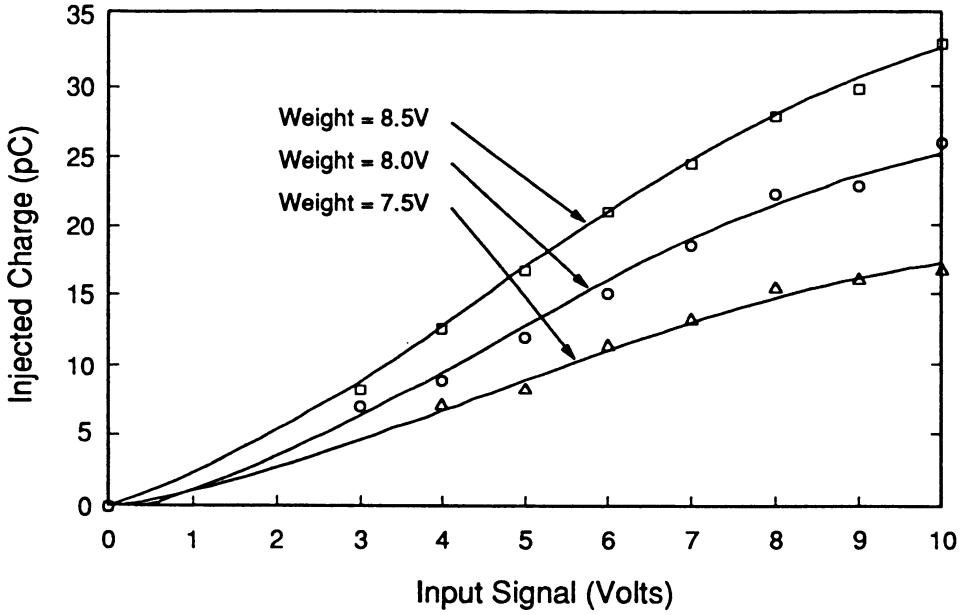


Fig. 3. Experimental results of the neuron weighting operation. Data points show measured data. Solid lines show a weighted sigmoid function.

CONCLUSIONS:

Our results indicate that a charge-based approach may be useful in the multiplexing of analog interconnect paths for planar implementations of neural systems. The circuit described here generates a linear multiplication of a stored weight (either dynamically refreshed or stored in a nonvolatile fashion on a floating gate) and a predistorted input signal, and delivers the results as an amplified current transient of exponential form. Experimental results show proper operation and predictable, controllable current impulses at an accumulation bus. Even though multiplexing necessarily reduces the throughput of any system, we conclude from our results that the distributed, concurrent multiplications of this architecture are efficient and accurate enough to be a useful compromise between total interconnection and the limitations of planar VLSI processing.

REFERENCES:

[1] C. Lau, "Research Challenges and Opportunities in Electronic Neural Systems," *Proceedings of the IEEE First International Conference on Neural Networks*, San Diego, 1987, vol. III, pp. 201-203.

[2] J. Bailey and D. Hammerstrom, "Why VLSI Implementations of Associative VLCNs Require Connection Multiplexing," *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, 1988, vol. II, pp. 173-180.

[3] B. Gilbert, "A Precise Four-quadrant Multiplier with Subnanosecond Response," *IEEE J. Solid-State Circuits*, vol. SC-3, pp. 365-373.

[4] D. Tank and J. Hopfield, "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, pp. 533-541.

[5] J. Sage, K. Thompson, and R. Withers, "An Artificial Neural Network Integrated Circuit Based on MNOS/CCD Principles," *1986 AIP Conference Proceedings 151*, p. 381.

[6] Y. Tsividis and D. Anastassiou, "Switched-Capacitor Neural Networks," *Electronic Letters*, vol. 23, no. 18, pp. 958-959.

[7] S. Eberhardt, T. Duong, and A. Thakoor, "Design of Parallel Hardware Neural Network Systems from Custom Analog VLSI Building Block Chips," *Proceedings of the IEEE International Conference on Neural Networks*, Washington, DC, 1989, vol. II, pp. 183-190.

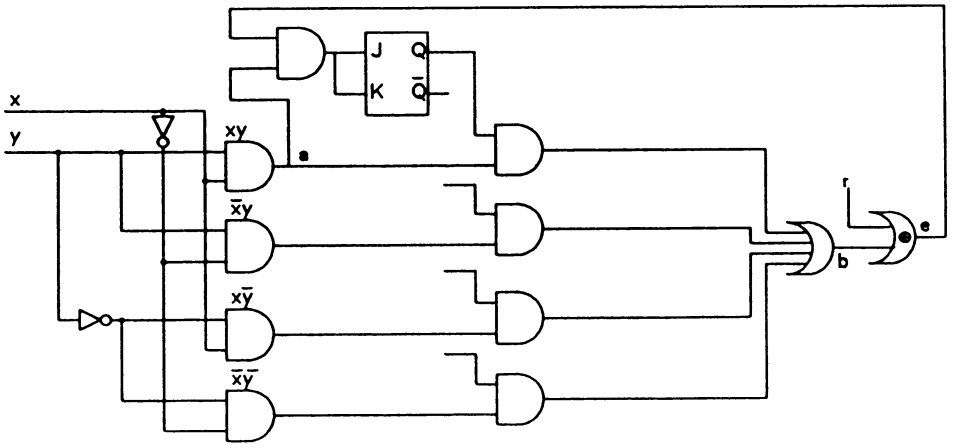
[8] to be published

[9] T. Sejnowski, P. Kienker, and G. Hinton, "Learning Symmetry Groups with Hidden Units: Beyond the Perceptron," *Physica 22D*, pp. 260-275.

LEARNING LOGIC ARRAY. Ethem Alpaydin. Microcomputer Laboratory, Swiss Federal Institute of Technology, Cour 37, 1007 Lausanne Switzerland.

When a logic function with some inputs and output is to be implemented, it is generally hardwired using some, preferrably optimal, number of logic gates. When the specification of the function is modified, the system is halt, the old circuit is removed, and a new circuit is placed in its stead. One approach is to make the gates programmable, thus allowing the modification of the logic function by just modifying the values of some memory elements, i.e., enabling or disabling connections. This second approach is the *soft* approach, as opposed to former *hard* approach. The third approach that I am proposing here is the *learning* approach, where modification of these memory elements are also performed by the system, thus removing the need of a higher-level supervisor completely, who either needs to build the new circuit in the hard approach, or determine and program the connectivities, in the soft approach.

The idea is very simple. The diagram for a two-input function is given. There lies a first layer which is a n to 2^n decoder. The decoder layer gives out unit vectors, i.e., one of the outputs is "on" at a time, others are all "off," thus the actual output of the function can be computed using an OR gate. The connections from the decoder outputs to the actual output are enabled or disabled according to the states of J-K flipflops, each governing one line. The state of the flipflop, as can also be written as a logic function, can be determined by the system itself. A feedback signal is required to inform the system when it commits an error.



x and y are inputs, a is the only "on" output of the decoder, b is system's output, r is the required output, e is the error signal when b and r do not match. Only one of the "and gate-flipflop" connection controller is shown, there are altogether four.

First, the connection from the decoder output a to output b is important only when a is "on." Besides, when there is no error, nothing needs to be modified. But when there is an error, the error should be due to the connection connected to the currently "on" output of the decoder. The state of that connection should be toggled, i.e, disabled if currently enabled, and vice versa. A J–K flipflop when both inputs are "on" acts as a toggle. The error signal can be fed to the system directly, or when the required output is known, can be computed by a XOR gate. No special initialization phase is required.

This work is supported by the Fonds National Suisse de la Recherche Scientifique.

FRAMEWORK FOR DISTRIBUTED ARTIFICIAL NEURAL SYSTEM SIMULATION

Roger S. Barga
Ronald B. Melton
P.O. Box 999
Pacific Northwest Laboratory[1]
Richland, Washington 99352

Abstract

In this paper, we describe a distributed artificial neural system (ANS) simulation environment for research on large-scale ANS networks. The ANSkit, as the environment is called, allows local graphics workstations to utilize ANS simulation engines implemented on remote supercomputers and to interactively display the results of the simulation. The ANSkit design uses extensive data compression and standard network protocols to permit its use over wide area, as well as local area networks. The ANSkit demonstrates an easily used, simulation visualization capability for ANS research in a heterogeneous, distributed computing environment. Further, ANSkit demonstrates how remote supercomputing resources can provide a cost-effective tool for ANS research.

1. Introduction

Research in artificial neural system (ANS) technology is largely based on computer software simulations[1]. This approach offers researchers the opportunity to simulate a variety of ANS paradigms in order to assess and compare their capabilities. However, the simulation of even a moderately large ANS network on a conventional digital computer requires an immense amount of computation and thus, makes this approach both time-consuming and expensive. As the need for research on large-scale ANS networks grows, so must the computing power used for ANS simulation.

Supercomputing systems offer a powerful workbench for ANS simulation and provide an effective near-term solution. With the supercomputer, a researcher can test ideas and optimize ANS network designs faster and more cost-effectively than would be possible by any other conventional means. Although only a few years ago supercomputers were a scarce resource for the general scientific community, the creation of national supercomputer centers have made them accessible through wide area networks[2].

Because of the enormous quantities of data that ANS simulations can produce, graphical display capabilities are a major concern to ANS researchers. In certain cases, graphics have led to a better understanding of the computation taking place in ANS networks [2]. By translating data into images, computer graphics provide an effective way to reveal information that otherwise would remain buried in an avalanche of numbers. A computer system well-suited for ANS research would use a dedicated high-performance computer for ANS simulation that is tightly coupled to a dedicated graphics workstation for network visualization. However, this is not a cost-effective approach when the simulation engine is a supercomputer.

Our approach in satisfying these somewhat conflicting resource constraints is to distribute the processing of ANS simulation across computer systems. A distributed ANS simulation environment known as ANSkit (Artificial Neural System toolkit) was designed at Pacific Northwest Laboratory (PNL) to make use of both remote supercomputing resources and local graphics workstations. This approach allows us to use remote supercomputing resources to achieve appreciably higher-performance ANS simulation. Visualization support and ANS simulation control is incorporated in the ANS simulator implemented on local graphics workstations.

The design of the ANSkit system is intended to integrate heterogeneous computer systems through standard network protocols, while avoiding any architectural dependencies. By using Berkeley sockets [3] for interprocess communication (IPC), the ANSkit system can operate over the many networks that make up the TCP/IP Internet. This offers an environment which is conveniently extensible, both in terms of scale (to other supercomputer systems and graphics workstations on the network) and for future improvements (to incorporate the benefits of new ANS paradigms and visualization techniques). It is hoped ANSkit will provide an example of how remote supercomputing resources can be used to provide a cost-effective tool for research on large-scale ANS networks.

The focus of this paper is to describe the physical architecture of the ANSkit environment. A brief synopsis of the network environment at PNL will be presented in order to provide some perspective for the system architecture in context of the overall computer network.

2. Previous Work

The utility of supercomputers for ANS simulation has been recognized by many researchers. Several systems, some of them very effective, have been proposed and/or implemented. For example, the multilayered perceptron network has been implemented on the massively parallel Connection Machine [4], the Warp[3] systolic array computer [5], and the Cray X-MP vector processor [6]. Most of this work has focused on the design and implementation of a particular ANS algorithm on a supercomputer. In contrast, our work presents a framework for a distributed ANS simulation environment capable of incorporating multiple ANS algorithms. Advances in implementing ANS algorithms on supercomputers and our distributed ANS simulation environment are complementary. Our work proposes a methodology so that ANS algorithms running on remote supercomputers are accessible through the internet to local graphics workstations.

3. ANSkit Design Principles

Our motivation, in designing and implementing ANSkit, was to facilitate research with large-scale ANS networks. It was not intended as a simulation tool for any particular ANS network model. Instead, our goal was to develop a distributed ANS simulation environment that could easily incorporate additional ANS models and computing hardware as our research progressed. This goal led to the set of key design principles listed below:

- **Natural distribution of processes.** Each process of the simulation system should be executed on a computer system that is natural for that process. For example, ANS learning mode simulation is naturally done on a high-performance floating-point architecture, while the rendering of the ANS network under study is most naturally done on a high-resolution graphics workstation.

---

1 Operated for the U.S. Department of Energy by Battelle Memorial Institute under contract DE-AC06-76RLO 1830.

2 The sites that supply supercomputing to the general scientific community are: National Magnetic Fusion Energy Center (NMFECC), National Center for Atmospheric Research (NCAR), and the National Science Foundation supercomputer centers (SCSC, PSC, JvNC, NCSA). Access to these supercomputing centers is via wide area networks.

3 Warp is a servicemark of Carnegie-Mellon University.

simulator on a supercomputer produces an appreciable performance difference. Therefore, the ANS simulation modules for learning were implemented to perform operations on vectors of data which represent the ANS network and training data. As described in more detail elsewhere [6], the resulting modules are both highly vectorizable on supercomputers and quite efficient on scalar processors. Third, the learning process must provide the user interface process with the ANS network state information in near-real time. From a remote supercomputer, the learning process must transmit both the network connection weights and learning state information over the internet to the user interface process.

To reduce both the amount of data transferred and the transfer time, the data is compressed by limiting the significant digits and removing redundancy. To compress the connection weights we use what amounts to a sequential quantization entropy reduction technique. This is similar in principle to differential pulse code modulation [7]. The small loss of accuracy in the connection weight values does not adversely affect the quality of the graphics display of the ANS network. When the ANS learning process is complete, the final connection weights are sent over the network without compressing the data, to preserve the accuracy of the values.

The ANSkit learning process can be used effectively over the internet since it does not send out huge amounts of data to the user interface process on the local workstation. On a Cray X/MP, the value of one connection weight requires 64 bits of storage. Each connection strength is data compressed prior to being transmitted with 12:1 -15:1 compression. With the 50Mbit/sec HYPERchannel, the data transfer rate is approximately 11 million connection values per second. At 1.5Mbit/sec, the data rate is considerably slower going across the microwave connection to the Ethernet. However, these data transfer rates are sufficiently fast to support the near-real time display of ANS network information on the local workstation. To illustrate, in a *large* ANS network with 1,000 processing elements and 160,000 connections, the connection weight values and learning trace information amounts to approximately 83 Kbytes of compressed data that must be transmitted for each graphics display update. This takes slightly over one-hundredth of a second at 50 Mbits/sec, and slightly less than one-half a second at 1.5Mbit/sec.

5.2. Simulation User Interface Modules

The simulation user interface modules are responsible for interpreting user commands, and executing the ANS simulation on either a remote supercomputer or a local graphics workstation. The user interface is designed to present the researcher with an interactive interface to the ANS simulation, regardless of the machine on which the simulation is being run. It also provides support for creating and archiving ANS networks and training sets, and furnishes the graphics interface module with the data required to create a visual display of the ANS network.

ANSkit has a graphical interface which allows the user to design an ANS network interactively, using a mouse and pull-down menus to create and edit information which corresponds to the network's structure. The output of this process is two data files. One specifies the ANS network configuration and model parameters. The other contains the training data for ANS learning. A validation module then performs error and validity tests on the user specified network. The resulting ANS network is displayed on the graphics display by the graphics interface module. In a few seconds, the researcher can define and build an ANS network, and view a graphical display of the resulting ANS network.

From the user interface, the researcher can elect to run the ANS network on the local workstation to conduct experiments (i.e., classification performance, error tolerance, etc.) with the network under study, or execute the learning process. ANSkit has provisions for controlling learning mode simulation runs on remote machines. Once the user has created the ANS network and selected the training data, the user opens an authorized session on the supercomputer through a terminal emulator on the workstation. By selecting an option from the user interface menu, the user begins the learning mode process, which in turn establishes the interprocess communication path on the network. The user interface process then transmits the data files which contain the ANS network specification and the training data to the ANS learning mode process. At this point the learning process on the supercomputer creates the ANS network from the specifications file and begins the learning process. Optionally, for small ANS networks, the learning mode simulation can be run on the local graphics workstation.

When the learning process is running on a remote supercomputer, it must transmit a file containing the numeric representation of the ANS network (connection weights and processing elements values) back to the local workstation. The user interface process decompresses this data into the standard vector representation of the ANS network, and provides this data to the graphics interface module.

Once the learning process is complete, a file containing the final connection weights is transmitted to the user interface. From the user interface, the researcher can display the ANS network and perform experiments, to evaluate the results of the training. The ANS network definition can also be saved and reloaded from files in text formats. An ANS network definition includes the network connection topology, underlying data structures, and optionally, selected state information (such as connection weight values and processing element states).

5.3. Graphics Interface Module

The graphics interface module can render graphics images from the vector representation of the ANS network and display the network in near-real time. This allows the researcher to view ANS network information as the computations are taking place, independent of the process running the ANS simulation. The graphics interface module presents this interactive graphics display of the ANS network during both the learning and execution modes of simulation.

Graphics primitives are rendered by scanning the numeric representation of the ANS network stored by the user interface process. Each ANS network processing element and connection is displayed as a separate icon whose size, shape, or shading varies with the current value of that processing element or connection. As the simulation runs, the icons are updated to reflect changing values, providing a continuously changing view of what the network is doing.

Supporting the graphics interface module are two graphics representation algorithms that are general enough to be used for displaying any ANS network model. The first is an algorithm to display the ANS connection weights and processing element values as Hinton diagrams. The user may select which processing elements and connections are displayed, and may select the rate at which the connection weight display is updated during learning mode simulation. The second is an algorithm to display the ANS network connection weights and processing element values as network graph diagrams. This algorithm provides a mechanism for displaying the complete network topology of the network under study. Here, the user may select the rate at which the connection weight and processing element state display is updated.

Other graphical operations on the ANS network display include zoom and pan on the processing element and connection icons, and the ability to trace the value of any icon over time. The result is an interactive graphics display that allows the researcher to graphically view and examine the ANS network under study at any time without disturbing the simulation.

6. Future Directions and Enhancements

Though the ANSkit environment is still under development, it is being used by researchers to develop applications of ANS network models to real-world problems. For example, we have used the ANSkit environment to develop a multilayered perceptron network for a matched filtering problem (Barga and Melton [8]).

Future plans for enhancing the ANSkit environment include the incorporation of additional ANS algorithm implementations and computers systems, and enhancements to the user interface. As illustrated in Figure 3, several ANS algorithms have been implemented on computer systems throughout the PNL computer network. Currently, only the multilayered perceptron (MLP) network is fully integrated into the ANSkit environment. Our ultimate objective is to integrate all of the ANS simulation algorithms into the ANSkit environment. In addition, we will utilize other high-speed computer systems for ANS algorithm simulation, such as the Convex (refer to figure 2). The ANSkit environment will continue to grow, incorporating new ANS simulations and computers systems, in response to resources required to support our ANS research efforts.

- **Standard network interface.** The system must utilize standard network protocols and avoid architectural dependencies. This is to permit the extension of the system to computing resources that are accessible over the internet and which run TCP/IP.

- **Vectorization.** The learning mode process should be able to exploit vectorization and pipelining. Calculations that are similar should be done together.

- **Common representation.** Most of the simulation system should work with a single representation of the ANS network. To accomplish this, we represent the ANS network and training data as vectors. All of the ANS simulation calculations and visualization algorithms are performed exclusively on these vector representations. This is not only an efficient method of transmitting ANS network information over the internet, it also facilitates the vectorization of ANS simulation.

- **Large ANS models.** The system should be designed to simulate a large number of ANS network processing elements and be able to process large training sets.

- **Near-real time display of the ANS network.** The graphics display of the ANS network must be highly efficient, as we expect the display to be updated in near-real time.

4. The Network Environment

At PNL, the three principal networks associated with the ANSkit system are Ethernet, HYPERchannel, and the long-haul network NSFnet. The local area network based on Ethernet links all the computers with the exception of the Cray X-MP computer system. The Cray X-MP is linked to the network by HYPERchannel. Both of these local-access networks connect computers located within PNL. These laboratory-wide connections are made with microwave technology, repeaters, and bridge connections for the Ethernet and HYPERchannel networks, as indicated in Figure 1.

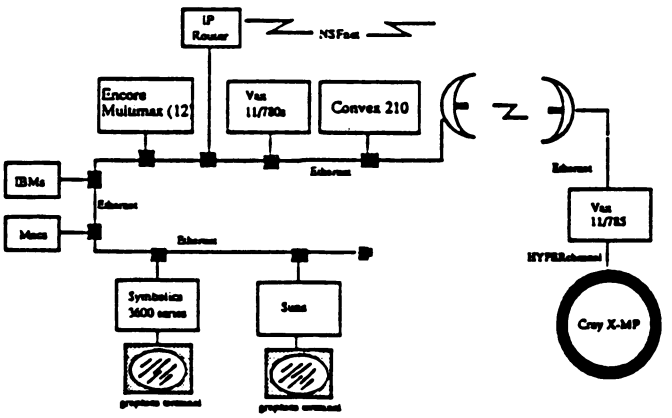


Figure 1. Partial hardware configuration of the PNL computing facilities.

All the computers shown in Figure 1 run the TCP/IP communication protocol. In addition, Berkeley-style networking commands are supported on many of the larger systems, which further facilitates the distribution of application systems. The use of TCP/IP provides remote users transparent access to the computing resources on the network, including the Cray X-MP, through the Ethernet and HYPERchannel gateways.

The internet accommodates multiple, diverse computing systems, network technologies, and operating systems while providing a uniform set of conventions for usage. ANSkit is designed to take advantage of this extensive technology. The success of distributing the ANSkit system across the internet is that ANSkit "sees" and uses only one, uniform interface: TCP/IP and Berkeley sockets. The result is a highly extensible system that can easily incorporate any additional computer systems that run TCP/IP with the Berkeley socket library.

5. The ANSkit Distributed ANS Simulation Environment

ANSkit was designed to take advantage of the heterogeneous computing resources described above. The ANSkit system design involves two processes: the user interface process and the ANS learning mode process, as shown in Figure 2. The user interface process is designed to run on a local graphics workstation, while the ANS learning mode process is designed to run either locally on the workstation or on a remote supercomputer, as the user sees appropriate. These processes communicate over the Ethernet or HYPERchannel networks using TCP/IP protocols and the Berkeley socket library. The user interface process controls the ANS simulation and graphically displays the ANS network under study. The learning mode process simulates the ANS learning mode and sends the network learning state information over the internet to the user interface process. By remotely simulating the ANS network on the supercomputer, numerical data instead of image data can be transmitted, reducing the bandwidth requirements for interactive simulation. The user interface process then displays the state of ANS network processing elements and connections on the local workstation, in near-real time, independent of the learning process running on the supercomputer. The user interface process is designed to simulate both ANS execution and learning modes. This is useful for executing the learning procedure for small ANS networks and for experimenting on trained ANS networks.

There are 3 main modules in ANSkit: the simulation engine for ANS network learning mode; the ANS simulation user interface; and finally the ANS network graphics display. These software modules are illustrated in figure 2a. As shown in Figure 2b, the modules can be connected procedurally on a local graphics workstation, or, as shown in Figure 2c, by interprocess communication between a local graphics workstation and a remote supercomputer.

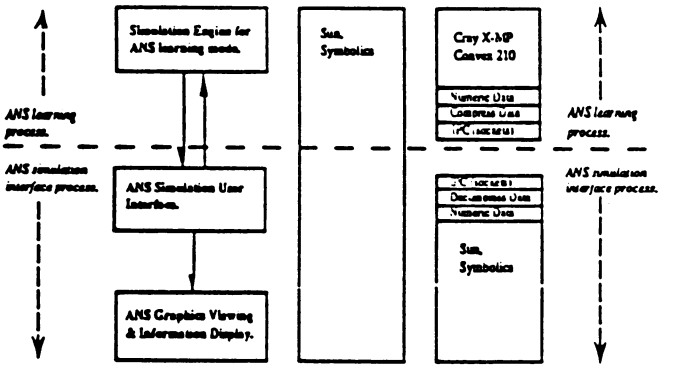


Figure 2. The distributed architecture of the ANSkit system. The ANSkit software modules and interfaces are illustrated in Figure 2a. The modules may be connected procedurally on a local workstation or by interprocess communication between a local workstation and a supercomputer, as shown in Figures 2b and 2c, respectively. The partitioning of the modules is specified by the user at runtime.

This is a complementary combination of technologies. Each machine is permitted to do what it does best (interactive graphics vs. intensive computation) and utilizes supercomputing resources available over wide-area networks. Most importantly, the researcher is presented with an interactive simulation environment for investigating large-scale ANS networks. The ANSkit software modules are discussed in more detail in the following sections.

5.1. Simulation Engine for ANS Learning

We had several goals related to the design of the modules for the ANS learning mode process. First, the software had to be able to run on both supercomputers and workstations. The researcher could then elect to run the ANS learning mode simulation on the local workstation or the supercomputer, depending on the complexity of the ANS network and training set data. This meant we had to avoid architectural dependencies. Second, whenever possible we wanted to take advantage of the fine-grained parallelism afforded by the pipelined vector processors found on supercomputers at our lab. This way, running the ANS learning

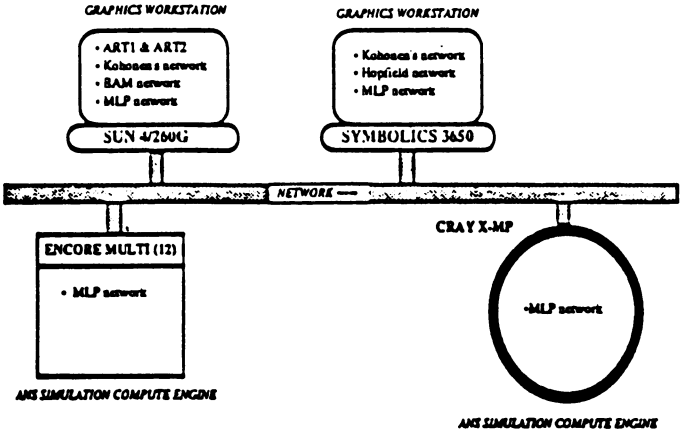


Figure 3. A simplified illustration of the PNL computer network depicting the ANS simulation algorithms implemented on computers systems throughout the network. A more detailed illustration of the PNL computer network is presented in Figure 1.

Additional user interface features are planned to present the user with a transparent and seamless interface to simulations running on remote computer systems. The most important is a programmatic interface to the ANS simulation programs that run on the remote computers. A programmatic interface would provide the local user interface with interactive control of the remote simulation, instead of operating in the background. Through a programmatic interface, the user could, for example, update the simulation algorithm parameters in response to an undesirable state depicted in the graphics display of the network. Coupling the user interface process to the remote simulation process through a programmatic interface will provide more user control and interaction with the simulation.

7. Conclusions

We have presented the framework for a distributed simulation environment for ANS research. The methodology lends itself nicely to distributed processing over wide area networks and utilizes both remote supercomputing resources and local graphics workstations. The utilization of standard network protocols and the avoidance of architectural dependencies permits the extension of the environment to any of the supercomputing resources available through the internet.

Our specific accomplishments include:

- The development of a methodology for distributing ANS simulation computations. This methodology is based on the utilization of local graphics workstations and remote supercomputer resources accessible over wide area networks.
- The development of an operational distributed ANS simulation environment for research with respect to the methodology.

Our experience with ANSkit demonstrates that distributed ANS simulation can be quite successful in wide area networks, processing on both remote supercomputers and local graphics workstations. It is our view that the development effort expended to implement the ANSkit environment on the hardware described is well justified by the increase gained in performance. We feel that the use of supercomputers is essential for the simulation of large-scale ANS networks to be performed in a feasible amount of time. For a more complete treatment on the use of supercomputers for ANS simulation (with a detailed example), see Barga [6].

Acknowledgments

The authors thank Nancy E. Miller and D. Mike DeVaney of Pacific Northwest Laboratory for their constructive comments in preparation of this manuscript.

References

[1] DARPA Neural Network Study (1988). Available from AFCEA International Press. 4400 Fair Lakes Court, Fairfax, VA. 22033. (703) 631-6190.

[2] Gorman, P., & T.J. Sejnowski (1988). Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets. Neural Networks Vol. 1, No. 1, 75-90.

[3] Comer, D. Internetworking With TCP/IP: Principles, Protocols, and Architecture. Prentice Hall, 1988.

[4] Belloch, G., Rosenburg C.R. (1987) Network learning on the Connection Machine. Proceedings of the Tenth International Joint Conference on Artificial Intelligence, Milan, Italy. Vol. 1, No. 323-326.

[5] Pomerleau, D.A., et. al (1988). Neural Network Simulation at Warp Speed: How We Got 17 Million Connections Per Second. Proceedings of the 1988 IEEE International Conference on Neural Networks. Vol. 2, 143-150.

[6] Barga, R.S. (1989). Supercomputer Support for Artificial Neural System (ANS) Research. Technical Report PNL-SA-16681. Pacific Northwest Laboratory, Richland, WA. 99352.

[7] Lynch, T. Data Compression, Van Nostrand Reinhold, 1985.

[8] Barga, R.S., & R.B. Melton (1988). Artificial Neurocomputing for Data Primitive Extraction. Technical Report BN-SA-2793. Presented at the *4th Annual Aerospace Applications of Artificial Intelligence Conference*, Dayton, Ohio.

COMPUTER AIDED RADIOLOGIC DIAGNOSIS USING NEURAL NETWORKS

John M. Boone, Ph.D., George W. Gross, M.D.
and Gary S. Shaber, M.D.

Department of Radiology
Thomas Jefferson University
111 South 11th Street
Philadelphia, PA 19107

ABSTRACT

The interpretation of radiological images is a two step process requiring 1) the identification of abnormal anatomical structures as they appear on the radiographic image, and 2) the interpretation of any abnormal findings into a list of plausible diagnoses. The first task involves pattern recognition, where the second step in this process is a cognitive procedure. We have evaluated the performance of neural networks in both of these steps, independently, and found that in both cases neural networks were capable of performance comparable to radiologists.

INTRODUCTION

Medical diagnosis is an endeavor requiring the assimilation of complex, often sparsely related and sometimes conflicting data. Consistent and accurate medical diagnosis is a prerequisite to efficacious medical treatment. The quality of medical diagnosis, however, is very much related to the training and experience of the medical diagnosticians that the patient encounters in his or her clinical evaluation. The aim of computer aided diagnosis (CAD) is not to replace, but rather to aid and assist in the diagnostic work-up of patients. The hypothesis surrounding CAD is that computer assisted diagnostic tools, if they can be implemented, tested and demonstrated to be reasonably accurate, will introduce improved consistency and accuracy in the diagnostic phase of patient care.

Radiologists sit at the front line of the battery of diagnostic tests in medical practice, and often the "x-ray" (radiograph) is the first diagnostic test ordered in the work-up of a patient. Thus, the radiologist works in a realm where the radiographic image is the primary, and often the only, source of diagnostic information. The sparsity of clinical information is only compounded by the enormity of the radiographic information. For example, chest radiography using the anterioposterior and lateral views (two images) comprises the analog equivalent of approximately 32 megabytes of data.

Radiologists first read the radiographic images, and compose a mental list of abnormal "findings", which may serve as clues in the diagnosis. This process is clearly one of pattern recognition. Due to the constraints imposed by minimizing the radiation exposure to the patient, the signal to noise ratio of radiographic images can be very low. We examined the performance of a two layer perceptron in evaluating simulated, very simple but noisy 25 pixel images and compared the network's performance

with two human observers given the same task. The second step in radiologic diagnosis is to interpret relevant radiographic findings, comprising a list of possible diagnosis, which in turn is used in guiding further diagnostic work-up. We evaluated this in the specific application of pediatric chest image diagnosis. The two phases in this process will be discussed sequentially.

STEP 1: PATTERN RECOGNITION IN A NOISY IMAGE

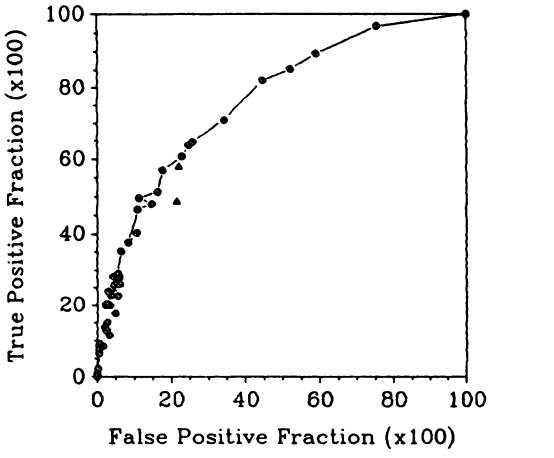
One hundred 25-pixel (5 x 5) images were generated with Gaussian noise, and in 50% of these a signal was added. The amplitude of the signal with respect to the noisy background was governed by the signal to noise ratio (SNR), a dependent variable in this study. The signal was a 3 x 3 square added to the center of the 5 x 5 image. The 100 images served as a training set for the feedforward 2 layer network, using 25 input nodes (1 for each pixel), 5 hidden nodes and 1 output node. The image data in each case was normalized to the interval (0,1) prior to being input to the network. The target nodes were set to 0 if no signal was present, and to 1 if a signal was present. The learning algorithm used was a variant of the generalized delta rule. The network solution converged with 100% accuracy in under 3000 iterations on the initial training set generated at a SNR of 5. The same noise ensemble was used to generate training sets at lower SNRs, and the network (along with its coefficients) was trained using progressively lower SNRs (5, 3, 2.5, 2 to 1.75). This "sensitizing" technique was used because the network was unable to converge on the training set when initially generated at a SNR of 2.

The network was tested on test sets of 1000 images (different from the training set), generated as above, at various SNRs. The numbers of true positives (tp), true negatives (tn), false positives (fp) and false negatives (fn) were scored, and the true positive fraction [tp/(tp+fn)] and the false positive fraction [fp/(fp+tn)] were calculated. A threshold could be applied and varied to the network output (which is continuous on the interval (0,1)), such that network outputs less than the threshold were considered as "no signal", and output above the threshold value meant that the network is calling the signal as present. The tp and fp fractions were calculated as a function of threshold value and are plotted against each other in Figure 1. These plots are known as Receiver Operating Characteristic (ROC) curves. Two human observers (one radiologist and one physicist) were tested using 300 images each, and their results are also shown on Figure 1. Because the human observers did not vary their decision threshold, only one point of the ROC curve was determined for each human observer.

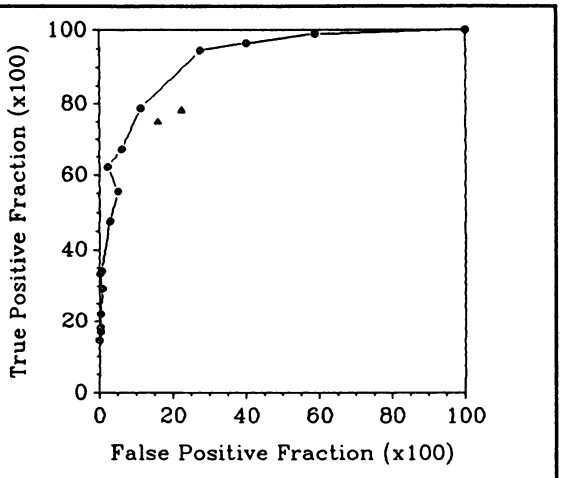
On ROC curves, a perfect observer would be represented by a plot climbing the ordinate to 100% and then spanning the abscissa at the top of the graph. A perfect guesser would be represented by the diagonal line from lower left to upper right. The ROC curves shown here illustrate that the neural network was able to perform comparable or better to human observers, at least in this

very basic visual perception task. This result suggests that continued research using neural networks in radiological pattern recognition is warranted.

FIGURE 1



ROC curve for a testing set with a SNR=0.5. Triangles are the human observers



ROC curve for a testing set with a SNR=1. Triangles are the human observers.

STEP 2: COMPUTER AIDED DIAGNOSES USING NEURAL NETWORKS

The second step that a radiologist performs is to convert a mental list of radiographic findings to a list of plausible diagnoses. This step encompasses cognitive reasoning based on the radiologist's medical training and experience. We developed a checklist with 50 possible choices for radiographic findings, and 23 possible diagnoses, pertinent to newborn chest radiographs. A training set of 77 images, all from patients under 48 hours of age, was read by a pediatric radiologist (RAD A). A separate set of 103 images, to be used for testing network performance, was read by two pediatric radiologists (RAD A and RAD B) in independent sessions. The 50 possible findings could be represented by 21 input nodes. Eleven of the 23 possible diagnoses did not occur in over 2 patients, and so only the 12 remaining diagnoses were used, in corresponding to the 12 output nodes. Table 1 lists the findings and diagnoses used in this study. A two layer feedforward perceptron (one hidden layer, 15 hidden nodes) was used, and again a variant of the generalized delta rule was used to train the network on the training data. After 20,000 iterations the network was able to correctly identify 150 of the 190 positive diagnosis (79%) and 727 of the 736 negative diagnoses (99%) made by the mentor radiologist, RAD A. The agreement in the test results between the two radiologists, between the network and either radiologist, and between two random guessers is shown in Figure 2. Positive agreement is defined as tp/(tp+fn+fp), negative agreement is

tn/(tn+fn+fp) and total agreement is (tp+tn)/(tn+tp+fn+fp). The random guessers were armed with only the prevalence of the various diagnoses as determined from the training data.

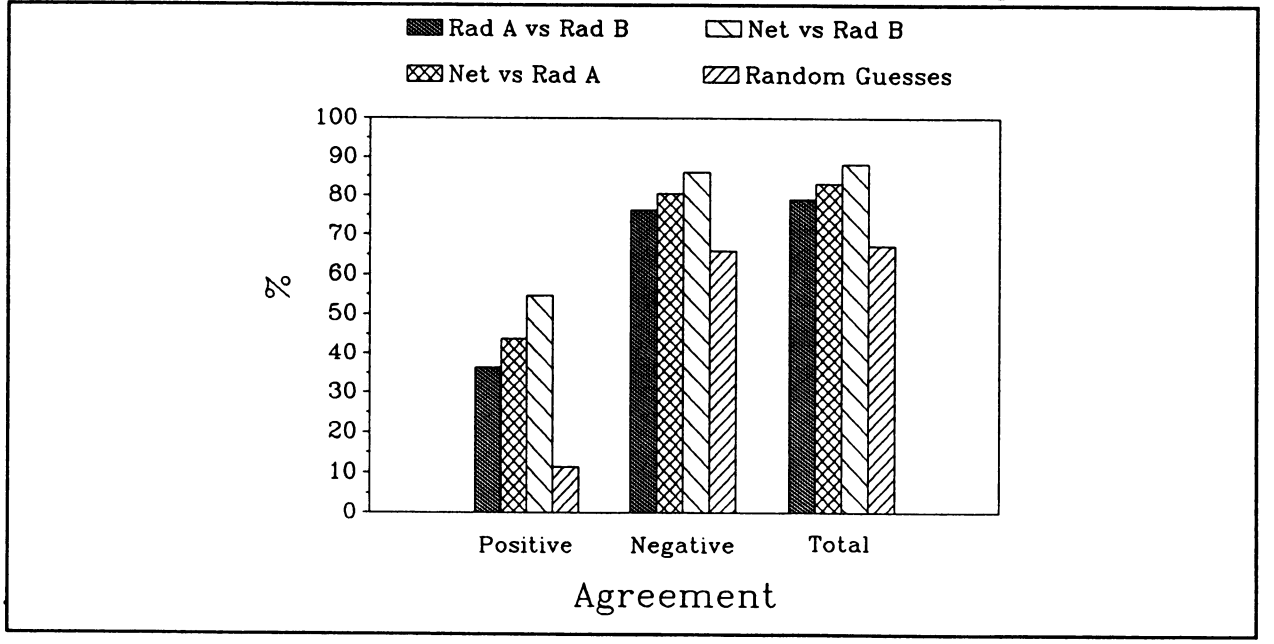
Table 1: The findings and diagnoses used in this study

FINDINGS	DIAGNOSES
Cardiomegaly	Amniotic Fluid Aspiration
Pulmonary Vascular Density	Neonatal Pneumonia
Mediastinal Shift	Hyaline Membrane Disease
Pleural Effusion	Meconium Aspiration Syndrome
Pneumothorax	Acute Pulmonary Hemorrhage
Pneumomediastinum	Atelectasis
Aeration-degree	Transient Tachypuca of Newborn
Aeration-spatial	Cyanotic Congenital Heart Disease
Pulmonary Infiltrate+8 characteristics	Acyanotic Congenital Heart Disease
Pulmonary Infiltrate Distribution	Congestive Heart Failure
Bowel Gas	Pulmonary Hypoplasia
Ascites	Normal Chest
Hepatomegaly	

SUMMARY

We have applied neurocomputing towards computer aided diagnosis in radiological diagnosis, and found that neural networks show early promise in both steps of image interpretation. The results presented in this study provide a basis and encouragement for further investigation.

Figure 2: Comparisons of performance between diagnosticians



Pulse Coding Hardware Neurons that Learn Boolean Functions

Sabine Canditt and Rolf Eckmiller *)
Div. Biocybernetics, Heinrich-Heine-Universität Düsseldorf, FRG
Tel: (211) 311-4540; e-mail: ECKMILLE@DDORUD81.BITNET

Abstract

A small network of electronic Pulse Coding Neurons (PCNs) is presented having features similar to those of its biological counterpart. Information is encoded in the instantaneous impulse rate IR(t). It functions in a completely analog and asynchronous way, with the synaptic weights being stored digitally (8 bit). Neural net topology can be selected flexibly. A personal computer is used for monitoring functions.

It is shown that the PCN network can learn logical functions including XOR when provided with a corresponding teach signal. The learning rule is embedded in the neural net topology rather than in a learning algorithm. To this purpose, some of the PCNs function as "learning neurons", having special contacts on synapses (presumably similar to pre-synaptic synapses) to modify the weights.

Introduction

There have been several efforts to build fast VLSI hardware emulators for Artificial Neural Networks (ANNs,[8]). Most of them are based on simple Processing Elements (PEs) that show only very few similarities with biological neurons. One of the main differences between these PEs and Pulse Coding Neurons (PCNs) is that PCNs use pulse trains as information carriers, whose temporal structure offers a wide range of free parameters. The PCN concept emphasizes the information processing in biological neural systems ([4], [7], [10]).

The literature describes several attempts to combine pulse coding schemes with VLSI technology. ([1], [2], [3], [5], [6], [9], [11]). Most of them stress not only biological relevance but technical advantages like small size, low power dissipation, and ease of design. Alterable synaptic weighting is usually not possible or accomplished by either gating incoming pulse streams ([5]) or modulating individual pulse widths ([2], [5]). However, in our approach, synapses weight incoming pulses by modulating their amplitude only without changing any temporal parameters.

Concept

The PCN network consists of two different hardware modules, neurons and synapses. The modules (built with discrete electronic components) can be flexibly combined to create various neural net topologies. A Personal Computer (PC) is used to initialize the synaptic weights, provide network- and teach-inputs for learning, and monitor the weights during learning.

Fig. 1 shows a block diagram of our electronic PCN. Rectangular pulses of unity amplitude (5 V, TTL level) and a duration of 1 ms represent action potentials (APs). Synaptic weights are digitally stored in 8 bit counters and A/D-converted. An incoming AP closes a switch and creates a pulse of 1ms and an amplitude corresponding to the stored weight. Weighted pulses of an arbitrary number of synapses are connected to a common low-pass filter. The filter output is the linear superposition of the single AP responses and represents the membrane potential (U_M). A comparator triggers a pulse generator that generates a new AP whenever U_M reaches the threshold potential

*) supported in part by a BMFT-Grant to R. Eckmiller

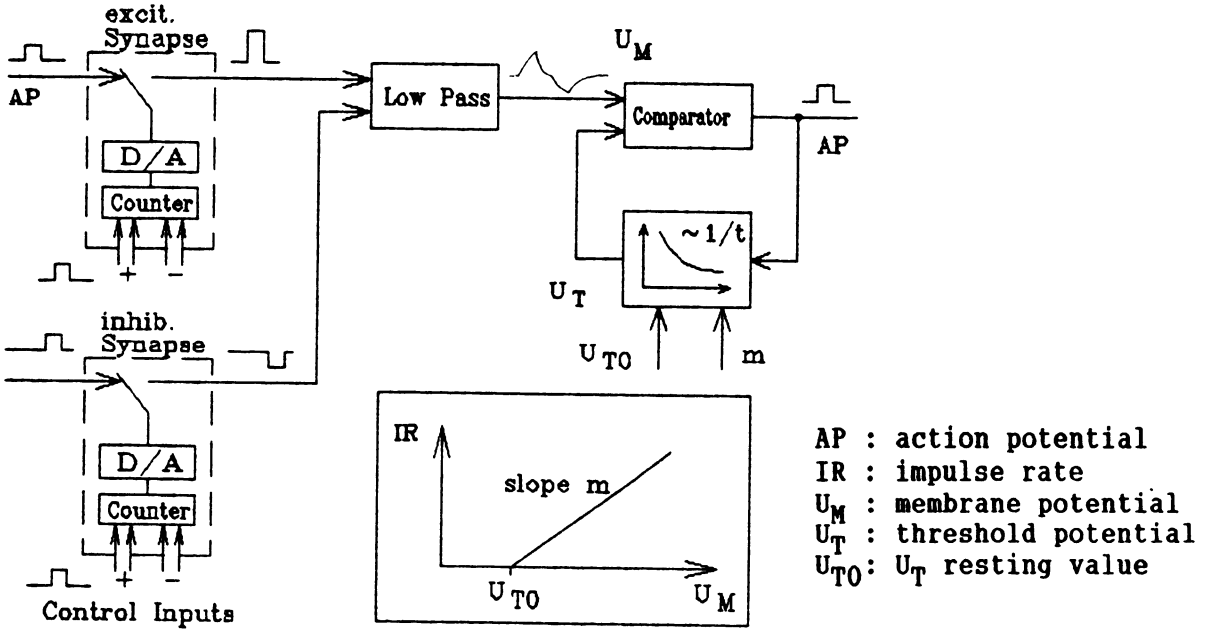


Fig. 1: Block diagram of Pulse Coding Neuron (PCN)

(U_T). To simulate the refractory period, the AP at the output drives U_T to its maximum value (usually the supply voltage). U_T then decays towards the stationary threshold U_{TO} following a hyperbolic time function

$$U_T(t) = 1 / (m \cdot t) + U_{TO}$$

(see Fig. 2). Consequently, the instantaneous impulse rate IR(t) is proportional to the suprathreshold membrane potential U_M (t) (see: inset Fig. 1).

As the synaptic weights are stored in counters (8 bit resolution), they can only change by +/- 1 bit. To effect weight changes, each synapse has two control inputs (presynaptic synapses) in order to increase (+) or decrease (-) weights by means of arriving APs. A weight will be changed only if the AP at the control input coincides at least partially with another AP at the regular synaptic input.

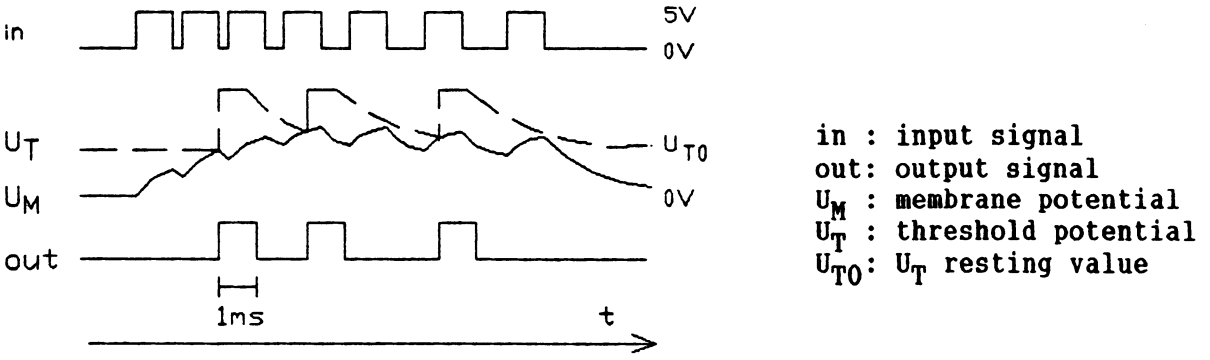


Fig. 2: Time course of PCN signals while processing a burst of APs

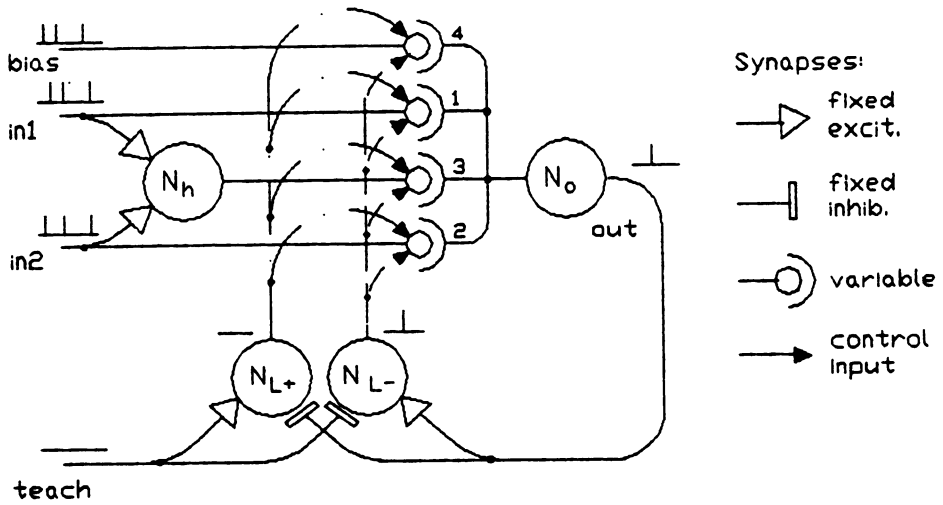


Fig. 3: Network that learns Boolean functions

in1, in2, bias, teach	: network input signals
out	: network output signal
N_h	: hidden neuron
N_o	: output neuron
N_{L+}, N_{L-}	: learning neurons

Application

A neural network of PCNs, which learns Boolean functions of two variables (e. g. XOR), is shown in Fig. 3. The logical values (1, 0) are represented by the following coding scheme:

"1" -> "one ore more APs (burst)",

"0" -> "no AP".

The PC generates 2 inputs (**in1** and **in2**), an additional **bias** input, and a **teach** signal as desired output. All the signals exhibit bursts of the same type (shown at the top of Fig. 2) to denote "1". The "hidden unit" N_h fires whenever both **in1** and **in2** are active. The only changeable weights in this network are those of the synapses 1, 2, 3, and 4 of output unit N_o.

N_{L+} and N_{L-} are "learning neurons" that "compare" **teach** and output signal **out** generated by N_o, detect errors, and change the weights. N_{L+} fires when **teach** = 1 and **out** = 0. Its output is directly connected with one of the "+" control inputs of synapses 1, 2, 3, and 4. N_{L-} operates respectively for the "-" control input.

In the following experiment to learn XOR, the four possible input combinations together with **bias** and **teach** were presented repeatedly with a pause of 10 ms between them. **Teach** was "1" when either **in1** or **in2** were "1" (XOR). Fig. 4 illustrates synaptic weight changes while the number of learning cycles increases. Initial weights were set to +1. Learning curves were monitored until no further error could be detected for at least 200 more learning cycles. The experiment was repeated with all the other possible truth tables and proves that the network is indeed able to learn Boolean functions.

This approach, using asynchronous information processing in PCNs, is fundamentally different to standard self-organizing mechanisms in neural networks: "learning neurons" projecting to weight-changing inputs of synapses embed a modified version of the Delta-rule in the neural net topology. Consequently, no analytical calculation of learning rules is necessary.

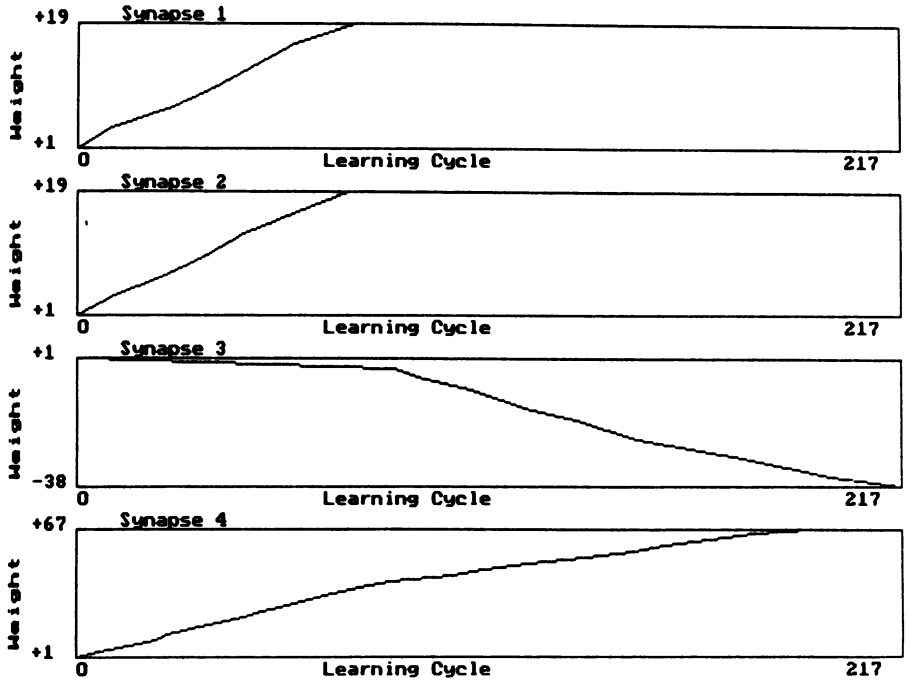


Fig. 4: Synaptic weights of the output unit N_o while learning XOR

References

[1] D. D. Coon, A. G. U. Perera, "New hardware for massive neural networks", in: D. Z. Anderson, ed., Neural Inf. Proc. Systems, AIP, New York, pp. 201-210, 1988

[2] N. E. Cotter, K. Smith, M. Gasper, "A pulse width modulation design approach and programmable logic for artificial neural networks", in: Proc. of 5th MIT Conf. Advanced Research in VLSI, MIT, Cambridge, pp. 1-15, 1988

[3] R. Eckmiller, "Electronic simulation of the vertebrate retina", IEEE Trans. BME-22, pp. 305-311, 1975

[4] G. L. Gerstein, A. M. H. J. Aertsen, "Representation of cooperative firing activity among simultaneously recorded neurons", J. Neurophysiol., vol. 54, pp. 1513-1528, 1985

[5] A. Hamilton, A. F. Murray, L. Tarassenko, "Programmable analog pulse-firing neural networks", in: D. S. Touretzky, ed., Advances in Neural Inf. Proc. Systems I, Morgan Kaufmann, San Mateo, pp. 671-677, 1989

[6] N. El-Leithy, R. W. Newcomb, M. Zaghioul, "A basic MOS neural-type junction - a perspective on neural-type microsystems", in: Proc. of IEEE First Int. Conf. on Neural Networks, San Diego, Vol.III, pp. 469-477, 1987

[7] R. Lestienne, B. L. Strehler, "Time structure and stimulus dependence of precisely replicating patterns present in monkey cortical neuronal spike trains", Brain Res., vol. 437, pp. 214-238, 1987

[8] C. Mead, "Analog VLSI and Neural Systems", Addison-Wesley, Reading, 1989

[9] J. L. Meador, C. S. Cole, "A low-power CMOS circuit which emulates temporal electrical properties of neurons", in: D. S. Touretzky, ed., Advances in Neural Inf. Proc. Systems I, Morgan Kaufmann, San Mateo, pp. 678-686, 1989

[10] L. M. Optican, B. J. Richmond, "Temporal encoding of two-dimensional patterns by single units in primate inferior temporal cortex. III. Information theoretic analysis", J.Neurophysiol. vol. 57, pp.162-178, 1987

[11] S. A. Ryckebusch, J. M. Bower, C Mead, "Modelling small oscillating biological networks in analog VLSI", in: D. S. Touretzky, ed., Advances in Neural Inf. Proc. Systems I, Morgan Kaufmann, San Mateo, pp. 384-393, 1989

BIOLOGICAL LEARNING PRIMITIVES IN ANALOG EEPROM SYNAPSES

H. C. Card
Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Canada R3T 2N2

W. R. Moore
Department of Engineering Science
University of Oxford
Oxford, U.K. OX1 3PJ

Introduction

Biological neural systems employ synaptic learning rules which include associative learning of temporal relationships between multiple inputs. Much of the cellular basis for this learning has been identified experimentally in the neural ganglia of simple invertebrates such as worms, snails and insects [1]. An example is the plasticity in the sensorimotor synapse which controls the gill withdrawl reflex of the marine mollusc *Aplysia* [2-4]. The synaptic weight in this case undergoes (a) habituation, in which repeated stimuli of the siphon elicit reduced gill withdrawl responses as the animal learns to ignore the stimulus (b) sensitization, in which a shock to the tail evokes the opposite response of increased sensitivity to siphon stimulation and (c) classical or Pavlovian conditioning, in which there is a temporal pairing of the conditioned stimulus (CS) to the siphon which precedes the unconditioned stimulus (US) to the tail by a critical time period. In the latter case the enhanced response is similar to sensitization but is learned faster and results in a larger synaptic weight [2-4]. This paper suggests that EEPROM devices of the floating gate tunnel injection geometry can implement in situ nonvolatile versions of all of these biological, non-Hebbian synaptic learning primitives.

EEPROM Synapses

Fig.1 illustrates an EEPROM device of the floating gate tunnel injection type [5]. This device will be shown to represent the equivalent for an artificial VLSI network of a biological synapse such as the sensorimotor synapse described above. The weight of this synapse is proportional to the channel conductance of the EEPROM device which in turn is determined by the charge Q_{fg} on its electrically floating gate FG. This device has an ultrathin oxide region (region 3 in Fig.1) over a diffused region which permits electron tunnelling to the floating gate. We will use the symbol V_p for the voltage applied to this diffused region since in our circuit it represents the programming potential. A potential V_g applied to the upper or control gate also affects the tunneling process by contributing to the potential of the FG given by [5-7]

$$V_{fg} = (Q_{fg} + C_1 V_p + C_3 V_g) / C_T \qquad (1)$$

where the capacitances C_1 , C_2 and C_3 are between the FG and the V_p terminal, between the FG and the channel, and between the FG and the V_g terminal, respectively. The total FG capacitance $C_T = C_1 + C_2 + C_3$. The electric field in the tunnel oxide is given by

$$E = (V_p - V_{fg}) / d \qquad (2)$$

with d the tunnel oxide thickness. The tunnel current to the FG is responsible for the learning process; this current is due to Fowler-Nordheim tunneling and is described by

$$J = \kappa E^2 \exp (- \theta / E) \qquad (3)$$

where $\kappa = 9.625 \times 10^{-7}$ AV^{-2} and $\theta = 2.765 \times 10^{8}$ Vcm^{-1} [8] . The sign of E determines the direction of J; it is the absolute value or magnitude of E which appears in (3). The charge Q_{fg} determines the synaptic weight via the conductance of the MOSFET channel. This conductance depends upon the threshold voltage V_T of the device which in turn is related to the charge on the floating gate via

$$V_T = V_{T0} - Q_{fg} / C_3 \qquad (4)$$

for an n-channel device, with V_{T0} the threshold voltage when Q_{fg} is zero. Modifications to this charge due to the tunnel current thus result in changes to the threshold voltage of

$$\Delta V_T = - \Delta Q_{fg} / C_3 = - J \Delta t \eta / C_3 \qquad (5)$$

where $\eta = A_1 C_2 / (A_2 (C_1+C_2))$, with A_2 and A_1 the areas of the floating gate over the channel and over the tunnel oxide. This factor takes account of charge redistribution on the FG following tunneling from the V_p terminal. In (4) and (5) the charges and capacitances are per unit area. Finally, the channel current in the EEPROM which represents the quantity of neurotransmitter release NT is given by

$$I = NT = (\mu C_0 / 2) (W / L) [(V_g - V_T)^2] \qquad (6)$$

for saturation conditions in the transistor.

Habituation is incorporated into our model by employing the control gate CG and V_g in Fig.1. Assume that the US is zero and that a series of pulses (of magnitude $V_g = 10V$ and duration 1ms) are applied as a CS to the drain and control gate terminals. Initially there is no charge on the FG so the threshold voltage is V_{T0}. The potential V_p and Q_{fg} in (1) are zero and the floating gate potential is $V_{fg} = V_g (C_3 / C_T)$. The magnitude of the electric field E in the tunnel oxide is V_{fg} / d and the tunnel current is given for this E by (3). This current results in a buildup of *negative* charge on the FG, an increase in V_T and thus a decrease in the channel current NT. It also causes a reduction of V_{fg} which reduces E and J. The result is that further pulses continue to increase the threshold voltage but with diminished effect. This is shown in Fig.2 which presents the dependence of ΔV_T on the number of CS pulses (the curve marked habituation). For this simulation we have that $V_g = 10V$, $\eta = 0.1$ and $\Delta t = 10^{-3}$ s. We have used values of $C_3 = 0.6 C_T$, C_3 per unit area = 3.45×10^{-9} Fcm^{-2}, and d = 50 Å .

Also shown in Fig.2 are the effects of sensitization and classical conditioning on threshold voltage. During a sensitization trial US pulses are applied to the V_p terminal with the CS set to zero. Thus V_{fg} in (2) is obtained from (1) with $V_g = 0$ in this case. The charge on the FG now becomes positive. The shift in threshold voltage due to US pulses is shown in Fig.2 by the curve marked sensitization, where we have employed the values $V_p = 6V$ and $C_1/C_T = 0.1$. During classical conditioning a very similar process occurs to that during sensitization. If the US pulse described above is immediately preceded by a CS pulse within a critical time period then circuitry external to the EEPROM can facilitate the positive charging of the FG in three distinct ways: (i) the CS can be made to leave a temporary negative charge on the upper gate, (ii) the CS may provide a substrate bias, or (iii) the CS together with the US can be used to increase V_p. We have developed circuitry which accomplishes (iii) for the example of *Aplysia*

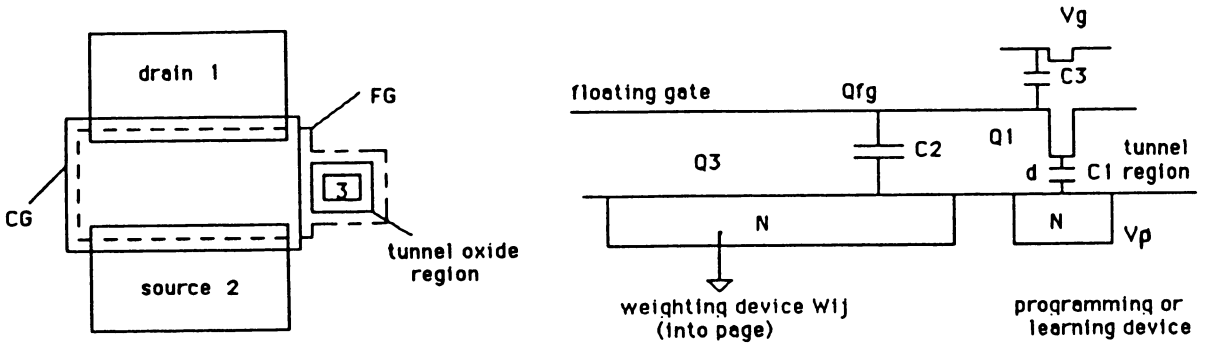


Fig.1. Synaptic EEPROM in plan and edge views showing terminal capacitances.

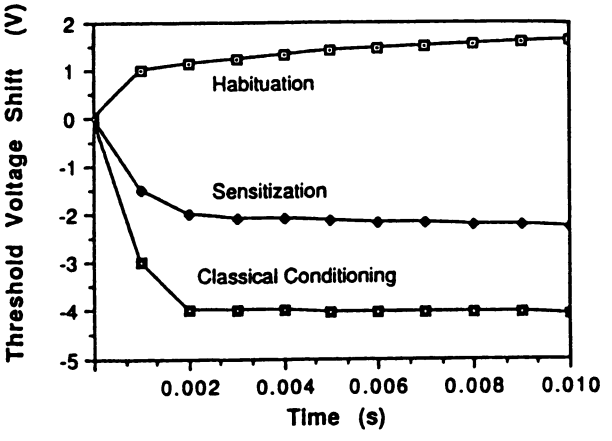


Fig.2. Simulation of shift in threshold voltage ΔV_T of sensorimotor EEPROM following 1ms pulses of conditioned stimulus CS alone (habituation), 1ms pulses of unconditioned stimulus unpaired with test pulses of CS (sensitization), and temporally-paired 1ms CS and US pulses (classical conditioning).

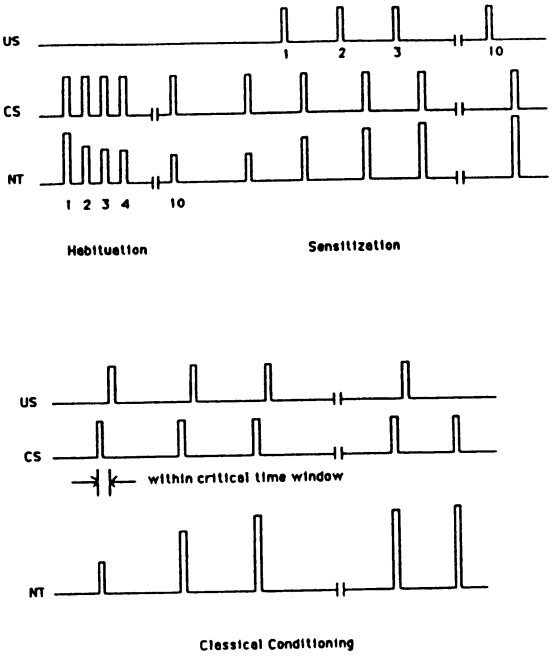


Fig.3. Simulation of learning behaviour at EEPROM synapses from Fig.2 and Eqn.(6). In top trace simulation assumes that delay between CS and US is beyond critical time window for classical conditioning. Relative magnitudes of neurotransmitter release NT are to scale.

synapses by using three extra transistors [9]. This circuitry also prohibits backward conditioning (thereby enforcing a causal relation between CS and US) at the cost of a single transistor. This provides the desired time window for classical conditioning, and the shape of this window may be controlled via the time constants of the external circuitry.The increase in V_p in turn provides a larger electric field in the tunnel oxide layer of the EEPROM and enhanced tunnel current for classical conditioning as compared to sensitization. We demonstrate the effects in Fig.2 (the curve marked classical conditioning) with a value of V_p of 7V. The same EEPROM device thus accomplishes all three biological learning primitives.

The influence of the threshold voltage shifts of Fig.2 on the learning behaviour is illustrated in Fig.3. This is a schematic timing diagram in which the relative magnitudes of the synaptic currents (the NT pulses in the figure) have been determined quantitatively from the results of Fig.2. More or less rapid response is readily accomplished by adjusting assumed values of device and circuit parameters in the simulation. In this sense the silicon model is more flexible than its biological counterpart. An initial threshold votage of 1V was assumed for the EEPROM and the sensitization and classical conditioning results begin with a habituated device. We conclude that in situ synaptic learning primitives for habituation, sensitization, and association including Pavlovian conditioning can be efficiently realized with analog EEPROMs.

Acknowledgements: I am grateful to the Natural Sciences and Engineering Research Council of Canada for financial assistance. The hospitality of Oxford University and Jesus College during the sabbatical year 1988-1989.

References

[1] Shepherd, G.M.: *Neurobiology*, 2nd Ed., 1988, New York: Oxford Univ. Press, Chap.29.
[2] Kandel, E.R., Klein, M., Hochner, B., Shuster, M., Siegelbaum, S.A., Hawkins, R.D., Glanzman, D.L., Castellucci, V.F. and Abrams, T.W.: "Synaptic modulation and learning: new insights into synaptic transmission from the study of behaviour", in Edelman, G.M., Gall, W.E. and Cowan, W.M., editors: *Synaptic Function* , 1987, New York: John Wiley, Chapter 19, pp. 471-518.
[3] Bailey, C.H. and Kandel, E.R.: "Molecular approaches to the study of short-term and long-term memory", in *Functions of the Brain*, C.W. Coen (editor), Oxford: Clarendon Press, 1985, pp. 98-129.
[4] Abrams, T.W., Hawkins, R.D. and Kandel, E.R.: "Cellular studies of an associative mechanism for classical conditioning in *Aplysia*: activity-dependent presynaptic facilitation", in *Model Neural Networks and Behaviour,* A.I. Selverston, editor, New York: Plenum Press.1988, Chap. 12, pp. 211-236.
[5] Lai, S.K. and Dham, V.K.: 'VLSI electrically erasable programmable read only memory', in *VLSI Handbook* , N.G. Einspruch, editor, 1985, Academic Press: New York, pp.167-176.
[6] Card, H.C. and Moore, W.R.: " EEPROM synapses exhibiting pseudo-Hebbian plasticity", *Electronics Letters,* 1989, 25, pp. 805-806.
[7] Card, H.C. and Moore, W.R.: "Implementation of plasticity in MOS synapses", *First IEE Conference on Artificial Neural Networks*, London, Oct 16-18, 1989.
[8] Lenzlinger, M. and Snow, E.: " Fowler-Nordheim Tunneling in SiO_2" , *J. Appl. Phys.*, 1969, 40, pp. 278-283.
[9] Card, H.C.: "Analog VLSI models of neurobiological learning and memory", Invited paper, *Canadian Conference on Electrical and Computer Engineering*, Montreal, Sept 17-20, 1989.

Introducing A Neural Network Design Language

Sing-Chai Chan
Department of Information systems and Computer Science
National University of Singapore
Singapore, 0511
(scchan@nusvm.bitnet)
(Fax: 65-779-4580)

and

Yaohan Chu
Department of Computer Science
University of Maryland at College Park
College Park, MD 20742
(ychu@mimsy.umd.edu)
(Fax: 301-454-8346)

1. Introduction

This paper introduces a neural network design language [1] for solving problems using the neural three-layered perceptron model [2] with three-valued-logic network [3]. Users may describe the features of a problem in a natural-language-like manner instead of going through the pain of using the conventional binary representation. The following introduces the syntax semantics of the language. It also illustrates the language with an example.

2. An Example

Consider a car-dealer record system where the used cars are specified according to selling price, year manufactured, and engine displacement. Suppose the range for the specifications are as follows:

Price (10K+, 5-10K, 5K-);
Year (2YRS-, 2-5YRS, 5YRS+);
Displ (2000cc+, 1600-2000cc, 1600cc-);

In the above, the price is divided into 3 catagories: larger-than $10,000, equal to or between $10,000 and $5,000, and less than $5,000. Similarly, the year and displacement each has 3 catagories. Furthermore, each car is identified by tag number, engine number, make, model, and year as described below.

Car-id (Tag#, Engine#, Make, Model, Year);

Suppose the following cars are available at the dealer:

Car (10K+, 2YRS-, 2000cc+);
Car (5-10K, 2YRS-, 1600-2000cc);
Car (5K-, 2-5YRS, 1600-2000cc);
Car (5K-, 2-5YRS, 1600cc-);
Car (5K-, 5YRS+, 2000cc+);

The corresponding car identifications are:

Car-id (XYZ123, 112233, Buick, Century, 1988);
Car-id(MDS7345, 340703, Honda, Accord, 1987);
Car-id (ABC987, 987654, Toyota, Selica, 1985);
Car-id (MNK222, 128268, Chrysler, Labron, 1984);
Car-id (WAT357, 778211, Ford, Escort, 1978);

The car dealer tries to match these cars for the buyers, according to the specifications given by the buyers. There may be no match, one match, or serveral matches. The program in the neural network design language [1] (NNDL) is shown to be developed below.

3. Program Heading

The program of NNDL begins with the program heading followed by the user-defined program name. The enclosed program parameters consist of input name, output name, and unknown input name. The program heading is shown below.

Program Progname (Input-name, Output-name, Unknowninput);

Input-name and Output-name are user-defined names to represent the set of input and output vectors, respectively. Unknowninput is also defined by the user to represent an unknown input vector. an example is:

Program Dealer (Car, Carid, Buycar);
(* This example is to buy a car from the stock in a car dealer *)

4. Input and Output Attributes

Suppose there are m attributes: A_1, A_2, ..., A_m to represent m input features (or characteristics), and n attributes: B_1, B_2, ..., B_n to represent n output features. Then, the NNDL will appear as follows:

Input Attribute:
    Inputname (A_1, A_2, ..., A_m);
Output Attribute:
    Outputname (B_1, B_2, ..., B_n);

Using the same example, the following declarations are for input and output attributes.

Input Attributes
    Car (Price, Year, Displ)
Output Attributes
    Car-id(Tag#, Engine#, Make, Model, Year)

5. Ranges for Input Attributes:

The constants of each input attribute usually represent range of values instead of a specific value. For example, the ranges of "Agegroup": AG1, AG2, AG3, AG4 and AG5 are shown below:

AG1: 65+ (retired age);
AG2: 40-65 (middle age);
AG3: 21-39 (adult);
AG4: 12-21 (youth);
AG5: 0-12 (child);

In many applications, it is more meaningful to use age group than age.

In NNDL, the ranges of each input attribute can be defined as follows:

Attribute Range:

A_1 (R11, R12, ..., R1r); (* r ranges for A_1 *)
A_2 (R21, R22, ..., R2s); (* s ranges for A_2 *)
...
A_m (Rm1, Rm2, ..., Rmt); (* t ranges for A_m *)

where R_{ij} is the jth range value for attribute A_i. Using the same example, the attribute range is:

Attribute Range

Price (10K+, 5-10K, 5K-)
Year (2YRS-, 2-5YRS, 5YRS+)
Displ (2000cc+, 1600-2000cc, 1600cc-)

6. Input and output Constants

A set of initial exact input and output values must be given to the system to serve as the database for self-learning, which is used to match exact or possible answers for any unknown inputs. For a given input vector,it is possible to have more than one answers.

Suppose that the constants of a set of k input and output vector-pairs are given to form the database. In NNDL, the initial input constants can be defined as follows:

Input Constant

Inputname (C_{11}, C_{12}, ..., C_{1m});
Inputname (C_{21}, C_{22}, ..., C_{2m});
...
Inputname (C_{k1}, C_{k2}, ..., C_{km});

Where C_{ij} (i=1,2, ..., k, j=1, 2, ..., m) is the input value for attribute A_j in the ith input vector. Likewise the initial output constants can be defined as:

Output Constant:

Outputname (D_{11}, D_{12}, ..., D_{1n});
Outputname (D_{21}, D_{22}, ..., D_{2n});
...
Outputname (D_{k1}, D_{k2}, ..., D_{kn});

where D_{ij} (i=1,2, ..., k, j=1, 2, ..., n) is the initial output value for attribute B_j (or jth element) of the ith output vector. Using the same example, the following declarations are for input and output vector constants.

Input Constants

Car (10K+, 2YRS-, 2000cc+);
Car (5-10K, 2YRS-, 1600-2000cc);
Car (5K-, 2-5YRS, 1600-2000cc);
Car (5K-, 2-5YRS, 1600cc-);
Car (5K-, 5YRS+, 2000cc+);

Output Constants

Car-id (XYZ123, 112233, Buick, Century, 1988);
Car-id(MDS7345, 340703, Honda, Accord, 1987);
Car-id (ABC987, 987654, Toyota, Selica, 1985);
Car-id (MNK222, 128268, Chrysler, Labron, 1984);
Car-id (WAT357, 778211, Ford, Escort, 1978);

Now, the definitions required for a NNDL program is completed, and the neural network model is formed.

7. Program body

Once the neural network model is formed, we are ready to accept unknown inputs and to match for exact output solution or possible output solutions. The program body defined in NNDL is as follows:

```
Begin
    Do
        Read Unknowninput (A1, A2, ..., Am);
        Write Outputname (Unknowninput);
    Until EoF
End.
```

Using the same example, the program body is,

```
Begin
    Do
      Read Buycar (Price, Year, Displ);
      Write Car-id (Tag#, Engine#, Make, Model, Year);
    Until EoF;
End.
```

8. Complete Program and Execution

By combining the above constructs, the complete program is thus obtained. For execution, assume that the input for the Read statement is:

Read Buyer (10K +, 2YRS-, 2000cc-);

After matching in a 3-layered perceptron, the output is found to be:

XYZ123, 112233, Buick, Century, 1988

The above output is a matched one. Assume that another input for the Read statement is:

Read Buyer (5K-, 2-5YRS, unknown);

where "unknown" refers to the input attribute of engine displacement. There are two outputs:

ABC987, 987654, Toyota, Selica, 1985

MNK222, 128268, Chrysler, Lebaron, 1984

In summary, the program structure consists of the definitions of program name, Input Attribute, Output Attribute, Attribute Range, Input Constant, Output Constant, and program body. The program body so far has only three statements: Do ... Until, Read, and Write. The Read statement is used as usual to accept unknown input attributes. The write statement prints out only exact or possible solutions for the unknown input. The design language provides a friendly and easy environment for user to describe the features of the problem in a natural-language-like manner rather than goes through the pain of using the conventional binary representation in neural network model.

9. Remarks

The data sets of the neutral 3-valued logic network are in vector form, and the processing in the network is parallel. The design language can be made interactive. An interactive design environment can be implemented for direct interaction and debugging.

10. References

[1] S.C. Chan and Yaohan Chu, "A neural Network Design Language" Technical Report TR-2260, Department of Computer science, University of Maryland, College Park, MD; July, 1989

[2] Rumelhaut, D.E. and J.L. McClelland. Parallel Distributed Processing. MIT Press, Bambridge, MA, 1987, Vol. I & II.

[3] Chan, S.C., L.S. Hsu, S. Broody and H.H. Teh. "Neural Three-Valued Logic Network", Abstract, Proceedings, International Joint Conference on Neural Networks, Washington, D.C., June 18-22, 1989. Vol 2, p. 594.

[4] Yaohan Chu, J.L. Ding and Y./f. Ding, "Software Implementation of a Pascal Direct-execution Machine", Technical Report TR-1824, April, 1987

Hybrid Neurocomputer Using Optical Disk

Kyusun Choi, Taiwei Lu, William S. Adams, and Francis T. S. Yu

Department of Electrical Engineering
The Pennsylvania State University
University Park, PA 16802

Abstract

A neural network model can be implemented with an optical disk. By doing so, the capacity and speed of the associator are increased beyond that of any existing neural network model implementation. Justification of this claim is presented by the design and analysis of the binary weight pattern associator using an optical disk.

Introduction

Neural networks are noted for their applications. Hand written character recognition, speech recognition, visual data comprehension, language comprehension, optimal control, data classification, inference engine, etc. are practical applications. Some of them are discussed in the literature [Ande88] and [Rume86a]. With such potential applications in mind, there has been an effort to incorporate the neural network in computer architecture. This paper presents the design of the *neurocomputer* which incorporates digital electronics, optics, and optical disk technology which enhances the usefulness with significantly increased capacity and speed.

Neurocomputer

Invariably, the models of a neural network exhibit an associative characteristic; thus, an implementation of a neural network in this paper is called an *associator*. It can be attached to a conventional computer system in a similar way as a CPU. The computer system which has an associator is referred to as a *neurocomputer*.

Figure 1 is the block diagram of the neurocomputer using optical disks. It employs both the digital electronics associator as well as the optical associator. The block AP is the associator implemented with digital electronics. The details of its architecture are presented in [Choi89b]. The OD blocks are the optical disk units with hybrid optics for associative retrieval. Their details are the focus of the following presentation. Note that there are two sets of buses. The BUS1 is used to interchange data among the ODs and AP. It is crucial to accommodate the highest data bandwidth on the interconnections among optical associators and the electronics associator in order to fully exploit the speedup advantages. The data width of an associator is very large. It has the same number of bits as the number of neurons in an associator. The BUS2 is a conventional computer bus; it connects OD and AP to a CPU and memory. There can be multiple AP as well as OD units.

Neural network model

The neural network model implemented both electronically and optically in this paper is the *Binary Weight Pattern Associator** (BWPA) which is perhaps the simplest model of a neural network. It was originated by D. J. Wiilshaw [Will69]. His studies of information

* This name is an invention of the author; [Rume86a] calls it pattern associator.

storing and retrieving using optical holography led Willshaw to his ingenious network model correlograph and its simplified version, the BWPA. Willshaw then carried out the analysis of the network in [Will71] and [Will81]. The network analysis is followed by another, [McCl86a]. The associator is then applied in [McCl86b] to read English words and in [Rume86b] to learn past tenses of English verbs. Also in [Aust87], it is used in pattern recognition and 3 dimensional scene analysis. Anderson and Rosenfeld, in their introductory remark for Willshaw's paper [Ande88], noted its suitability for modern VLSI implementation.

Furthermore, the model can be efficiently implemented with an electro-optics system. The main advantage of an optical associator is its speed. The vector-matrix multiplication is carried out at the speed of light. The whole process of retrieving a pattern can be done as fast as displaying a pattern. More importantly, the optical implementation enables the use of an optical disk as massive associative storage at fast retrieval speed.

Optical disk based neural network architecture

Figure 2 illustrates the optics arrangement of the optical disk interface for the retrieval operation. Starting from the upper left, the pulsing laser light is fanned and directed to the moving head scanning on the disk surface. The moving head consists of a mirror M_1, a polarized beam splitter PBS_1, a quarter wave plate QWP_1, and a lens L_2. The moving head travels along a diagonal line of the disk and its travel distance covers the most inner tract to the most outer tract. The laser beam reflected off the disk surface is directed to the lens L_2 which in turn projects the image, with magnification, on the diffuser D_1. The arrangement past the diffuser is identical to the optical neural network proposed by T. Lu et al. in [Lu89]. The diffuser replaces the TV screen which displays the interconnection weight matrix. Through the lenslet array, the sub-blocks of the weight matrix are projected on to the input vector displayed on the spatial light modulator, SLM_1. In the resulting image, the product of the weight matrix and the input vector is then focused on the photo-detector array to be converted into the digital electronics signal.

Based on currently available technology, the following example parameters are calculated. The area on the disk surface, on which the laser beam will be focused, is 1 mm^2. This is primarily due to the limitation of L_2 and L_3, the microscopic lenses' field of view, in which the magnification factor is about 70. The area of 1 mm^2 amounts to 529×529 bits, a matrix block for 529 neurons. Note that the square is distorted radially. This distortion can be corrected by the lens or it can be accounted for by the SLM and the photo-sensor. Also note that the number of sectors varies radially. This way, it uses the disk surface area more efficiently.

To retrieve an association, the moving head seeks (locates) the desired matrix block, and the laser unit emits pulsed light to capture the matrix block on the diffuser screen. The approximate formulation of the required laser power is $\eta \times P_L > S_{ph}$ where P_L is the laser power, S_{ph} is the photo-detector sensitivity over its response time, and η is the absorption coefficient. η can be written as $\eta = (1/2) \times t_1 \times t_2 \times t_3$ where t_1 is the reflectivity of the disk surface, t_2 is the transparence of the SLM, and t_3 is the energy loss factor between the diffuser and the SLM. We assumed, as an example, that $t_1 = 0.5$, $t_2 = 0.2$, $t_3 = 5 \times 10^{-3}$, and $S_{ph} = 10^{-5}$ watts over a 10 $nsec$ response time. The resulting power is $P_L > 10^{-5}/(0.5 \times 0.5 \times 0.2 \times 5 \times 10^{-3}) = 4 \times 10^{-2}$ watts. It is also required to maintain the matrix image on the diffuser over the duration of the

photo-detector response time. This limits rotational speed of the disk to at the most $V_{rpm} = (l_p/l_s) \times t_{ph} = (10^{-6}/(\pi \times 12 \times 0.0254)) \times 60 \times 10^8 = 6,265$** rpm where l_p is a pit length, l_s is the circumference of a disk, and t_{ph} is the response time.

Capacity and speed

Based on the above design parameters, there are 3,663 matrix blocks on each side of the disk (7,326 blocks per disk***). Assuming that each block can store 50 associations, there are 366,000 associations that can be stored per disk. This number well exceeds the 150,000 entries in the Webster's Collegiate Dictionary.

With the use of 529 parallel photo-detectors, retrieval of a pattern can take within 100 *nsec* – 50 *nsec* for AD conversion and 50 *nsec* for thresholding (10 bit arithmetic). Since each matrix block consists of 279,841 connections, this yields a peak processing rate of $279,841/(100 \times 10^{-9}) = 2.8 \times 10^{12}$ connections per second. Compare this with the multiprocessor of 529 processors and a 50 *nsec* memory access time. The parallel algorithm takes $O(n\ log(n))$ time steps [Choi89], which yields a peak processing rate of 1.1×10^9 connections per second. Furthermore, the currently available single processor, specialized for neuro computing, is rated at a peak 10^6 connections per second [Hnc88]. The speed up factor of the optical disk-based processing over the parallel computer algorithm is 10^3 and over the best single processor algorithm is 10^6.

However, the operational speed of the optical disk-based neurocomputer is limited by practicality. The input and output of the OD units are directed from and to the AP, an electronics associator. Thus the operating speed is limited by the slower of the two, the AP's. But, this is still an advantage since ODs may allow AP's speed be sustained at its peak.

Summary

The original idea and the design of a neurocomputer using an optical disk is presented. It takes the advantages of recent technological developments such as neural networks, digital electronic computers, optical information processing, and optical disk technology. Through this effort the usefulness of the neurocomputer has been increased. The gained advantages are (1) a large associative data base capacity, (2) a better method of providing stable, permanent, compact, and economic associative data storage, and (3) a speedy retrieval of a large amount of associative data (2.8×10^{12} connections per second peak processing rate). Perhaps this neurocomputer will make the real time recognition of image and voice a reality in the near future.

Also, this design reveals some of the difficulties and potential problem areas of using the optical disk as associative storage. First, writing the information on the disk still relies on the old method of a serial bit stream and presently the disk allows writing only once. Second, the weight of the moving head (including the set of lens, mirror, and beam splitter) may significantly degrade the seek time. Lastly, the size of the network is limited. However, these difficulties can be overcome by further development of the technology.

REFERENCES

[Ande88] James A. Anderson and Edward Rosenfeld, Eds., *Neurocomputing*, The MIT Press, Cambridge, MA, 1988.

** Based on a 1 μm pit length and a 12 inch disk diameter [Opti89].

*** A single optical disk provides 2×10^9 connections [Opti89].

[Aust87] James Austin, "ADAM: A Distributed Associative Memory for Scene Analysis," Proc. *IEEE Int'l Conf. Neural Networks*, 1987, pp IV285–IV292.

[Choi89] Kyusun Choi, "VLSI implementation of BWPA Neural Network model," *Research Report Jul89-ECL*, Engineering Computer Lab, The Pennsylvania State University, Jul. 1989.

[Hnc88] HNC ANZA Plus/VME sales brochure, HNC Inc., San Diago, CA, 1988.

[Lu89] T. Lu, et al., "A 2–D Programmable Optical Neural Network," to be appear in Applied Optics.

[McCl86a] James. L. McClelland, "Resource Requirements of Standard and Programmable Nets," In D. E. Rumelhart, J. L. McCelland, et al. (Eds.), *Parallel Distributed Processing*, The MIT Press, Cambridge, MA, 1986, vol. 1, pp 460–487.

[McCl86b] James. L. McClelland, "The Programmable Blackboard Model of Reading," In D. E. Rumelhart, J. L. McCelland, et al. (Eds.), *Parallel Distributed Processing*, The MIT Press, Cambridge, MA, 1986, vol. 2, pp 122–169.

[Opti89] Optimem 1000 sales brochure, Optimem Inc., Mountain View, CA, (1989).

[Rume86a] David E. Rumelhart, James L. McCelland, et al., Eds., *Parallel Distributed Processing*, The MIT Press, Cambridge, MA, 1986, vol. 1 and 2.

[Rume86b] David E. Rumelhart and James. L. McClelland, "On the Learning the Past Tenses of English Verbs," In D. E. Rumelhart, J. L. McCelland, et al. (Eds.), *Parallel Distributed Processing*, The MIT Press, Cambridge, MA, 1986, vol. 2, pp 216–271.

[Will69] D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins, "A Non-Holographic Model of Associative Memory," *Nature*, **222**: 960, 1969.

[Will71] D. J. Willshaw, *Models of Distributed Associative Memory*, Unpublished doctoral dissertation, University of Edinburgh, 1971.

[Will81] D. J. Willshaw, "Holography, Associative Memory, and Inductive Generalization," In G. E. Hinton & J. A. Anderson (Eds.), *Parallel Models of Associative Memory*, Erlbaum Publ., Hillsdale, NJ, 1981, pp 83–104.

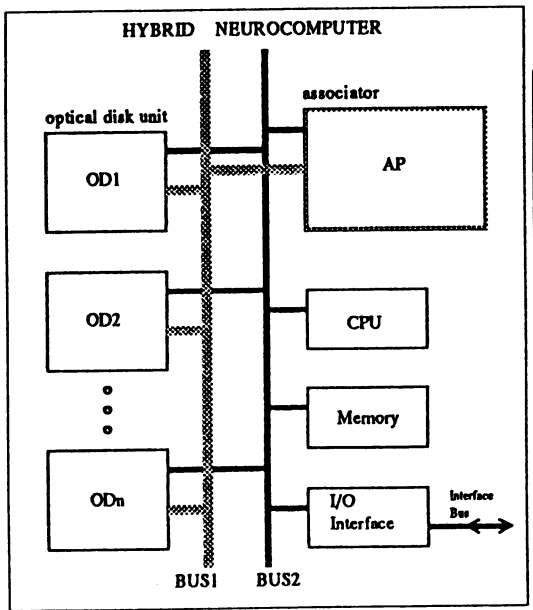


Figure 1 Block diagram of the neurocomputer using optical disk.

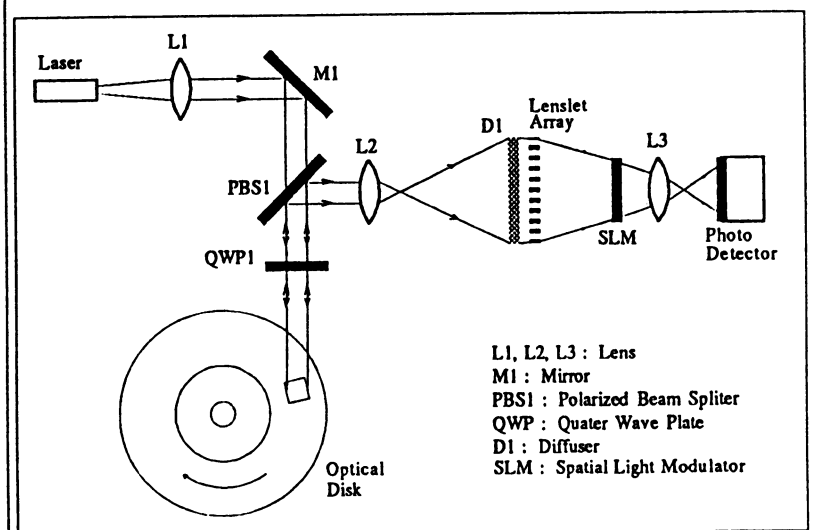


Figure 2 Optics arrangement of the optical disk interface for the association retrieval.

An Analog CMOS Implementation of a Self Organizing Feedforward Network

James J. Clark

Division of Applied Sciences
Harvard University
Cambridge, MA

Abstract

We describe the design of a CMOS current mode neuron for use in neural networks and describe its application to the construction of a self organizing feedforward network. The synapses are implemented with a modified version of the Bult-Wallinga [2] four quadrant analog current multiplier. We present the design of a current mode based circuit which allows the adjustment of the synaptic weights according to a Hebbian learning rule. This allows the circuit to be used in a self organizing network, such as the Kammen-Yuille orientation selective network [6,10]. The neural elements described in this paper can be implemented in a standard CMOS integrated circuit fabrication process.

1 Self Organizing Feedforward Networks

In [6] Kammen and Yuille showed that a two-dimensional feedforward network of linear summation elements with stochastic inputs can develop (learn) into a system that exhibits directional selectivity (i.e. the response of a neuron on the output layer of the network responds maximally to patterns of input oriented in a specific direction). The adaption mechanism they used to obtain their result was similar to that of Linsker [8] and used the following synaptic weight update equation:

$$w_{ij}(t+\Delta t) = w_{ij}(t) + \Delta t(-A - 2Bw_{ij}(t) - 4Cw_{ij}^3(t) + 2 < O_i, I_{ij} >) \quad (1)$$

where i indicates a particular neuron, j indicates a particular synapse on the neuron, $w_{ij}(t)$ is the weight of synapse j on neuron i at time t, O_i is the output of neuron i, I_{ij} is the input to synapse j on neuron i and $<,>$ indicates expectation or stochastic correlation. The terms with the B and C multipliers are needed in Kammen and Yuille's model to keep the w_{ij}'s bounded. We will follow Linsker [8] in obtaining boundedness by using sigmoidal nonlinear summing elements that saturate when the magnitude of the weighted sum of the inputs is large. Thus, in the circuit described below we set $B = C = 0$.

This research was supported in part by the Office of Naval Research and the Joint Services Electronics Program under grant N0014-84-K0504, as well as by the NSF Systems Research Center of Excellence, under grant number CDR-85-00108.

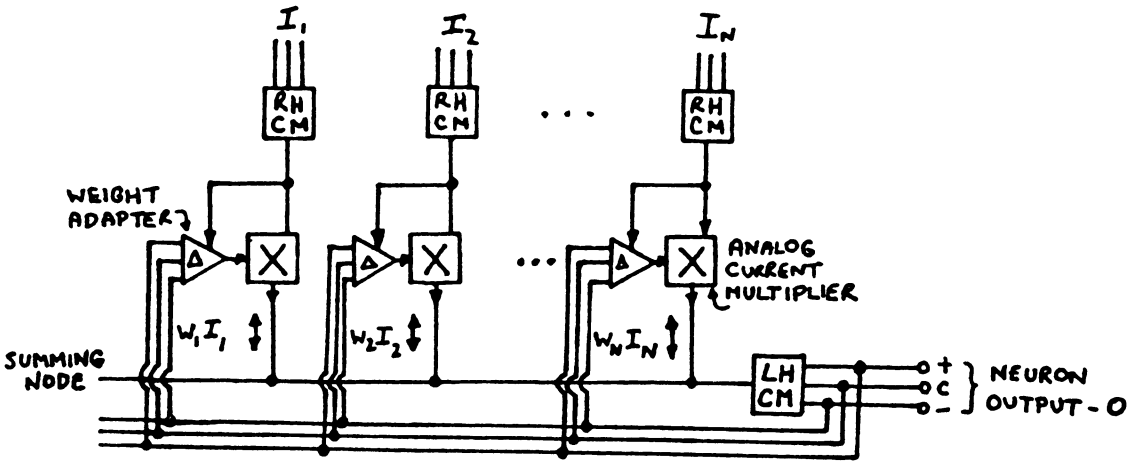


Figure 1: An analog multiplier based current mode neuron.

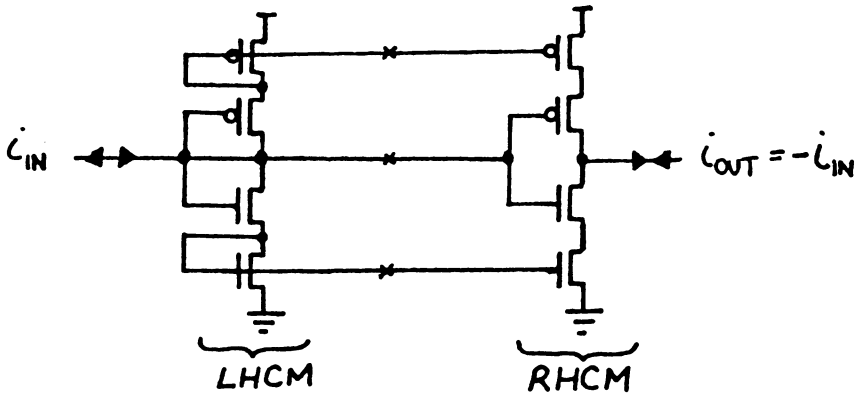


Figure 2: The bidirectional cascode current mirror.

2 A Current Mode Neuron

The standard model of processing elements in artificial neural networks [9] is a linear weighted summation of inputs followed by amplification by an amplifier having a nonlinear (sigmoidal) transfer function. We present here a current mode based approach to the implementation of this standard model. Current mode analog circuit design [7,4,5] is a methodology wherein currents are used to represent information rather that voltages.

The design of our neural element is shown in figure 1. There are two major building blocks of our current mode neuron. The first is an analog current multiplier, the second is a bidirectional cascode current mirror.

The analog current multiplier is based on the scheme of Bult and Wallinga [2]. The only difference between our version and that of Bult and Wallinga is that a cascoded current mirror is used in place of a standard current mirror in order to reduce the output current offset error (i.e. the output current when the input currents are both zero). The multiplier is a four quadrant multiplier which means that both the input and weight currents can be bidirectional, as required.

The bidirectional current mirror consists of p-channel and an n-channel cascode current mirrors [1]. It is a circuit that allows the replication and scaling of currents. This circuit is shown in figure 2. Note that one can split the current mirror into two halves, one which receives the input current, the other which produces the output current. We will refer the the left half of the current mirror as LHCM, and the right half by RHCM. The two halves are connected by three voltages which, as a set, represent the output current. By sending these three voltages to other RHCM's one can obtain replication of currents. This will be used to

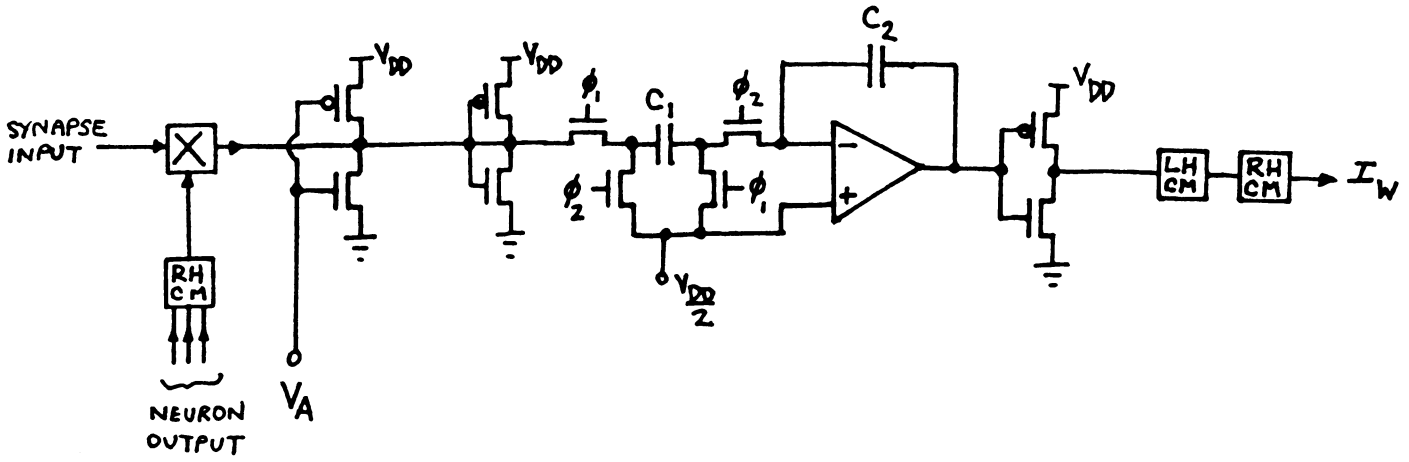


Figure 3: The synaptic weight adaption circuit.

send the output of our neuron to many different neurons.

The current mode neural element consists of three parts: a set of synapses, the summing junction, and the weight adaption circuits (one for each synapse).

The synaptic function is performed by an analog multiplier which multiplies the input current by a weight current. The input to the synapse is a set of three voltages from the LHCM at the output of the neuron feeding this particular synapse. These voltages are converted to a current with a RHCM and this current is multiplied by a weight current (in the form of three cascode mirror voltages) in the analog multiplier. The weight current is produced by the weight adaption circuit which will be described in the next section.

The summing of the weighted input currents is performed by a single current summing node connected to the outputs of the analog multipliers and to an LHCM. The LHCM takes up the excess current flow coming from the analog multipliers which, by Kirchoff's current law, is equal to the sum of the outputs of the analog multipliers.

3 The Synaptic Weight Update Circuit

Each synapse contains circuitry which allows it to adapt the weight applied to its input according to the learning rule given earlier (equation 1). The synapse weight adjustment circuit which has this task of updating the synaptic weights of a current mode neuron is shown in figure 3.

The weight adaption circuit consists of two main blocks, the input-output multiplier and a leaky two phase switched capacitor integrator. The input-output multiplier, which computes the $I_{ij} \times O_i$ term in the weight update equation uses the modified Bult-Wallinga multiplier circuit described earlier. The current output by the input-output multiplier is fed into the leaky integrator. A constant current (corresponding to A in the weight update equation) is subtracted from the input to the leaky integrator (this causes most of the leaking) by a current source/sink consisting of transistors M1 and M2. The voltage V_A adjusts the amount of current subtracted from the input (it is also used to compensate somewhat for offsets in the switched capacitor integrator opamp). The input current is converted to a voltage with the left half of a

standard (i.e. non-cascoded) current mirror (transistors M3 and M4). The transistors in this current mirror have a relatively large channel width to length ratio in order to provide for a small current gain (when the current source voltage is converted back to a voltage with transistors M9 and M10). This small gain corresponds to the Δt term in the weight update equation. The current mirror voltage is then integrated with a switched capacitor integrator to perform both the time averaging required by the correlation term in the weight update term, and the actual storage and updating of the weight.

Detailed description of the design and operation of the weight adaption circuit can be found in [3].

4 Summary

We have presented the design of an adaptive CMOS analog neural network which implements the learning rule of Kammen and Yuille's self organizing neural network. It utilizes CMOS analog current multipliers to perform the multiplication of inputs by weights. The weights are adapted according to a simple Hebbian learning rule, implemented with a correlator and a leaky integrator. The weights are stored dynamically on a capacitor, hence the circuit must be constantly running and learning, else all learned states will be lost.

A prototype of an implementation of a 10 synapse neuron is currently being fabricated through the MOSIS facility in a 2 micron CMOS process. The size of the neural element is roughly 1.0 by 1.4 millimeters.

References

[1] Allen, P.E., and Holberg, D.R., **CMOS Analog Circuit Design,** Holt Rhinehart Winston, NY, 1987

[2] Bult, K. and Wallinga, H., "A class of analog CMOS circuits based on the square-law characteristic of an MOS transistor in saturation", IEEE Journal of Solid State Circuits, Vol. 22, No. 3, pp 357-364, 1987

[3] Clark, J.J., "Current mode implementation of a self-organizing feedforward network", Harvard Robotics Lab Technical Report, No. 89-3, Division of Applied Sciences, Harvard University, 1989

[4] Dao, T.T., McLuskey, E.J., and Russell, K.L., "Multivalued Integrated Injection Logic," IEEE Transactions on Computers, Vol. 26, pp 1233-1241, 1977

[5] Horwitz, C.M., and Silver, M.D., "Complementary current mirror logic", IEEE Journal of Solid-State Circuits, Vol. 23, No. 1, pp 91-97, 1988

[6] Kammen, D.M., and Yuille, A.L., "Spontaneous symmetry breaking energy functions and the emergence of orientation selective cortical cells", Biological Cybernetics, Vol. 59, pp 23-31, 1988

[7] Kawahito, S., Kameyama, M., Higuchi, T., and Yamada, H., "A 32x32 bit multiplier using multiple-valued MOS current mode circuits," IEEE Journal of Solid-State Circuits, Vol. 23, No. 1, pp 124-132, 1988

[8] Linsker, R., "From basic network principles to neural architecture: Emergence of orientation-selective cells.", Proc. Natl. Acad. Science, USA, Vol. 83, pp 8390-8394, 1986

[9] Lippmann, R.P., "An introduction to computing with neural nets", IEEE ASSP Magazine, pp 4-22, April 1987

[10] Yuille, A.L., Kammen, D.M. and Cohen, D., "Quadrature and the development of orientation selective cortical cells by Hebb rules.", submitted Biological Cybernetics.

CASENET
Computer Aided Neural Network Generation Tool

R. W. Dobbins and R. C. Eberhart
The Johns Hopkins University
Applied Physics Laboratory
Laurel, MD 20707

INTRODUCTION

For many applications of neural networks, such as electro-encephalogram (EEG) spike and seizure detection, there is a need to explore different network architectures and paradigms. Generally a network must be hand coded and debugged for each new application. Many researchers are faced with the problem of developing their own neural network software and the time spent on this activity detracts from the application.

Various general purpose parallel processing system simulators have been proposed, in efforts to provide researchers with network development tools. These have generally been interpretive systems requiring large scale or special purpose processors to achieve reasonable performance. While such simulation systems may be useful for some applications, they have not been usable on personal computers and smaller general purpose machines. A large class of problems can be effectively solved on personal computers, provided that the neural network software is reasonably well optimized. The CASENET approach is to generate highly optimized machine code from the user's network specifications in order to maximize performance of the available computing machinery.

A second problem of the simulators that have been proposed, is that they present a complex command language that takes time to master and use effectively. Instead, CASENET provides a graphical interface, that allows the user to draw the network architecture on a graphics screen and enter network attributes from menus on the screen. The network graph together with the attributes attached to the nodes of the graph, form the network specification, from which executable code is automatically and quickly generated.

In this paper the basic CASE (Computer Aided Software Engineering) tools and mechanisms used by CASENET to automate neural network generation, are described. Preliminary results for typical applications using Back Propagation Networks, a paradigm well suited to EEG applications are described. Other types of networks are being incorporated into the development environment.

COMPONENTS OF CASENET

The CASENET system consists of four major sets of tools, namely the network definer, the analyzer, the code generator and the compiler. Figure 1 illustrates the CASENET system components. The network definer is a graphical network editor for drawing the desired network architecture. The editor translates the user's graphics into a formal representation. The analyzer validates this network definition and extracts essential network attributes. The code generator parses the network definition and emits code fragments. The compiler builds the executable network from the code fragments and a generic network skeleton. The results of each phase of the network translation, are available as intermediate files, which the advanced user can edit to customize or optimize any level of

the network.

These tools are described in detail in the following sections, with examples showing their use.

GRAPHICAL NETWORK SPECIFICATION

The architecture of a generic neural network can be defined using a few simple concepts. Neural networks are typically viewed as consisting of several layers, each layer comprising a number of processing elements. However, some applications may require several hidden layers, partitioned into slabs, as illustrated in Figure 2. A slab is a more convenient, more general concept to work with. It is a group of elements with identical attributes, activation functions and connections to other slabs. An example of the resulting specification for a three layer back propagation network follows:

```
slab(1,48).        connect(5,1).        inputs([1,2,3,4]).
slab(2,48).        connect(5,2).        outputs([6]).
slab(3,48).        connect(5,3).
slab(4,48).        connect(5,4).
slab(5,16).        connect(6,5).
slab(6,4).
```

The statement,

```
slab(id, number of units).
```

creates a slab with a unique identifier and number of processing elements,

```
connect(destination, source).
```

establishes a path from the source slab to the destination slab,

```
inputs(list of input slabs).
outputs(list of output slabs).
```

create connections from the neural network to the outside world. The graphical editor is used to build a visual executable model of the neural network. The user places slabs on the graphics screen with simple mouse commands and connections between slabs by drawing directed arrows. The diagram is annotated with the slab identifiers and number of units.

NETWORK ANALYSIS AND CODE GENERATION

The analyzer and code generator are automated tools, to validate the network definition and generate executable code. The analyzer checks that a valid neural network architecture has been defined. Two important facts are that all nodes are properly connected and, for a feedforward network, that there be no feedback cycles. These facts are easily established using Prolog. To search all slabs of the network and check that each slab is connected, the predicates,

```
connected(X, X).
connected(X, Y) :- connect(X, Y).
connected(X, Y) :- connect(X, Z), connected(Z, Y).
```

define the transitive closure of the connect() facts in the database, to recursively determine connections in the network.

The analyzer detects cycles by using the connect() facts to visit all slabs in the network, as in the predicates,

```
cyclic(X, Visited) :- member(X, Visited).
cyclic(X, Visited) :- connect(X, Y), cyclic(Y, [X | Visited]).
```

NETWORK CODE GENERATION

The code generator parses the validated network definition and emits C code fragments, each corresponding to a particular net task:

> allocate slab, read weights, read patterns, read targets,
> iterate net, delta net, iterate weights, sum squared error,
> write weights, write net, free slab.

The C code consists of a small code skeleton with several parameters. The Prolog code generator unifies parameters with the particular network architecture being defined. Typical parameters are,

> X, Y name of the slab, fan-in slab
> N, M size of the slab, fan-in slab

The parameterized code fragment for the "iterate net" task is

```
float  sum = BiasX[i];
for (j = 0;  j < N;  j++)
   sum  += wX[j] * outY[p][j];
outX[p][i] = 1 / (1 + exp(-sum));
```

NETWORK COMPILER

Compiles the network using a generic network skeleton that has slots for net tasks. Each slot is filled by a code fragment for a net task. The network is compiled directly by a C compiler or native code assembler.

DISCUSSION

CASENET is in use to generate working neural networks in EEG spike and seizure detection applications [1]. Ways to optimize code for numeric and other co-processors are under development. Once debugged, the code generator can reliably and quickly generate different kinds of network architecture to suit researcher's needs. This approach can easily be extended to radically different neural network paradigms.

REFERENCES

[1] R.C.Eberhart, R.W.Dobbins and W.R.S.Webber: CASENET A Neural Network Tool for EEG Waveform Classification, Proc IEEE Symp Computer Based Medical Systems, Minneapolis, June 1989.

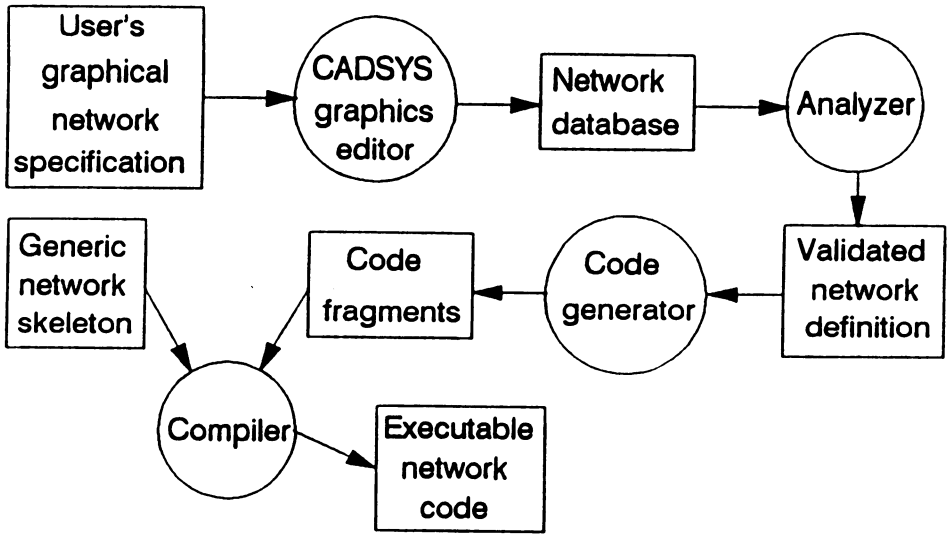


Figure 1. CASENET components.

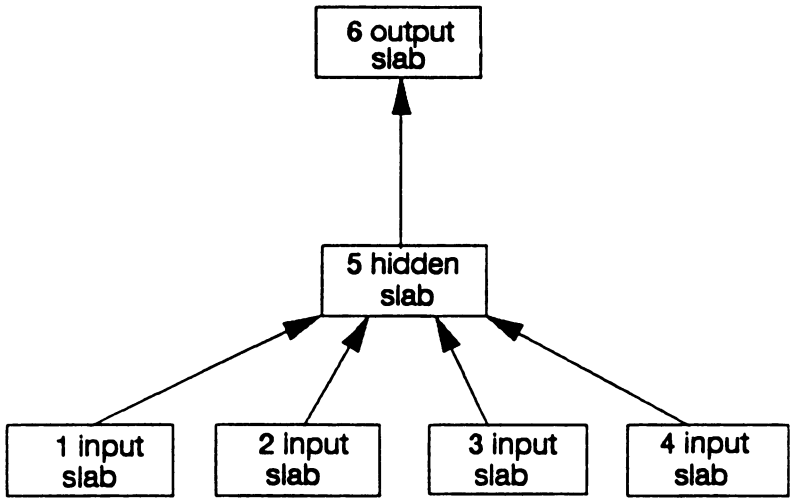


Figure 2. Generic CASENET neural network structure. Attributes specified for each slab of a backpropagation network include number of nodes, eta and alpha.

Adaptive Analog MOS Neural-Type Junction*

N. El-Leithy*, R. W. Newcomb*, M. E. Zaghloul**

*Microsystems Laboratory
Electrical Engineering Department
University of Maryland
College Park, MD 20742 USA
(301) 454-6869

**Department of Electrical Engineering and Computer Science
George Washington University
Washington, DC 20052
(202) 994-3772

ABSTRACT

Using the CMOS threshold adjust circuit recently presented in [1], this paper develops an adaptive analog neural-type junction. The junction itself is based upon the circuits presented in [2] and works according to charge control theory discussed in [2].

DEVELOPMENT

Current research trends concentrate upon means of adapting weights in analog neural networks. The main problem facing hardware designers is the feasibility of large scale implementations when adaptation is incorporated. Here we present a system which readily lends itself to simple VLSI implementation.

Figure 1 shows the input weighting stage including threshold adjustment where V_c is the threshold adjust parameter. This weighting stage is the three terminal device labelled G-D-S. The circuit of Fig. 1 is inserted into the neural-type junction of Fig. 2 at the correspondingly labelled points. It is noted that D, G, and S represent the drain, gate, and source of the input transistor being replaced in the original junction of [2].

The threshold adjust circuit accomplishes threshold voltage adjustments in order to modify the connection weights which determine the level of transmission from one neuron's output to another neuron's input. As discussed in the paper of 1987 [1], double gate MOS structures can be used to capture the adaptive properties of biological neurons. However, in present day technology the double gate transistor is relatively inconvenient to use. The circuit of Fig. 1 avoids this inconvenience by expeditiously employing the charge control concept. In Fig. 1 charge flows into the gate g and via the parasitics fixes the amount of charge on the right hand transistor. This right hand transistor acts as a normal transistor but with its characteristics controlled by V_c. This is to say, the I-V characteristics of the three terminal device labelled G-D-S are the same as those of a normal MOS transistor but shifted through the threshold adjust parameter V_c.

---

* Research supported by NSF Grant MIP 87-19886
** Research supported by NSF Grant MIP 88-08292

References:

[1]. N. El-Leithy, M. E. Zaghloul, and R. W. Newcomb, "CMOS Circuit for MOS Transistor Threshold Adjustment: A Means for Neural Network Weight Adjustment," Proceedings of the 1989 IEEE International Symposium on Circuits and Systems, Portland, OR, May 1989, pp. 1221 - 1222.

[2]. N. El-Leithy, R. W. Newcomb, and M. Zaghloul, "A Basic MOS Neural-Type Junction; A Perspective on Neural-Type Microsystems," Proceedings of the IEEE First International Conference on Neural Networks, San Diego, CA, June 1987, pp. III-469 - III-477.

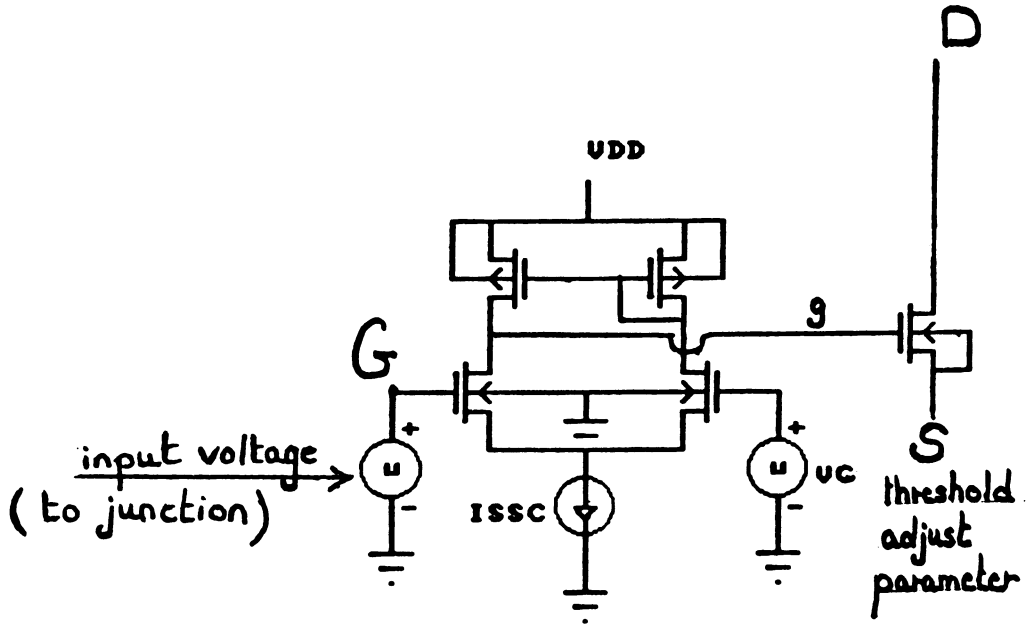
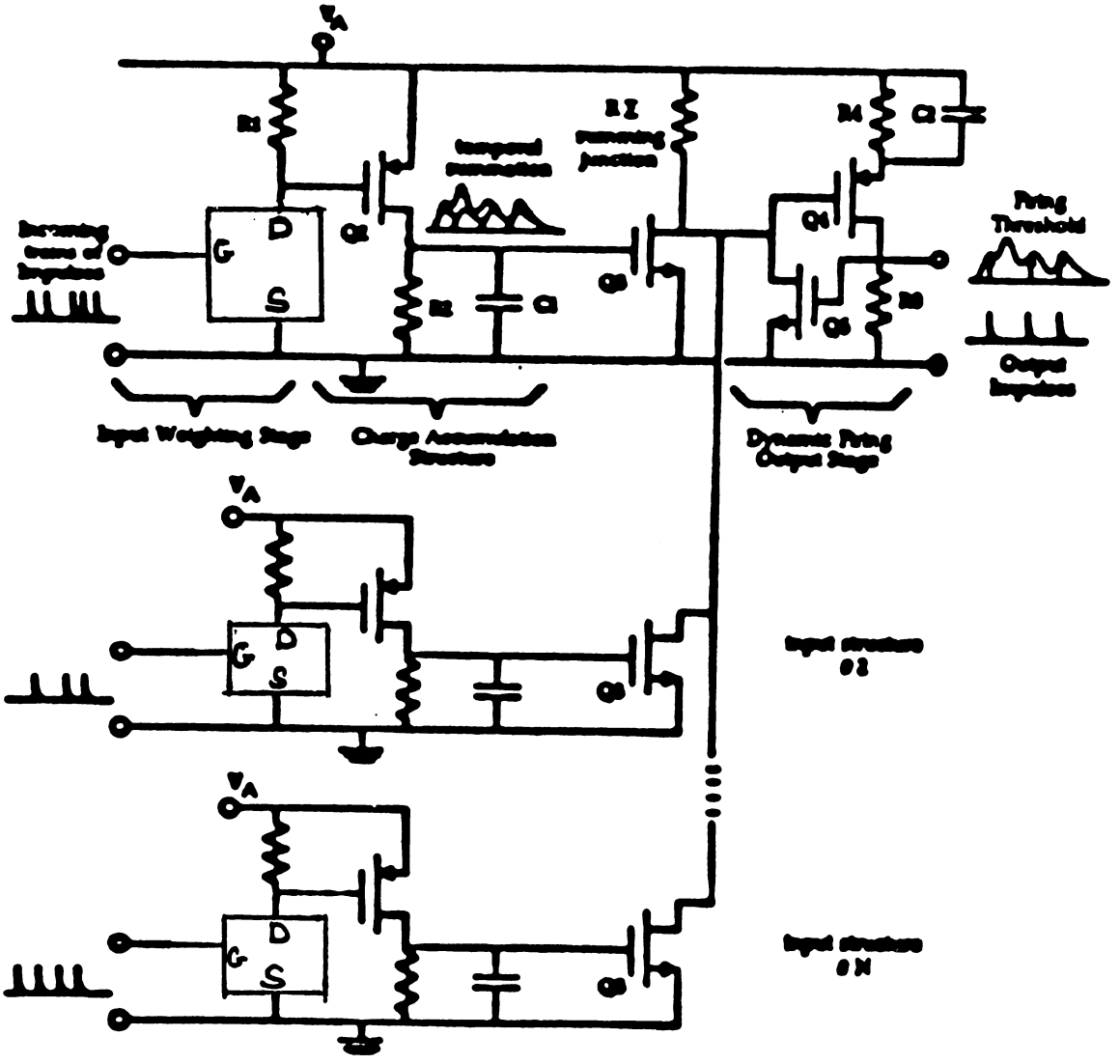


FIGURE 1
THRESHOLD ADJUST CIRCUIT

FIGURE 2.

A MOS NEURAL-TYPE JUNCTION (NTJ)

THE KEY SIGNAL PROCESSOR CAPTURING BOTH SPATIAL AND TEMPORAL ACTIVITY



An Optoelectronic Interconnection Scheme for Neural Networks†

David Yu-Shan Fong
Digital Image Processing Laboratory
Lockheed Missiles and Space Company
Palo Alto, CA 94304-1191

Christopher Tocci
Raytheon Company
Equipment Division
Marlborough, MA 01752

Abstract

This paper describes an interconnection scheme for the neural network which is implementable in current technology. The connection is made by projecting a light pattern onto a photoconductive layer (PCL), thus creating low-resistive paths on the layer. The connection pattern can be changed by modifying the LCD mask positioned between the panel light source and the PCL. The proposed implementation is extremely flexible and highly expandable, consumes very little power, and is insensitive to manufacturing variations.

Introduction

Two major implementable realities required by neural network paradigms, whether supervised or unsupervised, are a learning model/algorithm and a massive interconnection scheme. To date the implementations of artificial neural networks are mainly confined to small scales using software simulation because the implementation of the interconnection scheme has been regarded as a bottleneck which prevents the large scale realization of the neural network in hardware. The basic problems include a large number of paths, large fan-in and fan-out requirements, the variability of the weights, the training procedures, and the power consumption.

This paper attempts to address the issue of the interconnection scheme. We propose an implementation which is flexible, expandable, and feasible under the current technology. In addition, it is highly insensitive to manufacturing variations and requires relatively low power consumption.

The basic construction consists of a plasma light source, a pixel-addressable spatial light modulator (LCD mask), and a photoconductive layer (PCL). The light source, when modulated by the mask, projects a light pattern on the PCL. The light on the photoconductive layer is absorbed by the photoconductive material, and generates carriers in the PCL. When external electric field is applied to the layer, electric current is produced by the motion of the carriers, and a connection between two points is established.

The Photoconductive Neural Network Interconnect

Most of the electronic implementations of the modifiable synaptic weight have been evaluated and reported [1,2,3] using lumped circuit approaches. But according to the Defence Advanced Research Projects Agency (DARPA) Report on Neural Networks [4], the potential solution to the interconnection problem is in optics and electro-optics. It offers the advantages of higher speed and higher density of the interconnection process over the electronic (silicon or galium arsenide) implementations.

Figure 1 shows the construction of the proposed Photoconductive Neural Network In-

†Part of the work reported in this paper was carried out in the ECE Department, Clarkson University, Potsdam, New York.

terconnect (PNNI) scheme which utilizes conventional optical and electronic hardware. A simple planar light source illuminates a photoconductive layer. This illumination is modulated by a planar spatial light modulator. This construction allows any arbitrary two-dimensional interconnection pattern to exist on the PCL. Furthermore, if the SLM is allowed to have shades of gray (as opposed to only the binary settings of opaque and translucent), we have another degree of control over the connection strength.

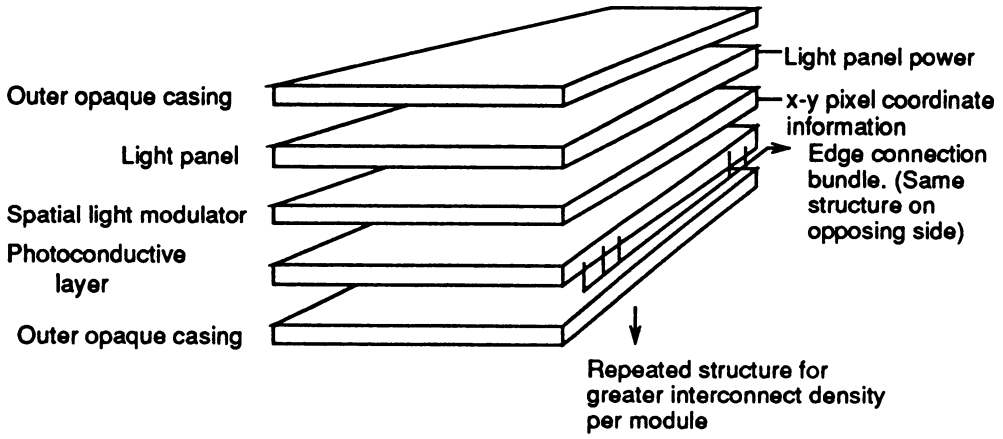


Figure 1.Photoconductive neural network interconnect (PNNI) structure

The edge connections along the edges of the PCL are simple wire bonds. These wire bonds could either be brought out or connected to a string of internal thin film integrated resistors to form the passive summation (see Figure 2). No implicit interconnection paradigm is assumed here. Only the generally required massive interconnection needed in any neural network implementation is considered.

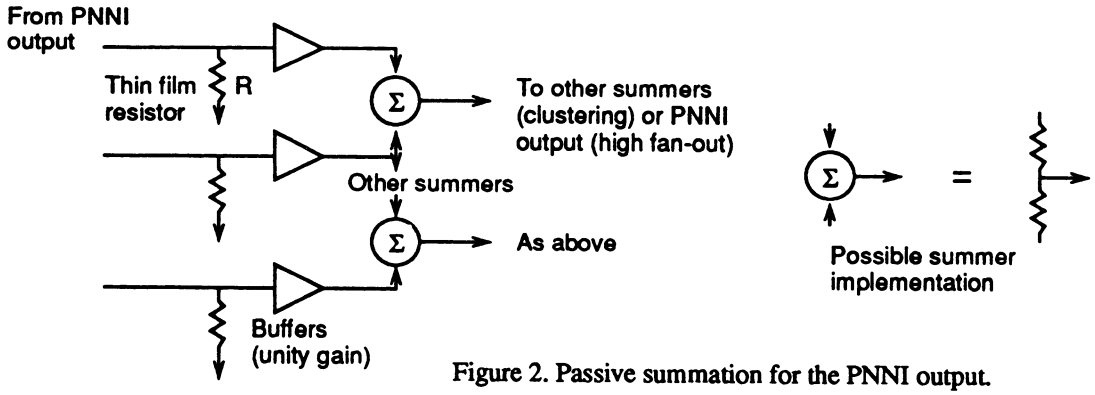


Figure 2. Passive summation for the PNNI output.

The light source, the SLM and the PCL are contained between two outer opaque casings. This structure could be repeated several times, thereby greatly increasing the interconnection density on a per-module basis.

Analysis of the Photoconductive Layer

When light is absorbed by the photoconductive material, electron-hole pairs are generated. This happens because the light quanta $h\omega \geq \varepsilon_G$, where h is the Planck constant, ω is

the wavelength of the light, and ε_G is the gap energy of the material.

The *injected* pairs will gradually diffuse throughout the material. If we assume that we have a point light source projected onto the surface of the material for a short time, the concentration of the carrier can be characterized by a Gaussian-shape function under the point source. Since equal amount of electrons and holes are injected, the differential increase in the carrier is much more prominent in the minority carriers. In the following description, we will consider the minority carriers only.

When we applied an external electric field E_0 on the material, we have both the diffusion and drifting of the minority carriers. The differential concentration of carriers can be described by [5]:

$$\Delta n = \frac{\Delta n_0 \exp(-t/\tau_n) \exp(-x'^2/4Dt)}{2\sqrt{\pi Dt}}$$

where Δn_0 is the number of injected pairs per sample cross section, D is the ambipolar diffusion coefficient, x' is defined to be $x-\mu E_0 t$, in which μ is the ambipolar drift mobility, and τ_n is the lifetime of the carrier. This equation shows that while the carriers diffuse, the pattern is drifting in the direction of the electric field. Thus if we have a path not totally in line with the field, the pattern will translate in the direction of the field.

To control the path width, we need to control the width of the Gaussian-shape function. The parameter related to this *is D*. The half-width (full width at half maximum) of the Gaussian-shape function is

$$\Delta_{1/2} = 4\sqrt{Dt \ln 2} = 3.33\sqrt{Dt} \ .$$

Suppose the light source is steady, then the injection of electron-hole pairs continues. A steady state of the path width may be reached. More analysis is needed to determine this steady state path width and the optimum energy input from the light. Since both doping concentration and material will affect *D*, the effective path width depends on the material used in the photoconductive layer.

Spatial Light Modulator

The spatial light modulator is a pixel-addressable LCD mask. On the mask patterns can be created so that specific paths between the edge connectors can be formed in the PCL. Initially, the conventional nematic type LCD could be used because it is inexpensive, readily available and of low power usage. It also has numerous 'off-the-shelf' geometries available. The contrast ratios are in the order of 100:1[6].

To avoid the problem of cross-over paths, the SLM patterns can be time-multiplexed. The response time of the nematic-type LCD is approximately 25 msec [6]. If faster SLM response times are desired, the use of a ferroelectric LCD can be utilized. The ferroelectric LCD has its opacity related only to the electric field applied across the device. However, speed in this initial study is not as important an issue as the development of a modifiable 2-D interconnect scheme in the PCL.

A trade-off also exists between the photoconductive diffusion and optimum SLM resolution. The nematic-type LCD has a medium resolution of 20 lines/mm [6]. When the development of a 1-2 mil wire bonding process is considered in the future, the issue of LCD resolution will become important. But at this moment, depending on the neural network para-

digm and training algorithm used, a sparse interconnection scheme may be sufficient, thus obviating the need for higher cost PCL and SLM devices.

Discussion

The proposed scheme is a first attempt to ease the bottleneck which prevents the large scale implementation of neural networks. It offers some immediate advantages:

1) Simple construction with existing technology
2) Extremely easy to expand -- the interconnection density is doubled just by repeating the interconnect construction layers vertically
3) Low power consumption -- the only major power needed is in the light panel source
4) Highly insensitive to manufacturing variations in processing
5) The binary bit-map of the SLM patterns can be stored in conventional digital formats.

A number of issues need to be addressed before this scheme can be fully implemented. When a connection network is specified by the participating network in either the training stage or the operation stage, the specific paths between the two banks of connectors representing two layers have to be computed. Since crossover of the paths is not allowed, scheduling of paths in the time-multiplexing scheme is the major task in the path computation. The covariant neighboring effects [6,7,8,9] observed in the biological synapse interconnections in the cortex may prove to be effective in reducing the crossovers needed, and thus reducing the computing time. Finally, the effects of this interconnection scheme on the learning algorithms have to be investigated.

References

[1] M. Mahowald and T. Delbrück: 'An analog VLSI implementation of the Marr-Poggio stereo correspondence algorithm', *Abstracts of the First Annual INNS Meeting*, Boston, MA , 1988.

[2] B. Nabet and R.B. Darling:'Implementation of optical sensory neural networks by simple discrete and monolithic circuits', ob. cit.

[3] E.A. Reitman, R.C. Frye, C.C. Wong, and C.D. Kornfeld: 'Properties of amorphous silicon photoconductive synapses', ob. cit.

[4] *DARPA Neural Network Study,* Oct. 1987 - Feb. 1988, *Final Report,* M.I.T. Lincoln Laboratory, Lexington, MA 1988.

[5] K. Seeger: *Semiconductor Physics, and Introduction,* Third Edition, Springer Series in Solid-State Sciences, Vol. 40, Springer-Verlag, New York 1985.

[6] J.L. Horner (Editor): *Optical Signal Processing,* pp. 477-518, Academic Press, 1987..

[7] C.L. Giles, R.D. Griffin, and T. Maxwell, ob. cit.

[8] C.L. Giles and T. Maxwell: 'Encoding invariances in higher order neural networks', *Proceedings of the IEEE Conference on Neural Information Processing Systems - Natural and Synthetic,* Denver, Colorado, Nov. 1987.

[9] J.S. Morgan, E.C. Patterson, and A.H. Klopf: 'A network of two drive-reinforcement neurons that learns a solution to a real-time dynamic control problem', *Abstracts of the First Annual INNS Meeting,* Boston, MA 1988.

Simulation of Artificial Neural Network Models Using an Object-Oriented Software Paradigm

†Gregory L. Heileman ‡Harold K. Brown ‡Michael Georgiopoulos
†Department of Computer Engineering
‡Department of Electrical Engineering
University of Central Florida, Orlando, FL 32816

Abstract

A highly flexible simulation environment based on object-oriented software techniques was developed and successfully used to implement and test novel neural network models. The architecture of this system is based upon the objects the simulation is intended to manipulate — the neurons in the network. This is in contrast to the prevailing procedure-oriented approach in which the software architecture is based upon the functions the system is intended to perform. The benefits obtained from using such an approach, which are presented in this paper, are a direct result of the methods used to encapsulate the data structures. This method of data encapsulation allows a flexible system architecture to be constructed in which one can easily extend or modify the functionality of the software product to simulate any neural network model.

1 Introduction

Artificial neural network research is largely dependent upon the use of computer simulation. However, the appropriate tools for the development and simulation of neural network models and learning algorithms are currently lacking. A recent study commissioned by the Defense Advanced Research Projects Agency (DARPA) [1] found this issue to be of critical importance to the development of neural network theory. This report stated that much of the research in the field is inhibited by the lack of inexpensive and easily accessible simulation facilities.

Recently, a number of neural network simulation systems have been developed that make use of object-oriented programming techniques [2,3]. There are a number of benefits obtained from using such an approach. The method of data encapsulation provided by an object-oriented language allows the development of a flexible simulation environment in which one can easily extend or modify the functionality of a neural network model. Furthermore, this extension can be accomplished without knowledge of the implementation details of the original neural network software product. The manner in which these benefits can be achieved are presented in this paper. A brief introduction of the fundamental concepts involved in object-oriented programming, and the philosophy of their use are discussed. In addition, the ability to specify and simulate arbitrary neural network models using these concepts is addressed. It should also be mentioned that the method of communication used by the objects of an object-oriented neural network software system facilitates their mapping onto parallel processing hardware. This is an area in which we are actively pursuing research. However, the topic of implementing these simulations on parallel computer architectures is not addressed here.

2 Object-Oriented Programming Languages

The design of an artificial neural network software simulation involves a mapping of the objects and actions occurring in the problem domain to a corresponding set of operations on data in the computer domain of the software system. In the design of such a system, a decision must be made whether to structure the software architecture around the *functions* the system is intended to perform or the *objects* that these functions manipulate. The classic design approach is based upon the decomposition of the functional requirements of the system. This is embodied in the practice of top-down functional design. It is widely acknowledged that the major difficulty encountered using this approach involves dealing with changes or modifications to the system [4,5,6]. That is, the ability to evolve into a more useful product is not inherent in systems designed using classical software design methodology. Object-oriented design, on the other hand, affords the opportunity to greatly reduce the difficulties involved with the introduction of change. It bases the system architecture on the classes of data (i.e., objects) the system manipulates as opposed to the functions the

system is required to perform. The rationale for this approach follows from the observation that as software system requirements change or evolve, the functions that the system performs may change drastically; however, the classes of data that the system manipulates tend to remain much more stable [6].

In the object-oriented design of a software system, one strives towards retaining as much flexibility as possible in the system architecture. The goal of this approach is to allow the software system to be easily extended to improve its functionality, or reused in other systems that require its services. Ideally, the extension or reusability of the software product does not require a knowledge of the details of system implementation. The ability to develop software in this manner enables software components to be packaged in such a way that others are able to modify and incorporate them into their products as needed. This ease of reusability is currently lacking in traditional software technology.

A *class* in an object-oriented system is considered a means of packaging the implementation of an abstract data type. A class contains all the information necessary to construct separate instances of itself, these instances are the *objects* that the software system will manipulate. If the objects are chosen to represent objects in the real-world (e.g., the neurons in the network) a more natural mapping of the problem domain to the software system can be obtained.

Every object contains its own set of data elements, *instance variables*, that determine the individual state of that object. In addition, a class may store information that is common to all instances of the class in *class variables*. The instance variables are private to the object and may only be accessed through the accessing routines provided by the class. The class also provides accessing routines for manipulating the class variables shared by all objects of a particular class. In the terminology of object-oriented systems, these accessing routines are called *methods*.

An object-oriented programming language must support *inheritance*. This concept enhances the extendibility of the system as well as the reusability of system components. Inheritance gives the user the ability to create a new class that is an extension or specialization of an existing class by simply specifying the differences between the new class and the existing class. In this case, the new class is said to be an *ancestor* of the class from which it was derived, the *parent* class. Many object-oriented programming languages also support *multiple inheritance* in which a derived class can inherit more than one parent class.

An ancestor inherits instance variables, class variables and methods from the parent class. The ancestor may also add new instance variables, class variables and methods that are necessary for its specialized functions. Additionally, an ancestor of a class may redefine any method provided by the parent class by simply supplying a new method that has the same name as the old method in the parent class. In this case, the new method in the derived class is said to *overload* the method with the corresponding name in the parent class. This allows different meanings to be attached to the same method name; which method is invoked when the name is called in a program depends upon the class being used at that particular time.

Computation in an object-oriented system centers around *messages*. Objects in the system manipulate other objects by sending messages requesting them to perform specific actions. These messages invoke the appropriate methods in the object's class, or possibly a method in a class from which the object is derived. This model of computation maps well to loosely coupled multiprocessor computer systems.

There is a fundamental distinction between message passing and the conventional procedure calls used in procedure-oriented systems. A message can be viewed as a request of an object to perform some action. How the object responds to this request depends upon available methods. This approach allows objects from different classes to respond appropriately to the same messages, a trait known as *polymorphism*. When an object receives a message, it is up to the object to decide what to do. If a new response needs to be added to the system, then an appropriate method can be incorporated into the system while inheriting a class. An object of this derived class will now use the new method if an applicable message is received. The important aspect of this approach is that the original code does not have to be modified. Therefore, an addition to the system requires just that — addition and not modification [4].

Using traditional software technology, the developer is responsible for system modification. By allowing the objects to determine how a message should be interpreted, the responsibility of implementing system modifications and additions can be shifted away from the developer of a class to the user of a class. Such a fundamental change in focus supports the notion that software can be developed and packaged for later use just as hardware components are packaged for convenient use in integrated circuits (ICs) [4].

3 Neural Network Simulation

As mentioned previously, artificial neural network research is largely dependent upon the use of computer simulation. Consequently, numerous neural network simulation languages and environments have been developed to help meet the needs of researchers in this field [7,8]. These simulation systems require the user to specify the neural network architecture and learning rule through the use of a *network specification language* which is then translated into a lower-level implementation language (e.g., FORTRAN, C or LISP). The specification language normally allows one to simulate some of the more popular neural network models; however, if the user wishes to simulate a novel learning algorithm or activation function, he must provide the system with a procedure written in the appropriate implementation language that performs the desired function. In order to do this, the user must be familiar with the implementation details of the network components being simulated.

However, if an object-oriented approach is used, the developer is able to supply the user with a generic base class patterned after a general neural network model. This base class may then form the framework for simulations of specific neural network models. The details of the particular architecture or learning algorithm can be incorporated into the system through the use of inheritance. Instead of writing a new procedure using a low-level implementation language, the user can add additional functionality to a neural network simulation through the incorporation of new methods defined in terms of the high-level methods provided by the base class or classes derived from the base class. Therefore, the inheritance mechanism now allows system modification to be accomplished in a more abstract fashion. Such an approach was used to develop a general purpose neural network base class using the C++ programming language. This base class provided the scaffolding upon which more complex neural network models were built. The hierarchy of classes created through the use of inheritance is shown in Figure 1. In each case, a new model was specified by inheriting a previously developed class while providing methods necessary for the specialized functions of the new neural network model. For example, the Hopfield [9], back-propagation [10] and ART [11] networks were all simulated by first inheriting base class, and then adding specialized methods that implement the node activation functions and learning rules required by the specific model. These additions were accomplished in a straight forward manner through the use of the methods provided by base class. In addition, a network that implements the delta-bar-delta learning rule [12] was simulated by inheriting the back-prop class while providing a new method for implementing the weight update rule required by that algorithm. The classes shown in Figure 1 represent a small portion of the neural network models simulated. A number of novel models were also simulated using some of these classes as a starting point.

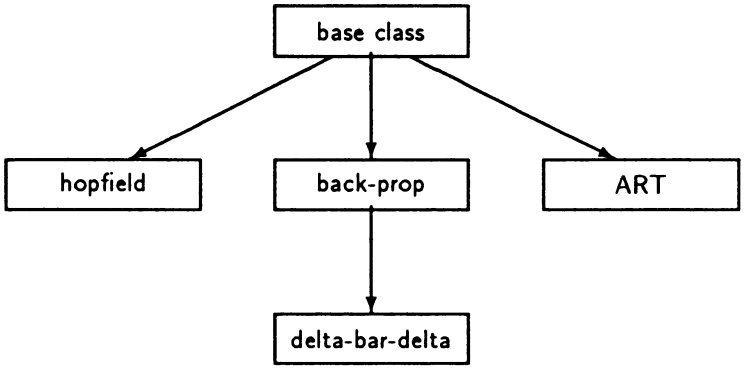


Figure 1: Hierarchy of object-oriented neural network classes created.

4 Conclusion

The recent resurgence in neural network research has resulted in the development of a wide variety of neural network models. All of these models are similar in that each contains a collection of nodes connected via adjustable weights. The issues that differentiate the various models are: the manner in which the nodes are interconnected, what type of function each node computes, and how the network weights are adjusted. An object-oriented design allows the software simulation system to capture these similarities while maintaining the ability to easily add distinguishing network properties. The significant contribution of this approach is the manner in which these distinguishing properties are added to the system. The ability to inherit the functionality of a generic neural network base class circumvents the need for writing the low-level routines that directly manipulate the computer representations of the network elements. Thus, developers of a novel neural network models do not have to understand the complex issues involved in implementing these routines — they only must understand what the routines do. This object-oriented approach allows the design of specialized neural network models to proceed from a higher level of abstraction than is possible in simulation systems using a procedure-oriented methodology. This gives neural network researchers the ability to rapidly implement and test new ideas.

Acknowledgements

This research was supported in part by grants from PM Trade (contract #N61339-88-G-0002 order 0009) and The Institute of Simulation and Training at the University of Central of Florida.

References

[1] "DARPA neural network study," B. Widrow, Study Director. AFCEA International Press, 1988.

[2] G. L. Heileman, H. R. Myler, and M. Georgiopoulos, "An object-oriented approach to the simulation of artificial neural networks," in *Progress in Simulation*, (G. W. Zobrist, ed.), Ablex Publishing Corp., 1990.

[3] T. Kraft, "Anspec language definition," Tech. Rep., Science Applications International Corporation, August 1987.

[4] B. J. Cox, *Object-oriented Programming: An Evolutionary Approach.* Reading, MA.: Addison-Wesley, 1986.

[5] R. E. Fairley, *Software Engineering Concepts.* New York: McGraw-Hill, 1985.

[6] B. Meyer, *Object-oriented Software Construction.* Englewood Cliffs, N.J.: Prentice Hall, 1988.

[7] C. L. D'Autrechy, J. A. Reggia, G. G. Sutton III, and S. M. Goodall, "A general-purpose simulation environment for developing connectionist models," *Simulation*, vol. 51, no. 1, pp. 5–19, 1988.

[8] D. Zipser and D. Rabin, "P3: A parallel network simulating system," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, (D. E. Rumelhart and J. L. McClelland, eds.), Cambridge, MA.: MIT Press, 1986.

[9] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Science USA*, vol. 79, pp. 2554–2558, 1982.

[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representation by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, (D. E. Rumelhart and J. L. McClelland, eds.), Cambridge, MA.: MIT Press, 1986.

[11] G. A. Carpenter and 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.

[12] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, vol. 1, no. 4, pp. 295–307, 1988.

A two level pipeline RISC processor array for ANN

Atsunobu Hiraiwa
Shigeru Kurosu
Shigeru Arisawa
Makoto Inoue

Sony Corporate Research Laboratories
Sony Corp.,4-14-1,Asahi-cho,Atsugi-shi,Kanagawa-ken,243,Japan

ABSTRACT

We describe mapping a fast back-propagation algorithm[1] onto a GCN(Giga CoNnection) and the architecture of our GCN.
This architecture is a two level pipeline processor array. In the first level, a Cray_1 like intra-chip pipeline is used and in the second level,the inter-chip pipeline is a systolic (Wavefront) array[2][3][4] with asynchronized communication.
We propose the Net-Data Partition method which is tailored to the mesh array processors and Copy Network method which reduces inter chip communication cost. These methods improved the performance of a large multi_processor array and just fitted for the back-propagation algorithm of large scale network used for learning .
From results of our computer simulation, we get one Giga connection per second using a one hundred twenty eight processor system (GCN-128).

1. INTRODUCTION

The artificial Neural Nets (ANN) model is useful in a number of applications, such as speech recognition, image processing, robot control, etc.
In order to improve ANN capability, many learning data and a large scale network are demanded.
First we describe the architecture of the GCN and how the back-propagation algorithm can be mapped onto the GCN which has mesh interconnected PEs (Processing Elements). Then we describe the performance of the GCN which was obtained by executing a back-propagation program on software simulator.
Finally planned extensions and improvements to our GCN system are discussed.

2. THE ARCHITECTURE OF GCN

Fig.1 shows the mesh architecture of our GCN. Each PE has an 80860, local memory (4 Mega byte), and 2 FIFOs(first in first out memory, 64bit x 256W) for mesh connection. The 80860 is designed by Intel Corp. as a 64bit General purpose RISC processor. The 80860 obtains high performance with three parallel units (integer core, 32/64bit floating-point multiplier and adder), pipelined processing mode, 64bit external and 128bit internal data busses, 12kbyte large on-chip data/instruction caches, pipelined external memory access, and a full customized VLSI chip using CMOS 1.0 μm technology.

A our GCN is a two level pipeline processor array.

In the first level, the intra-chip pipeline is implemented by the 80860's pipeline mode which floating point operations can be executed in a three stage pipeline units.

In the second level, the inter-chip pipeline is a Systolic array like Warp [2].

Since the adjacent PE can be asynchronously communicated to using a very high bandwidth FIFO (160M bytes/sec,if cpu clock is 40MHz), the GCN avoids synchronous overhead. Then the 80860's dual instruction can execute FIFO load/store instructions and floating-point instruction in parallel.

The local memory is used for storing weights, data, and intermediate results.

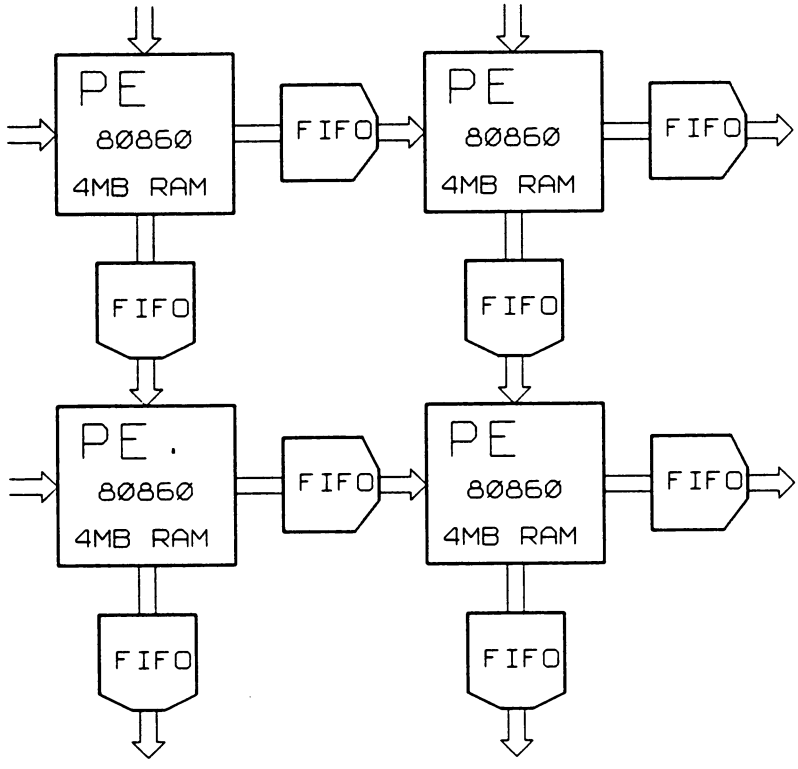


Fig 1 The architecture of PE

3. MAPPING A BACK-PROPAGATION ALGORITHM ONTO THE GCN

In systems over one hundred processors, the linear ring connection has a large communication time, while the mesh connection has a much smaller communication time.

Fig.2 shows the Net-Data partition method which maps the back-propagation algorithm onto the GCN.

In the vertical ring, each PE is used for the Network partition [5], has different weights, and transfers hidden output values and intermediate results for delta hidden at every forward and backward computation.

In the horizontal ring, each PE is used for the Data partition [2], has different data and same copied weights to reduce communication time (we call this the Copy method), and transfers delta weights and updated weights at every cycle of all data presentations.

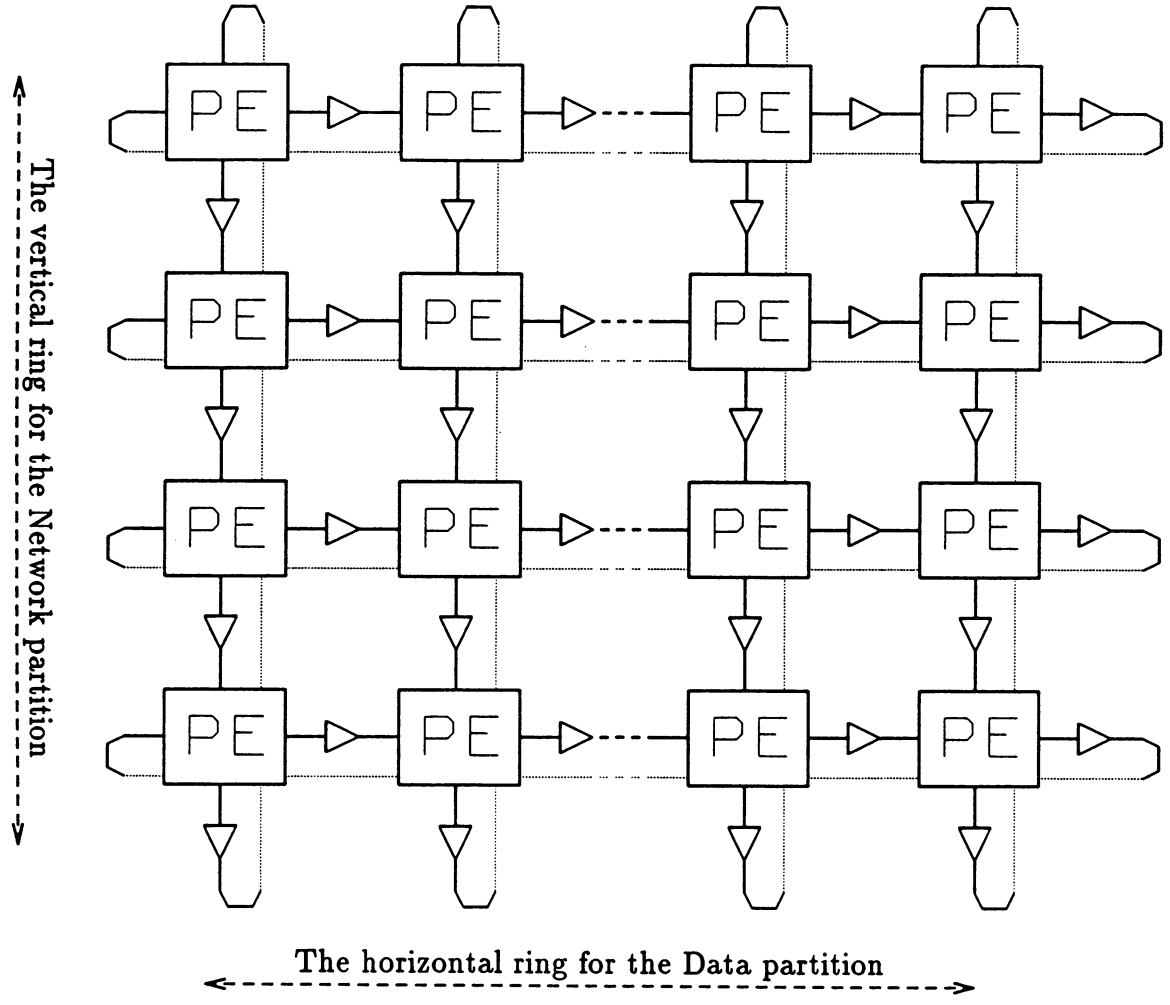


Fig 2 The Net-Data partition on the mesh connected GCN.

4. SIMULATION OF PERFORMANCE

Speed evaluation of our GCN was obtained by executing back-propagation on the Sim860 which is a software simulator of the 80860.
In the three layered Neural-Net, the input layer has 256 units, the hidden layer has 80 units,and the output layer has 32 units. There are 5120 input data points and the GCN has 128 PEs. Each PE executes calculator with 32bits of floating point accuracy. The network is partitioned into four parts and data is distributed into 32 PE groups.
We get exactly 6.4 machine cycles required per connection in each PE.
This includes the forward and backward computation time and the communication overhead.
If the clock of each PE is 20ns (50 MHZ),the speed of the GCN-128 will be over one giga connection per second.

5. CONCLUSION

We conclude that two level pipeline RISC processors in a mesh connection configuration can obtain good performance in a large scale Neural Network simulation.
Our next plan of the GCN is to integrate each node (core like 80860 ,2 FIFOs,4Mbyte Memory) into super_chip by 1992.

References

[1] D.E. Rumelhart, G.E. Hinton and R.J. Williams : Learning Internal Representations by Error Propagation. In Parallel Distributed Processing, Vol.1, pp.318-363, 1986.

[2] D.A. Pomerleau, G.L. Gusciora, D.S. Touretoky and H.T. Kung : Neural Network Simulation at Warp Speed : How we got 17Million Connection Per Second. ICNN88,Vol 2,pp143-150,1988.

[3] S.Y Kung,J.H. Hwang : Parallel Architectures for Artifical Neural Nets, ICNN88, Vol 2,pp165-172,1988.

[4] S.Y Kung :VLSI Array Processors.,Prentice-Hall Inc.,1988.

[5] A. Iwata, Y. Yoshida, S. Matsuda, Y. Sato and N. Suzumura : An Artificial Neural Network Accelator using Genral purpose 24 bits Floating Point Digital Signal Processors.,IJCNN89, Vol 2,pp171-175,1989.

Parallelizing the Self-Organizing Feature Map on Multi-Processor Systems

Russel E. Hodges*, Chwan-Hwa Wu* and C.-J. Wang**

* Department Electrical Engineering, Auburn University, Auburn, AL 36849

** Department of Electrical Engineering, University of Colorado at Colorado Springs, Colorado Springs, Co 80933-7150

ABSTRACT

Parallel algorithms for implementing the Kohonen Self-Organizing Feature Map (SOFM) are presented in this paper. The algorithms can maintain a high degree of load balancing and minimize the communication overhead. We have implemented the algorithms on a linear chain and a two-dimensional mesh of transputers. Significant speedup has been achieved. In addition, models to describe the performance of the algorithms are also presented. The performance of massively parallel parallel processing systems is predicted from the models.

INTRODUCTION

Kohonen's Self-Organizing Feature Map (SOFM) is a reliable and widely accepted vector quantization algorithm. A drawback to Kohonen's algorithm, is the increase in computation time associated with an increase in the number of node elements. Depending on the particular hardware arrangement on which the Kohonen algorithm is implemented, there is a maximum number of nodes which can be used for real-time vector quantization.

A possible technique that can be used to overcome this drawback is to use a multi-processor machine[3-5]. The nodes of the SOFM can be distributed across the processors with the corresponding computations likewise distributed. The multi-processor scheme allows more node elements to be added to the SOFM while maintaining the real-time vector processing. A sufficient number of processors can be added to ensure real time vector quantization with the addition of the extra nodes.

The goal of this paper is to develop parallel algorithms for the SOFM so that it can be adapted to any number of processors running simultaneously. For the modification of Kohonens's algorithm two, guidelines will be followed. These are :

1.) Develop an algorithm to maintain a high degree of load balancing

2.) Keep communication time to a minimum.

The structure of the SOFM makes it easily adaptable to a parallel processing environment. The nodes can be divided equally among the processors of the system. Each processor is responsible for updating the weights connecting the input nodes with the corresponding set of output nodes.

IMPLEMENTATION OF THE ALGORITHM

Linear Processor Arrangement

The first implementation of Kohonen's algorithm is carried out on the processor arrangement shown in figure 1. A node arrangement that maintains a high percentage of load balancing is shown in figure 2.

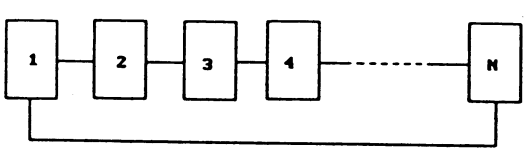


fig. 1. Linear processor chain

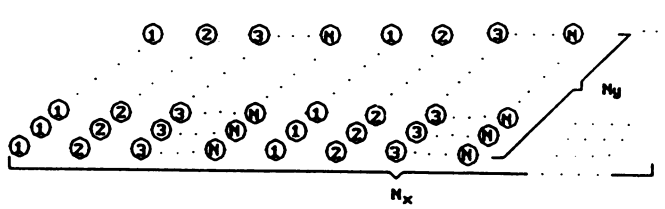


fig. 2. Node arrangement for figure 2

The numbers within the nodes correspond to the particular processor to which the node is assigned. This node arrangement allows for a high percent of load balancing while minimizing

communication time. One-hundred percent load balancing is maintained while the neighborhood radius is greater than or equal to N_p. As the neighborhood size decreases below N_p the percent of load balancing is approximately $\{(2[R(k)]+1)/N_p\} \times 100\%$, where [R(k)] is the integer part of the neighborhood radius for iteration number k. When a training vector is applied to the processor chain, each processor calculates the minimum node distance and determines the coordinates of this minimum distance node in the complete feature map. The coordinates of this node are determined by a mapping array that is generated by each processor during the initialization phase.

Once each processor has calculated the minimum node distance and determined the node coordinates, a global minimum must be determined. A global minimum can be determined by rotating each set of values N times clockwise or counter-clockwise allowing each processor to have a copy of these values to determine the minimum node. The time required for communication is therefore independent of the number of nodes and depends only on the number of processors in the chain and the number of iterations for each training vector. The computation time per processor will be a function of the number of nodes per processor and the number of iterations per training sample. The computation time should therefore decrease as $1/N_p$. The communication time increases as the number of processors and the number of iterations per training sample. The total time to process a training sample can be derived as follows :

The time required to process a training sample can be broken down into the sum of a computation time and a communication time. The computation time can be divided into three parts; a time required to determine the closest node to the present input; a time required to determine the new neighborhood, and a time required for updating. Communication is carried on only while $R(t) \geq 1$. Using a rotation method to establish a global minimum node requires the data to be rotated around the processor chain twice to ensure each processor has a copy of the necessary data.

Define the following values :

t_m = time required for a multiplication

t_a = time required for an addition or subtraction

t_s = time required to evaluate the expression $C^{1/2}$ for some constant C

N_I = number of input nodes

n_I = total number of iterations per training sample

T_m = time to transfer 12 bytes of information between processors

T_{TOTAL} = total time to process an input vector

N_x = Number of nodes in the x-direction

N_y = Number of nodes in the y-direction

$T_1^* = (2N_I + 3)t_a + (N_I + 2)t_m + t_s$

$T_2^* = 2N_I t_a + N_I t_m$

γ = Neighborhood contraction rate

The total time required to process a training sample is then calculated to be,

$$T_{TOTAL} = \frac{n_I N_x N_y}{N_p} T_1^* + \frac{\pi}{N_p} R(0) T_2^* \left[\frac{1 - e^{-2\gamma n_I}}{1 - e^{-2\gamma}} \right] + (2N_p - 1) T_m n_I \quad (1)$$

It is obvious from the above formula that there is an optimal number of processors such that adding more processors to the chain results in an increase in the processing time for an input. This optimal value can be calculated by differentiating (1) with respect to N_p and setting the result equal to zero. Note that the processorchain requires that the number of nodes in the x-direction be divisible by N_p. Figure 6 shows the

results of the simulation for the processor chain.

2-D Mesh Processor Arrangement

The second implementation of Kohonen's SOFM is carried out on the 2-D processor mesh as shown in figure 3.The corresponding node arrangement is shown in figure 4.

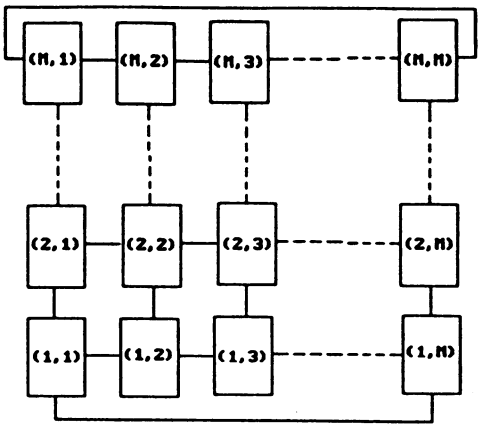


fig. 3. 2-D processor mesh

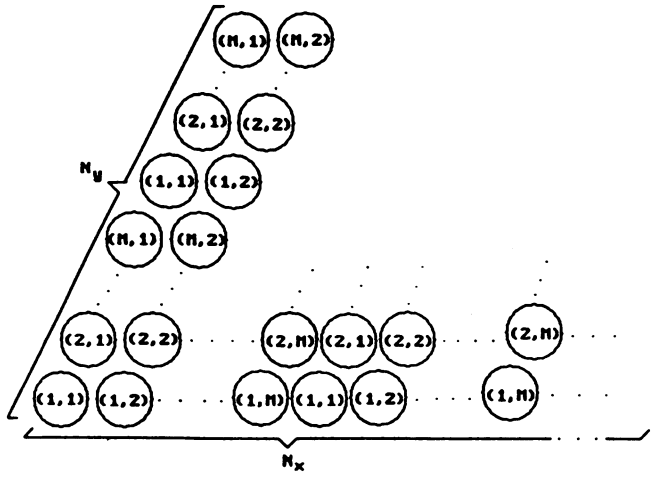


fig. 4. Node arrangement for 2-D processor mesh

The ordered pairs within the nodes correspond to the particular processor to which the node is assigned. One-hundred percent load balancing is maintained while the neighborhood radius is greater than or equal to M. When the neighborhood radius is less than M, the percent of load balancing is approximately $\{(2[R(k)]+1)/M\}\times 100\%$ for the k'th iteration.

The 2-D processor mesh allows for many more degrees of freedom in communication than the processor chain does. Similar to the processor chain, the computation time per processor will be a function of the number of nodes per processor and the number of iterations per training sample. The computation time should decrease as $M^{-2} = N_p^{-1}$. The communication time increases with the dimension of the processor mesh and with the number of iterations per training sample. The total time to process an input vector can be obtained in a similar manner as before. The result is,

$$T_{TOTAL} = \frac{n_I N_x N_y}{M^2} T_1^* + \frac{\pi}{M^2} R(0) T_2^* \left[\frac{1 - e^{-2\gamma n_I}}{1 - e^{-2\gamma}} \right] + (2M^2 - 2) T_m n_I \qquad (2)$$

As before, there is an optimal size processor mesh for implementing the SOFM. This can be calculated by differentiating (2) in a similar manner as before. For the 2-D mesh arrangement, communication time is decreased by a factor of $(2M^2-2)/(2N_p-1)$. For large n_I, the decrease in communication time becomes a noticeable effect.

The following figures show the results of the simulation for the linear processor chain and for the 2-D processor mesh for $N_x = N_y = 10$. Figure 7 shows the projected speed up for a massivley parallel implementation of the SOFM based on a 2-D model as in equation (2).Note that the model agrees well with the measured speedup in Fig. 6. Significant speedup can be observed in the figures. These results indicate that Kohonen's SOFM is suitable for parallel processing by the algorithms presented above. Moreover, on-line learning for the SOFM is feasible for practical applications.

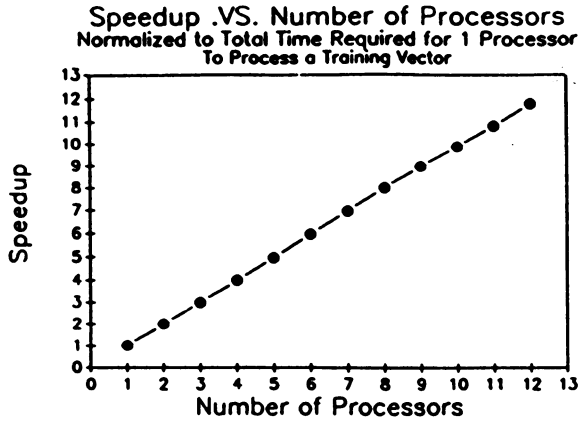


fig. 5. Speedup for the linear processor chain.

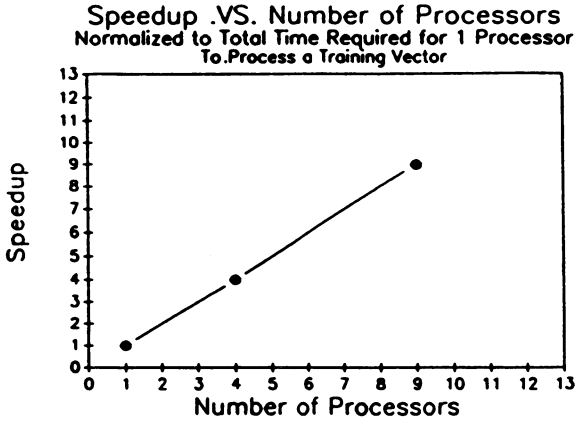


fig. 6. Speedup for the 2-D processor mesh.

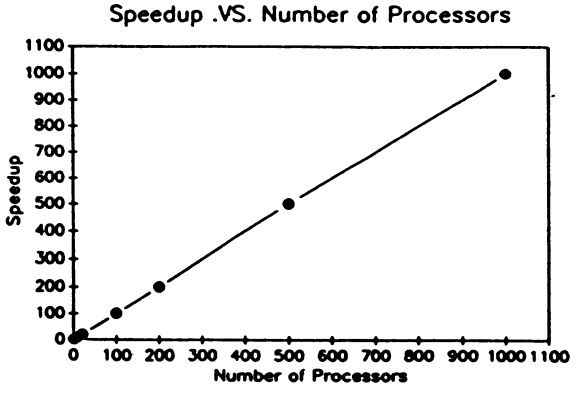


fig. 7. Projected speedup for a 1000x1000 system of nodes distributed across 1000 processors.

CONCLUDING REMARKS

There is significant speedup for the simulation of Kohonen's SOFM by the parallel algorithms presented above. The model further predicts that the algorithms are applicable to a massively parallel processing system. The algorithms optimally balance the load among processors and minimize the communication overhead. The 2-D mesh architecture has a better speed up over that of a processor chain. This indicates that a higher-performance interconnection network, such as a hypercube, is able to further increase the speedup of the parallel simulation of the SOFM.

The results have demonstrated that the large-scale real-time applications of the SOFM are feasible by utilizing the parallel algorithms.

References

1. T. Kohonen, Self-Organization and Associative Memory, Second Edition, Springer-Verlag, 1988.

2. T. Kohonen, "Self-Organized Formation of Topologically Correct Feature Maps," *Biol. Cybern.* 43, 1982, *pp* 59-69.

3. B.M. Forrest, D. Roweth, N. Stroud, D.J. Wallace and G.V. Wilson, "Implementing Neural Network Models on Parallel Computers," *The Computer Journal*, 30, 1987, pp. 413-419.

4. D.A. Pomerleau, G.L. Gusciora, D.S. Touretzky and H.T. Kung, "Neural Network Simulation at Warp Speed: How We Got 17 Million Connections Per Second, " *Proc.* IEEE ICNN 88, 1988, pp II-143-II150.

5 J. Barhen, N. Toomarian, V. Protopopescu and M. Clinard, "Concurrent Neuromorphic Algorithms for Optimization of the Computational Load of a Hypercube Supercomputer," IEEE ICNN 87, 1987, IV 687-IV 698.

OPTICAL ASSOCIATIVE PROCESSORS WITH ADAPTIVE LEARNING CAPABILITIES USING VARIABLE NONLINEARITY IN THE FOURIER DOMAIN

Bahram Javidi
University of Connecticut,
Department of Electrical and Systems Engineering
Storrs, Connecticut 06269-3157

Recently, we have introduced a nonlinear class of processors for pattern recognition, inverse filtering, and image processing [1,2]. The nonlinearity is used at the Fourier plane to threshold the joint power spectrum of the input signal and the processing function for various types of operations. The binary nonlinear optical processor has been used to implement a programmable optical associative memory for two-dimensional associative retrieval using binary spatial light modulators [3]. We have shown that binary nonlinear associative processors can produce will defined correlation signals, low cross-correlation sidelobes,. and high optical efficiency. As a result, the need for employing nonlinearity at the correlation plane may be eliminated.

The nonlinear processor has been investigated for a general type of a nonlinearity at the Fourier plane. It has been shown that the correlation signals produced by various types of filters can be produced simply by varying the severity of nonlinearity and without the need to synthesize the specific type of the filter. The discrimination sensitivity, peak size, and peak height of the processor can be adjusted by controlling the severity of the nonlinearity of the Fourier plane [4].

In this paper, we present a programmable optical processor that can perform associative retrieval using a variable nonlinearity in the Fourier domain. The nonlinearity is varied to reduce the correlation sidelobes and to improve the correlation performance at the output plane. We show that the selection of the nonlinearity is dependent on the associative signals employed in the processor. Statistical analysis of the nonlinear processor will be performed to study the effects of the Fourier domain nonlinearity on the performance of the processor. Experiments using the nonlinear associative processor for different degrees of the nonlinearities will be presented.

REFERENCES:

1. B. Javidi, "Multifunction nonlinear optical processor: correlation and convolution", *Optical Engineering*, Vol. 28, No. 8, August 1989.

2. B. Javidi and J. L. Horner, "Single spatial light modulator optical correlator, *Applied Optics*, Vol. 28, No. 5, March 1, 1989.

3. B. Javidi, "Programmable binary nonlinear optical processor for asociative retrieval", *Optical Engineering*, Vol. 28, No. 5, 1989.

4. B. Javidi, "Nonlinear joint power spectrum based optical correlator", *Applied Optics*, to appear in Vol. 28, 1989.

OPTICAL FORMATION OF INTERCONNECTION WEIGHT MATRIX FOR A NEURAL NET USING ELECTRON TRAPPING (ET) MATERIALS

Suganda Jutamulia, George Storti
Joseph Lindmayer, William Seiderman

Quantex Corporation, 2 Research Court, Rockville, MD 20850

Abstract

Electron trapping (ET) materials are capable of performing multiplication and storing optical information in memory. The stored information can be summed and subtracted optically. The interconnection weight matrix of a neural net can be formed optically and stored in an ET device.

1. Introduction

There are two major options to implement a neural net - namely electronic and optical. The paper will describe the use of ET materials to implement optical neural networks. A neural network for associative memory can be represented by a matrix vector multiplication [1]. The matrix vector multiplication can be easily performed optically utilizing the interconection capability provided by an optical system using two orthogonal cylindrical lenses [2], where the vector and matrix represent a 1D input and the interconnection weight function, respectively. However, the interconnection matrix cannot be formed optically from the given 1D memories. Instead, another means such as a serial electronic computer has to be used to do calculations for forming the matrix. The mathematical formation of an interconnection matrix from given memories is known as prescribed learning.

Many optical architectures for implementing neural network models have been demonstrated. In these architectures, an optical mask which might be an electronically addressed spatial light modulator (SLM) is used to display the matrix formed by other than optical means. The electron trapping (ET) device developed by Quantex as well as other SLMs can be used to perform matrix vector multiplication with a classic arrangement [3], in addition to 2D associative memory [4,5]. Furthermore, the ET device has an inherent capability of summing, subtracting, and storing optical data. This extra capability, which is not commonly found in SLMs, will be utilized for the optical formation of an interconnection matrix or prescribed learning [1].

2. Electron trapping (ET) materials

The ET materials are IIa-VIb compounds with two specific rare earth dopants added. Both ground and excited states of each impurity exist within the band gap of the wide-band-gap host material (Eg > 4 eV). Visible light

(e.g., 488 nm) excites electrons from the ground state to excited states of one of the dopants from whence the electrons transfer over to the second dopant. The electrons remain in the ground state of the second dopant for very long times. Subsequent exposure to IR light (e.g., 1064 nm) excites the trapped electrons to the excited states of the second dopant, the electrons transfer to the excited states of the first dopant and return to the ground state of the first dopant with the emission of orange to red light. At the present time, the ET materials are mostly applied as wavelength upconverters and erasable optical memories. Measurements show that the ET materials possess a large dynamic range covering 4 orders of magnitude.

3. Optical formation of matrix

The ET materials are currently used as an erasable optical memory. Blue light excites electrons in the ET materials to a higher energy state at which they become trapped. A subsequent IR exposure will release the trapped electrons to the ground state with an orange emission. The emission pattern is a replica of the information pattern that has been stored in memory. Thus, the ET materials can be straightforwardly applied to a neural network. The interconnection weight function of the matrix is stored as an optical memory in the form of a density pattern of trapped electrons in a higher energy level as a consequence of the blue light absorption. A subtraction of the weight function is done by releasing trapped electrons from the higher energy level to the ground state with an IR stimulation. The summation is performed by successive exposures to blue light.

4. Matrix vector multiplication

To explain the interaction between an input and entire memories, the neural network model assumes that an associative memory retrieval process is equivalent to a matrix vactor outer-product. Hopfield's model can be expressed as follows [1].

$$V_i \rightarrow 1 \quad \text{if} \quad \sum_{j=1}^{N} T_{ij} V_j > 0$$

$$V_i \rightarrow 0 \quad < 0 \quad , \qquad (1)$$

where V_i is output, V_j is input. The interconnection matrix T_{ij} is defined as

$$T_{ij} = \sum_{m=1}^{M} (2V_i^m - 1)(2V_j^m - 1) \quad , \text{ for } i \neq j$$

$$= 0 \quad , \text{ for } i=j \quad , \qquad (2)$$

where V_i^m and V_j^m are ith and jth elements of the mth memory vector V^m.

The matrix vector multiplication expressed by

Eq.(1) can be optically performed using a fan-out and a fan-in cylindrical lens [2]. The product of each term is provided by the ET materials. The emission intensity from the ET layer is proportional to the product between the intensities of the write-in blue matrix and the read-out IR vector.

5. Prescribed learning

The terms of $(2V_i^m - 1)$ $(2V_j^m - 1)$ in Eq.(2) will be either +1 or -1. Thus, a matrix element is a summation of a series of +1s and -1s. The matrix elements of +1 are written with blue light into the ET device to increase the trapped electrons. Similarly, the elements of -1 are written with IR light into the ET device to decrease the trapped electrons. The contributions of +1 and -1 are determined through the XNOR and XOR between vector elements V_i^m and V_j^m, respectively. A bias level is required to avoid the negative numbers of trapped electrons in the ET device. After the matrix has been formulated, an input vector carried by IR light is incident onto the ET device. Consequently, an emission representing the matrix vector outer-product or a 1D associative memory will be obtained.

6. Experimental results

An experiment has been conducted to evaluate the feasibility for the optical formation of an interconnection matrix. For example, the procedure to obtain a matrix consisting of elements of -1, 0, and +1 is described as follows. The ET device is first exposed to visible light to provide a zero bias level. Then, the elements of +1 and -1 are formed by illumination of visible and IR light with masks shown in Fig.1(a) and (b), respectively. The resulting matrix is analytically shown in Fig.1(c). Figure 2 shows the emission from the ET device with an uniform IR light stimulation indicating the matrix which has been formed optically. Three distinctive gray levels show matrix elements of -1, 0, and +1.

7. Concluding remarks

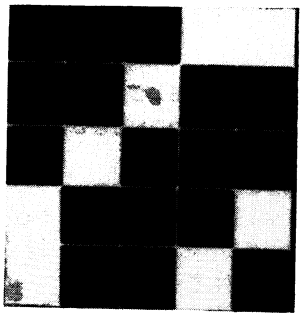
We have described an optical method to implement neural networks using ET materials. The advantage of ET materials over other SLMs is the large dynamic range provided so that an interconnection matrix with multiple levels can be constructed. This will substantially reduce the error of neural nets as compared with binary or ternary clipped interconnection matrices.

Another feature is the capability of optically performing a prescribed learning that will preclude the need to use a computer for re-calculating the whole interconnection matrix when a memory vector is changed. This is certainly useful for a learning scheme based on error propagation as well. We have shown one procedure for the

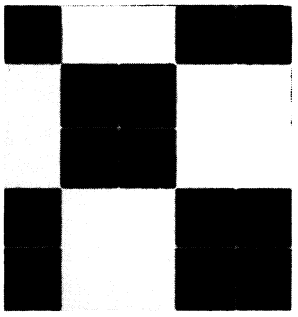
formation of the interconnection matrix. Other procedures are possible for performing XOR and XNOR operations and will be presented.

8. References

1. J.J.Hopfield, "Neural networks and physical systems with emergent collective computational abilities," Proc. Natl. Acad. Sci. U.S.A. 79, 2554 (1982).
2. J.W.Goodman, A.R.Dias, L.M.Woody, "Full parallel, hugh speed incoherent optical method for performing discrete Fourier transform," Opt. Lett. 2, 1 (1978).
3. A.D.McAulay, J.Wang, C-T. Ma, "Optical orthogonal neural network associative memory with luminescent rebroadcasting devices," IJCNN, June 18-22, 1989, IEEE Cat. No. 89CH2765-6.
4. S.Jutamulia, J.Lindmayer, G.Storti, "Optical pattern recognition and associative memory using electron trapping materials," Proc. SPIE, Vol. 1053, 67 (1989).
5. S.Jutamulia, G.Storti, J.Lindmayer, "Optical associative memory with nonzero diagonal interconnection matrix based on electron trapping materials," IJCNN, June 18-22, 1989, IEEE Cat. No. 89CH2765-6.



(a)



(b)

0	-1	-1	1	1
-1	0	1	-1	-1
-1	1	0	-1	-1
1	-1	-1	0	1
1	-1	-1	1	0

(c)

Fig.1. Masks for (a) visible and (b) IR illumination. White squares are transparent areas. (c) Resulted matrix consisting of elements of -1, 0, and +1.

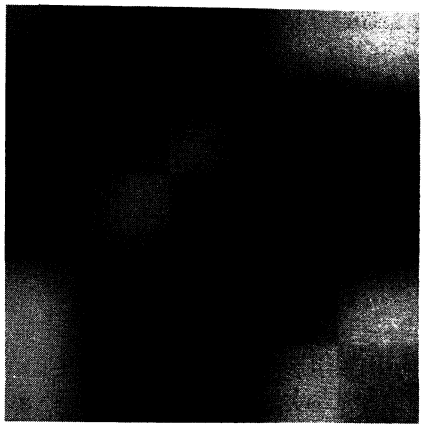


Fig.2. Emission from the ET device verifying the resulting matrix shown in Fig.1(c).

A Stochastic Neuron Model for Pattern Recognition

Danny Kilis and Eugene Ackerman
Health Computer Sciences, University of Minnesota, Box 511 UMHC, Minneapolis, MN 55455

Abstract

A stochastic neuron used in pattern recognition is introduced in this paper. The motivation is to incorporate indirect, dynamic neurophysiological activities in the brain. A Monte Carlo technique is used to cause biased randomness in deciding whether individual neurons should fire, and whether they should die. The stochastic neuron model is applied to a pattern-recognizing neural network, which is implemented using a simulation shell. The resulting network is a more realistic model of neurophysiological activities in the brain.

Introduction

A general, fundamental hypothesis in neural network modeling is that information is transmitted solely by signal impulses through individual neurons. Following this hypothesis, the basic structure of a simulated neuron consists of a cell body (*soma*), a group of fibers (*dendrites*), and an *axon*. A neuron simplified in this fashion receives impulse signals from its dendrites and transmits its output through the axon, which then splits into branches that connect to the dendrites of other neurons via synapses. Information is stored in the form of synaptic strengths. Communications among the neurons are carried by impulse signals that flow via these synapses. A neuron will fire impulses along its axon only when the sum of the excitatory and inhibitory signals that it receives from its dendrites exceeds a certain *threshold*. In this computational model, the products of each input signal and the corresponding strength are combined to produce one or more *activation values* for the neuron. The activation values are then passed through a function, usually non-linear, to produce an *output value*. Subsequently, the decision whether a neuron should fire is made by passing the output value through a threshold function.

The above computational model ignores indirect neurophysiological interactions resulting from slowly varying potentials and hormonal flows. Yet clinical measurements have indicated that these communication media display valuable information concerning the activities in the brain [1]. For example, electroencephalographic potentials (EEG), which are one type of surface potentials associated with the brain, have been used to diagnose, monitor, and control irregularities in the brain. To conform a neural network more closely to neural activities in the brain, there is a need to incorporate, in some fashion, these other forms of communication.

A major problem with any attempt to incorporate the afore-mentioned indirect communication media is the difficulty in quantifying the dynamic and chaotic phenomena observed. A Monte Carlo method used for this purpose is described in this paper. The technique is applied to a version of Fukushima's 1984 model [2] for pattern recognition. Using this technique, biased randomness is imposed on decisions whether to fire individual neurons as well as whether to allow neurons to die. The resulting neuron model is termed *stochastic neuron*. The version of Fukushima's model that is implemented is called COGNET, an acronym for cognitron network.

Other stochastic methods, such as a Boltzmann machine [3], have been used to model neural networks. The purposes of these stochastic methods, however, are to derive a measure of the learning mechanism and to improve this mechanism rather than to quantify activities in the brain. The stochastic neuron model is able to incorporate indirect neurophysiological activities in the brain.

Rationale for Stochastic Neurons

The main motivation for using a stochastic neuron is to incorporate dynamic neurophysiological phenomena into a neural network. The brain consists of about 10^{11} neurons connected in a dense network. A variety of characteristic local changes occur at each neuron. These changes sometimes cause a rapid increase in the electrical potentials of the neurons. In addition, there are slower electrical potentials at the surface of cerebral hemispheres. These potentials, known as the EEG, often occur in characteristic patterns, and are detectable on the surface of the scalp. Some patterns are rhythmic; they change depending on the state of consciousness, awareness, and the activity of the brain [1]. Several kinds of periodic waves, such as alpha and beta, have been identified. Certain circulating hormones and metabolites in the blood also influence the behavior of the neurons in the brain.

From a computational viewpoint, the use of stochastic neurons increases the random characteristic of the neural network model. Thus the output of the system is less predictable than when deterministic neurons are used. A neural network made up of stochastic neurons is a more realistic model of the brain.

Application

The structure and implementation of COGNET are briefly reviewed here. The central components of Fukushima's network include the multi-layered, hierarchical structure, self-organizing property, and bidirectional flow of information. The network consists of four-layered modules. Within each module, the neurons are further divided into subgroups. Each subgroup of neurons is arranged in a two-dimensional array called a *cell plane*. In each layer, except for the first, there are two inhibitory cell planes and nine cascaded excitatory cell planes. For the first layer, also known as the input and output layer, there is only one excitatory cell plane.

A simulation shell, called SUMMERS (acronym for Simulation Utilities and Monte Carlo Models for Event-driven Research Studies), was used for the implementation of COGNET. SUMMERS was first conceived by Seaholm [4] and has subsequently been significantly improved [5,6]. The simulation shell is composed of routines and utilities coded mainly in VAX VMS FORTRAN. The shell facilitates simulation of epidemiological models, especially those of the discrete-time and micropopulation types, using Monte Carlo techniques. The main distinction between a simulation shell and a utility package (e.g., IMSL) is that the shell provides not only re-usable code, but also conceptual commonalities that can be used to develop different models. The concept of a cluster, for instance, is used commonly in stochastic micropopulation studies to represent interacting groups of elements, see for example [7]. This concept can be used in neural network models to describe interconnected groups of neurons. Furthermore, functional programming techniques, similar to those in a LISP environment, have been employed to help achieve explicit and clean coding as well as the ease of adaptation of new models.

Monte Carlo Method and Examples

A Monte Carlo technique is used to emulate the neurophysiological phenomena in the brain. This technique has been used in epidemiological simulation studies [7] to allow for random variation of the input parameters and to compute the average outcomes of repeated simulations. The choice of this method for modeling a neuron stochastically arises from the ease of implementation using observed or fabricated data rather than rigorous mathematical and statistical formulations.

The technique first involves determination of the cumulative probability distribution for a selected parameter. This probability is obtained by initially sampling data, such as EEG potentials, which correspond to one or more characteristic patterns. The magnitude of the sampled data is then normalized to an appropriate range. A cumulative probability distribution graph is drawn based on the number of occurrences of the sampled data that fall within the selected range. (See Figure 1.) Subsequently, a pseudo-random number generator that follows a predetermined distribution, such as, a uniform or normal distribution, is used to pick a probability value. Last, a corresponding value for the selected parameter is derived from the cumulative probability distribution. In this manner, the parameter is estimated stochastically at each time interval of the simulation.

Using the Monte Carlo technique, the decision whether to fire individual neurons can be biased stochastically. For example, let φ and f be the threshold and non-linear activation functions, respectively, of the neuron. A parameter, say θ, is then combined with f to bias the activation value, u. This results in

$$u = \varphi(f - \theta) \tag{1}$$

The cumulative probability distribution for θ can be determined empirically as described in the preceding paragraph. Similarly, the decision whether to allow individual neurons to die is randomly determined. The generic processes in using the Monte Carlo technique are illustrated in Figure 2. In conclusion, the Monte Carlo technique offers a simple means of incorporating indirect neurophysiological activities in the brain into a neural network.

Acknowledgement

This work was supported in part by grant P41-RR01632 from NIH. The assistance of L.C. Gatewood and J.M. Lundgren is gratefully acknowledged.

References

[1] E. Ackerman and L. Gatewood, *Mathematical Models in the Health Sciences*, pp. 206–231. Minneapolis: University of Minnesota Press, 1979.

[2] K. Fukushima, "A hierarchical neural network model for associative memory," *Biological Cybernetics*, vol. 50, pp. 105–113, 1984.

[3] J. L. McClelland, D. E. Rumelhart, and the PDP Research Group, *Parallel Distributed Processing: Exploration in the Microstructure of Cognition, Vol. 1: Foundations.* Cambridge, Massachusetts: the MIT Press, 1987.

[4] S. K. Seaholm, *Adaptable Software for Simulation of Interacting Population Models.* PhD thesis, University of Minnesota, Minneapolis, University Microfilms International, Ann Arbor, Michigan, 1987.

[5] E. Ackerman, "The SUMMERS Model," 1988. Resource for the Simulation of Stochastic Micropopulation Models, Health Computer Sciences, University of Minnesota, Minneapolis, Minnesota.

[6] E. Ackerman, D. Kilis, and G. A. Hatfield, "A micropopulation model adaptation for neural network studies," in *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, (Seattle, Washington), 1989.

[7] E. Ackerman, L. R. Elveback, and J. P. Fox, *Simulation of Infectious Disease Epidemics*. Springfield, IL: C.C. Thomas Publisher, 1984.

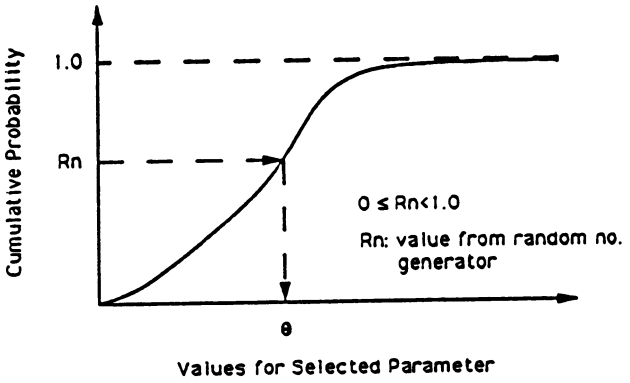


Figure 1: Monte Carlo decision

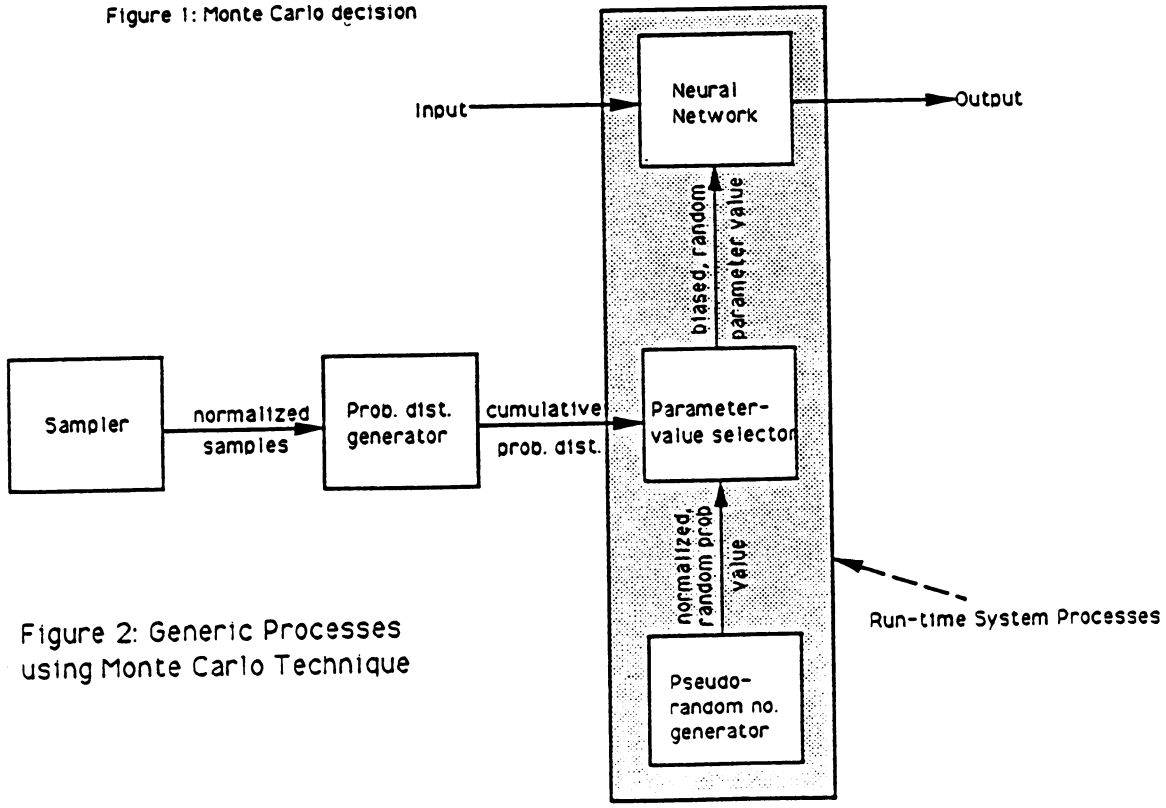


Figure 2: Generic Processes using Monte Carlo Technique

SYSTOLIC IMPLEMENTATION OF MULTI-LAYER FEED-FORWARD NEURAL NETWORK WITH BACK-PROPAGATION LEARNING SCHEME

Hon Keung KWAN and Pang Chung TSANG
Department of Electrical Engineering
University of Windsor
Windsor, Ontario
Canada N9B 3P4

Abstract

In this paper, systolic implementation of a multi-layer feed-forward network together with the back-propagation learning algorithm is presented. In this design, the network and the learning algorithm can be implemented separately or together. The resultant arrays are regular and of nearest neighbour interconnections which are attractive for VLSI implementation.

Background Theory

Consider a L-layer (layer 0 to layer (L-1)) feed-forward neural network consisting of N_k neurons at the kth layer. Define the input of the ith neuron at the kth layer

$$x_i[k] = \begin{cases} i'_i & \text{for } k = 1 \\ o_i[k-1] & \text{for } 2 \le k \le L-1 \end{cases} \quad (1)$$

where i'_i is the ith element of the input vector $\underline{I}'$; and $o_j[k]$ is the output of the jth neuron at the kth layer.

The recalling of such a L-layer feed-forward neural network can be described by (2)-(3). For k=1 to L-1

$$net_j[k] = \sum_{i=1}^{N_{k-1}} x_i[k]\, w_{ij}[k] - t_i \quad (2)$$

and

$$o_j[k] = f(net_j[k]) \quad (3)$$

where $w_{ij}[k]$ is the element at the ith row and jth column of the kth layer connection matrix (from (k-1)th layer to kth layer) $\underline{W}[k]$; t_i represents a threshold value; $net_j[k]$ is the activation of the jth neuron at the kth layer; and $f(\cdot)$ is the activation function (e.g. sigmoid function).

In the learning phase, according to the back-propagation learning rule [1], the weight change from the ith neuron at the (k-1)th layer to the jth neuron at the kth layer following the presentation of input pattern p is defined as

$$\Delta_p w_{ij}[k] = \alpha\, \delta_{pj}[k]\, x_{pi}[k] \quad (4)$$

where α is the learning rate parameter; x_{pi} is the ith element of the input pattern p; $\delta_{pj}[k]$ (which is defined by (5a-b)) is the output error of the jth element in the presentation of the pattern p.

For the output layer, k = L-1

$$\delta_{pj}[L-1] = (o'_{pj} - o_{pj}[L-1])\, f'(net_{pj}[L-1]) \quad (5a)$$

where o'_{pj} is the jth element of the target output; $f'(\cdot)$ is the derivative of the activation function.

For the hidden layers, $1 \leq k \leq L-2$

$$\delta_{pj}[k] = f'(net_{pj}[k]) \sum_{i=1}^{N_{k+1}} \delta_{pi}[k+1] \, w_{ij}[k+1] \qquad (5b)$$

Systolic Implementation of the Multi-layer Feed-forward Network

The resultant systolic implementation is shown in Figs. 1-2. The implementation is composed of (L-1) arrays of basic cells. Each array represents a connection between two adjacent layers in the feed-forward network. The kth array (connection from (k-1)th layer to the kth layer) is composed of N_k by N_{k-1} basic cells. Defining a clock cycle as the maximum time between the time required by one real multiplication and one real addition, and the time for thresholding and quantization. The input is fed into the network from the top of the 1st array. During the 1st clock cycle, the 1st element of the input pattern enters the basic cell at the 1st row and 1st column of the 1st array. The next element will enter the next right cell at the top row of 1st array at next cycle. Similarly, the remaining elements of the input pattern will enter the top row of the 1st array sequentially.

After receiving the input from the top, each basic cell multiplies the input with the weight stored and adds to the accumulated value which flows from left to right. Therefore, in the next clock cycle, the basic cell will pass the previous input and accumulated value, respectively, to the next cell below and in the next right cell. As the accumulated values flow from left to right, '0' should be fed into the left boundary (at the start of the accumulation path) of the array at all time.

After N_0 clock cycles, the activation of the 1st neuron at the 1st layer is known. This activation is required to pass through the thresholding and activation function (elements 'Q' at the right boundary of each arrays in the Fig. 1) in order to obtain the output value, before passing to the next array of the feed-forward network. The output of the remaining neuron at the 1st layer will pass into the next array (i.e., 2nd array) sequentially. The whole process is repeated for the next array, and so on, until the final output is obtained. After $\Sigma(N_k+1)$ (from k=0 to L-2) clock cycles, the first output of the network is obtained at the 1st element of the right boundary at the last array (i.e., the (L-1)th array). The remaining outputs can then be obtained sequentially.

Systolic Implementation of the Back-propagation Learning Rule

Figs. 3-4 show the systolic implementation of the learning rule. In Fig. 3, the kth learning array accepts sequential input of the error of jth neuron at the kth layer, $\delta_{pj}[k]$ at the right boundary of the array and outputs the computed error, $\delta_{pj}[k-1]$, of the jth neuron of the (k-1)th layer at the top of the array sequentially. The operation of the learning basic cell is to carry out multiply and add operation in (5b). As the accumulation goes from bottom to top in the learning array, '0' is needed to feed into the array at the bottom at the start of each of the accumulation path. Before the accumulation output is fed into the next array, the accumulation output is multiplied to the derivative of the activation function. As seen from (1) and (3), the latter can be obtained from the original input, $x_{pi}[k]$. For many activation functions, such as the sigmoid function, the hyperbolic tangent function, sine function etc, the derivative of the activation function is related directly to

the activation function itself. This implies that once $x_{pi}^{[k]}$ is known, $f'(net_{pi}^{[k-1]})$ of (5b) can be computed. we represent this computation by a block denoted by Q' in Fig. 3. As the original input exists and passes out at the bottom of the feed-forward array, a series of delay elements ($\Sigma(2N_i+3)$ from i=k+1 to L-1 excluding the triangular synchronization delays) can be added to the bottom of each of the feed-forward arrays in order to obtain the proper synchronization with the learning array. In doing so, the original input at the kth layer ($x_{pi}^{[k]}$) can be input to the learning array from the bottom sequentially. In this learning phase, this original input ($x_{pi}^{[k]}$) is also used to calculate change in weight according to (4). The change in weight is added to the instant weight immediate before the calculation in (5b). After a basic cell has carried its multiply and add operation, it passes the original input ($x_{pi}^{[k]}$), the accumulation value, and the error of the neuron ($\delta_{pj}^{[k]}$) to its appropriate neighbouring cells. Its neighbouring cells accept all inputs and update their weights within the same clock cycle. This accumulation and update process is repeated for the next clock cycle. After the input signal passes through the array, $x_{pi}^{[k]}$ will appear at the top of the learning array at the same time as the accumulation output. The derivative of the activation function can be computed using $x_{pi}^{[k]}$. The accumulation output is then multiplied by the derivative of the activation function to obtained the error of the jth neurons ($\delta_{pj}^{[k-1]}$) of the (k-1)th layer. All these outputs are fed into the (k-1)th learning array to calculate the errors of all the neurons at the (k-2)th layer and also the changes in weights of the (k-1)th layer. The whole process is repeated for the next array, and so on, until the error propagate back to the input of the network. The maximum limit of a clock cycle is equal to the maximum between time of one real multiplication and one real addition, and the time for computation of the derivative of the activation function. For synchronization, the clock cycle of the learning phase can be made equal to that of the feed-forward phase.

Combinated Array for Both Recalling and Learning

The implementation of the two phases can be combined to a combined array as shown in Figs. 5-6. Switches are added to the resultant array to cope with the differences between the two phases. Besides these switches, an additional switch is added to the update path of the weight. This switch is used to avoid undesirable update of the weight during recalling. In the recalling phase, all switches are at the position as shown. The network accepts the desired input pattern and calculates the output of the network. After all the outputs are obtained, target output pattern is fed into the network to calculate the errors at the output layer ($\delta_{pj}^{[L-1]}$). At the same time, all switches switch to the other positions. The errors at the output layer is fed back into the right boundary of the last array ((L-1)th array) in proper timing. Learning phase is then carried out as described in the previous section. After all the weights are updated, all the switches switch back to the original positions. The recalling phase is carried out again.

Acknowledgment

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada.

Reference

[1] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation," Parallel Distributed Processing, Vol. I Foundations, Cambridge, MA:MIT Press, 1986, pp.318-364.

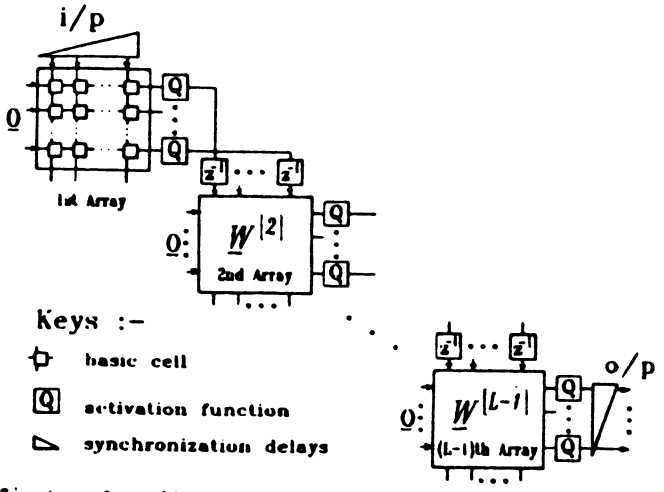


Fig.1 Systolic Implementation of the Recalling Phase.

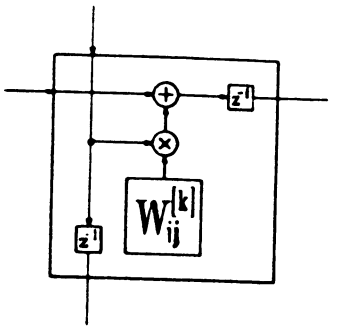


Fig.2 Basic Cell of the Recalling Array.

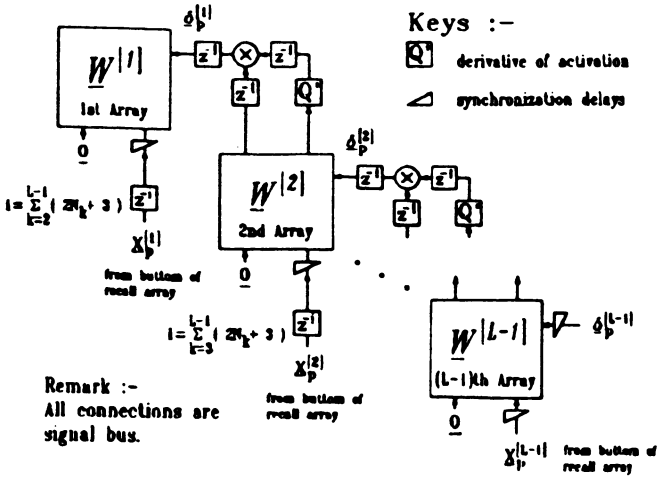


Fig.3 Systolic Implementation of the Learning Phase.

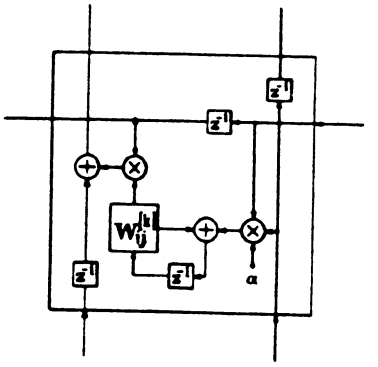


Fig.4 Basic Cell of the Learning Array.

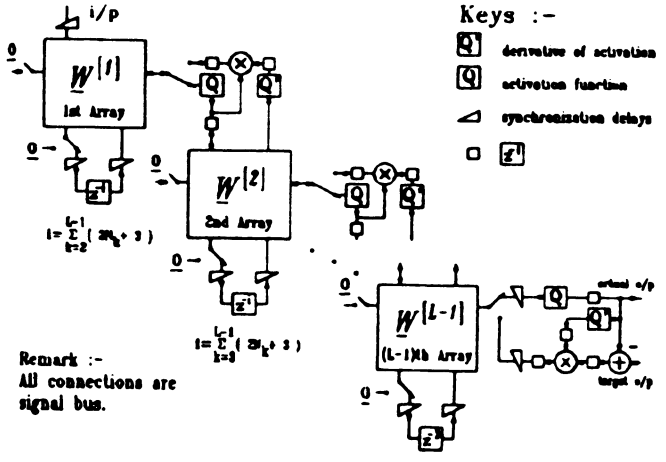


Fig.5 Combined Implementation of the Multi-Layer Feed-Forward Neural Network.

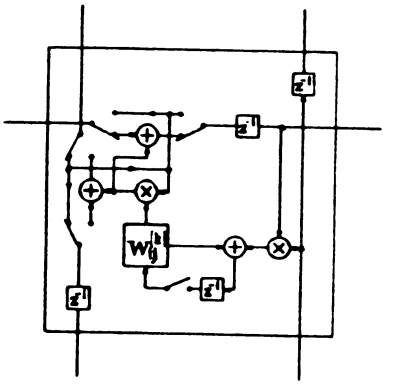


Fig.6 Basic Cell of the Combined Implementation.

GENERALIZED BOOLEAN OPERATIONS FOR NEURAL NETWORKS

Erwin Langheld
Department of Electrical Engineering
University of Stellenbosch
Stellenbosch 7600, South Africa

Karl Goser
Bauelemente der Elektrotechnik
Universitaet Dortmund
D-4600 Dortmund50, FRG

Abstract

In order to simplify neural network design we consider the *processing of analog signals by set operations rather than by arithmetic operations* ; e.g. by replacing the most commonly used sum-of-products formula by a generalized Boolean operation. A well-suited tool for implementing this idea are the socalled t-norms, wellknown from the domain of fuzzy logic. They offer a higher degree of freedom for hardware design than common Boolean operations and by exploiting this viewpoint of higher variability all to-day known fuzzy or stochastic hardware implementations can be treated under these norms. The sum-of-products formula turns out to be a subset of these generalized set operations.

1.Fuzzy Logic and Neural Networks

For direct implementations of neural networks it would be wishful to get large matrices of computational elements onto the chip; therefore designs with high regularity and as simple as possible processor elements have to be preferred. Regarding one matrix column (neuron) of a simple neural network model (fig.1), it is the implementation of the sum-of-products of input signals x_i and weights w_{ij} that has to carry the largest computational burden and therefore is in the focus of VLSI design.

In this article we concentrate on the sum-of-products function s_j, leaving out the sigmoid function $g(s_j)$ which is important for the feedback in connection with an individually chosen learning rule, and we also look rather for *function* than for *accuracy*. Following this path we emphasize the processing of analog signals by set operations rather than by arithmetic operations; and by using triangular norms as mathematical tools that are wellknown in fuzzy set theory [1].

To handle new developing fields (like neural networks) with methods taken from other fields (like fuzzy logic [1-6] or probabilistic logic [7-9]) is an attractive task, leading to new insights for both areas. From the phenomenological point of view the relationship between neural networks and fuzzy logic is based on the fact, that both use *approximative* approaches in their models (learning steps vs. membership functions) and furthermore, that these models are built up rather *implicitely*, e.g. applying experimentally verified learning rules, than by algorithmic formulation of the problem.

2. Sum-of-Products Formula

In our network model one matrix column is representing one neuron. Usually the product of weigth w_{ij} and input signal x_i of the different matrix points of one column j are summed up in some kind of sum-of-products rule $s_j = \sum x_i w_{ij}$ before feeding the sigmoid output stage.

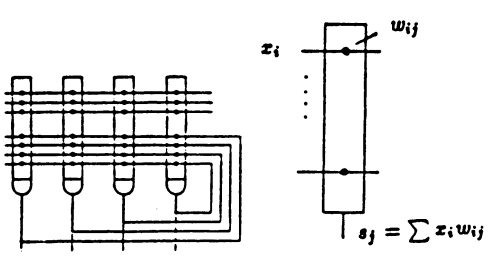


Fig.1 Simple neural network model and matrix column (neuron).

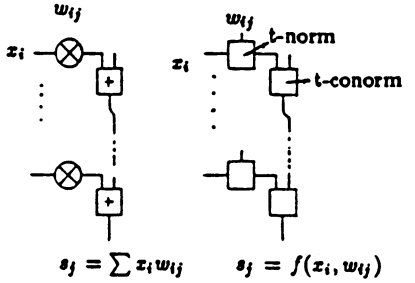


Fig.2 Sum-of-Products and generalized Boolean operation.

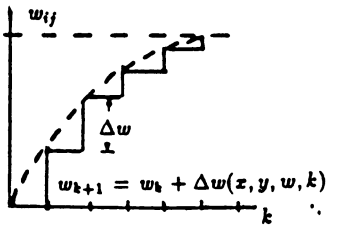


Fig.3 Adaptation of weights w_{ij} as a learning curve.

From the circuit designers point of view this sum-of-products formula deserves some attention. Summation and multiplication are well accepted concise and simple arithmetic operations; but what is regarded as simple by the arithmetician not necessarily has to be regarded as simple by the circuit designer and vice versa. (Since most simulations of neural networks are done on computers this problem usually goes undetected). To bring one example: multiplier and adder circuitry, wether in analog or digital form, seems to be rather costly in comparison to analog maximum and minimum circuitry built up e.g. by diode gates (fig.6).

Therefore we may ask: *Is the sum-of-products formula $s_j = \sum x_i w_{ij}$ which is used in a variety of neural network structures, replaceable by a more generalized form $s_j = f(x_i, w_{ij})$, built up from arbitrary approximation functions? (fig.2) — For example, can we replace arithmetic operations by set operations in order to perform a logical processing of analog signals in neural networks?*

Two observations are pointing into this direction:

- The sum-of-procucts formula is a composite formula characterized by integral and averaging features, thus being less dependant on local deviations. In principle, neurons may be regarded as analog or infinite-valued logical devices.
- The adaptation of the matrix point weights is following a learning curve (fig.3). It seems to be relevant to reach the final level in asymptotic approach; but it seems to be not so much relevant what size of steps is chosen.

These viewpoints are supported by similar approaches in the field of fuzzy logic where for example along with sum-of-products rules also other functions like max-min-operations [4] are used. As an example of a neuronal-network- like structure built up by max-min-functions see [5].

3. Generalized Boolean Operations

In following this approximative approach one has to look for generalizing functions. By applying set operations we assign to the task of a neural network two different characteristics in a very general sense:

- *AND-like (conjunctive) operations characterize cooperative events where different influences are acting together and in parallel* [10]. A typical example for this case is the Hebb-rule, represented by the product $x_i w_{ij}$.
- *OR-like (disjunctive) operations characterize the collection of separate and serial events in order to form a resulting aggregate action, preferrably if the events itself are disjoint* [10]. Examples for this case are the adaptation of the weight parameters and the summation of the different products of a matrix column; the preferrable condition of disjoint events however is not fulfilled by most models of neural networks where the sum-of-products is computed rather simultaneously (exception: stochastic processing [8,9]).

A well-suited tool to match these Boolean-like statements are the triangular norms (t-norms) [7] which are wellknown in the domain of fuzzy logic [1]. These norms range over the interval [0,1] and therefore are valid expressions for probability values or scaled analog values, in contrast to Boolean operations which are confined to the set {0,1}.

The t-norms and the corresponding t-conorms may be regarded as generalized Boolean conjunction (AND) and disjunction (OR). In comparison to Boolean operations (fig.4a) these norms offer a higher degree of freedom by different possible surfaces inside the skeleton (fig.4b). Apparently there are different possible ways how to get from the $z = 0$ level to the $z = 1$ level. This viewpoint is important for hardware design, thus opening a variety of design options and strategies.

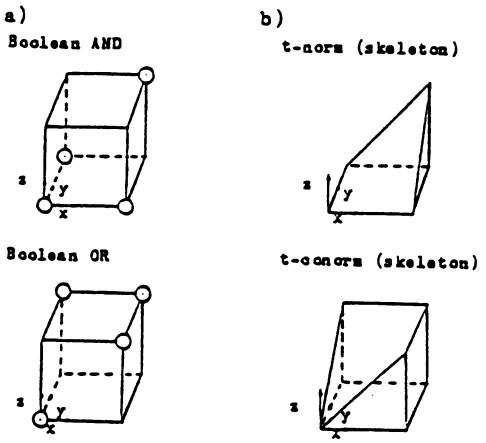


Fig.4 Boolean operations and corresponding t-norms.

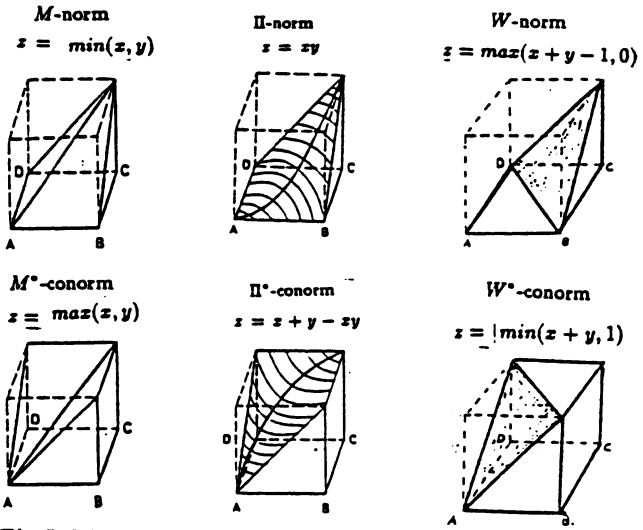


Fig.5 M-, Π-, W-norms and corresponding co-norms.

The t-norms are defined by the following cartesian mapping:
$t : [0,1] \times [0,1] \rightarrow [0,1]$ with

$t(0,x) = 0;\ \ t(1,x) = x$	(boundary conditions)
$t(x,y) \leq t(x',y')$ for $x \leq x',\ y \leq y'$	(monotonicity)
$t(t(x,y)z) = t(x(y,z0)$	(associativity)

The corresponding t^*-conorm is a reflection of the t-norm in the center of the unit cube:
$t^*(x,y) = 1 - t(1-x, 1-y)$, and is conform to the DeMorgan-rule $x \vee y = \overline{\overline{x} \wedge \overline{y}}$, if the fuzzy notation $\overline{x} = 1 - x$ and $\overline{y} = 1 - y$ are applied.

The most important t-norms:		Corresponding t-conorms:	
M-norm	$min(x,y)$	M^*-conorm	$max(x,y)$
Π-norm	xy	Π^*-conorm	$x + y - xy$
W-norm	$max(x + y - 1, 0)$	W^*-conorm	$min(x + y, 1)$

These norms are sketched in fig. 5 and demonstrate the relation: $W < \Pi < M < M^* < \Pi^* < W^*$

A relation between set operations and arithmetic operations already has been stated by Kolmogorov[10] :
$x \wedge y = \phi^{-1}(\phi(x) \cdot \phi(y))$
$x \vee y = \phi^{-1}(\phi(x) + \phi(y))$, for any strict monotone function ϕ .

4. Hardware Implementations of Generalized Boolean Operations

The following examples demonstrate the applicability of t-norms to any fuzzy or stochastic hardware implementation [3,4,6,8,9].

- *M-norm and M^*-conorm:* Simple devices following this norm are diode max- and min-circuits. The error caused by threshold voltages can be eliminated by use of operational amplifiers [3]; though in the case of cascaded max-min or min-max circuits this error is partly compensated (see fig. 9).
- *Π-norm and Π^*-conorm:* In stochastic (probabilistic) computing [8,9] signals are coded into series of statistically independent pulses with the number of pulses per time interval representing the analog value. This statistical puls frequency coding makes it possible to use digital gates as processing elements (fig.7).
- *W-norm and W^*-conorm:* Current mode fuzzy logic circuits developed by T.Yamakawa [6] are based on the bounded difference which can be used as a building block for the W-norm and W^*-conorm (fig.8) as well as for the M-norm and M^*-conorm.

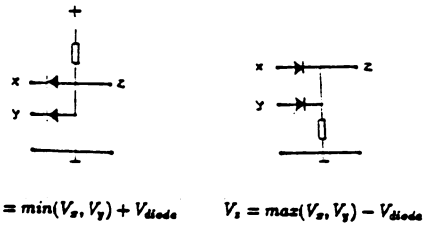


Fig.6 M-norm and M^*-conorm implemented with (ideal) diodes.

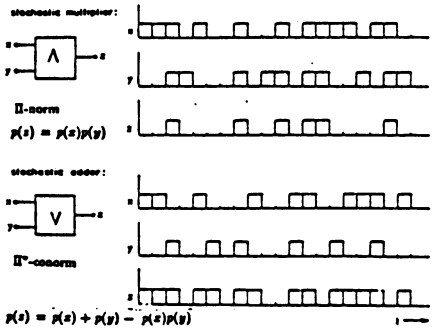


Fig.7 Π-norm and Π^*-conorm: stochastic computing with AND- and OR-gates.

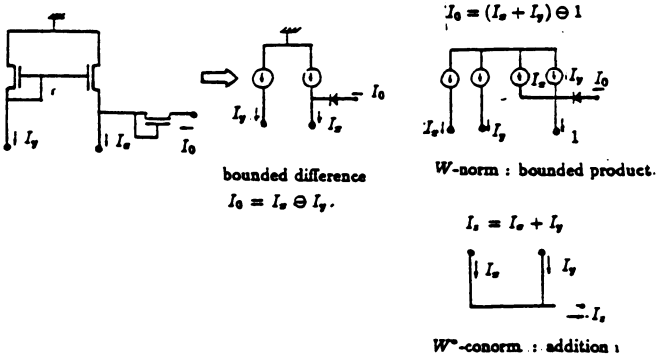


Fig.8 W-norm and W^*-conorm in current mode fuzzy logic [6].

5. Examples of Neuron Implementations

In fig.9 some examples of different possible implementations of a matrix column (neuron) are presented. It can be seen that inside the interval [0,1] the arithmetic sum-of-products implementation b) is a composition of the M-norm and the W^*-conorm. The implementation in Boolean algebra e) may be regarded as a degenerated sum- of-products solution. The Π-norm and conorm, though as circuit design identical with case e) follow a completely different behaviour, because of stochastical coding. Fig.10 shows simulation results.

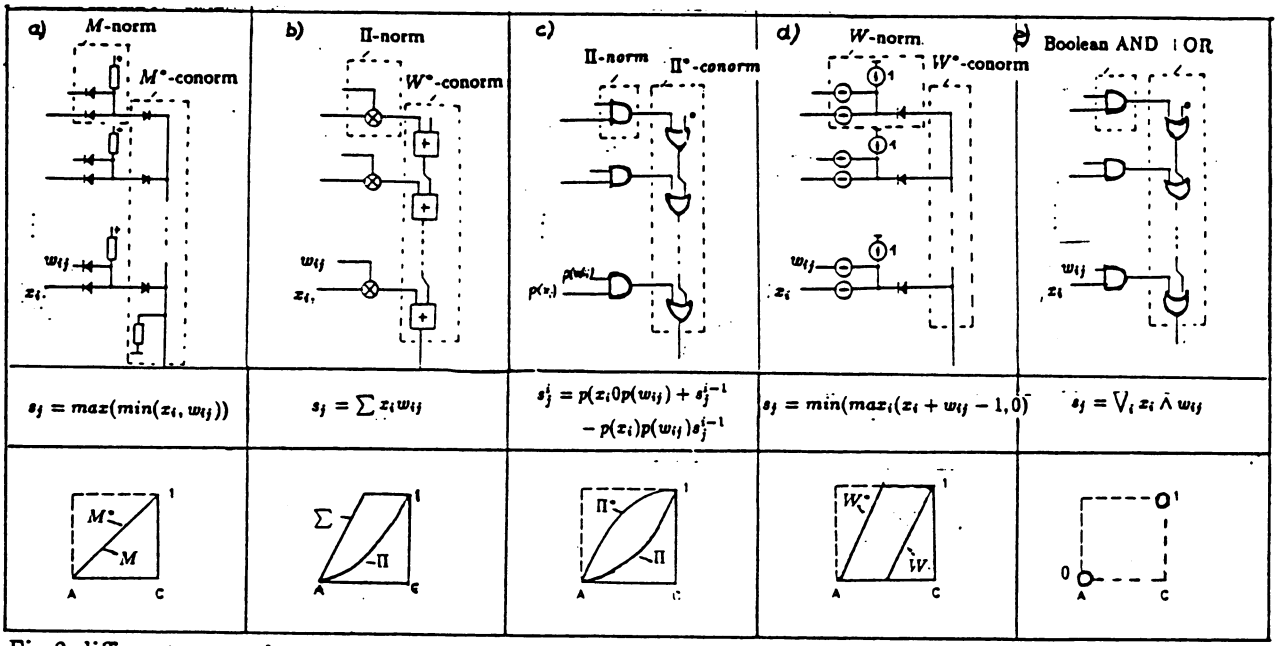


Fig.9 different example principles of the generalized sum-of-products formula of neural networks. For better clearness the diagonal sections A–C are shown too. a) M-norm and conorm, b) arithmetic sum-of-products formula (combination of M-norm and W^*-conorm inside the interval [0,1]),c) Π-norm and conorm (z_i and w_{ij} are given as probabilities (pulses)), d) W-norm and conorm, e) Boolean circuit (degenerated sum-of-products rule, operation not identical with c)!)

6.Outlook

Though implementations within this framework may be done in any analog, digital or stochastic form, the advantage of direct fuzzy hardware design is marked by functional transparency and conciseness which is essential for miniaturization of neural networks and for gaining high circuit density through regularity of design. — In general the processing of analog signals by set operations is offering new prospects in signal processing, demonstrated e.g. by rising attention in the related area of filtering (morphological filters) [11].

References

[1] RR Yager: On ordered weighted averaging aggregation operations in multicriteria decision making. IEEE Trans. Syst. Man, Cybernetics. Vol.18 (1988),pp.183-190.

[2] L Zadeh: Fuzzy logic. Computer 21 (1988),4, p.83-93.

[3] A Kandel; SC Lee: Fuzzy switching and automata. Crane Russel, N.Y.1979

[4] M Togai; H Watanabe: A VLSI implementation of a fuzzy-inference machine: towards an expert system on a chip. Information Sciences 38 (1986),pp.147-163.

[5] M Shimura: An approach to pattern recognition and associative memories using fuzzy logic,pp.449-476 in L Zadeh et. al: Fuzzy sets and their application to cognitive and decision processes. Academic Press,1975.

[6] T Yamakawa; T Miki: The current mode fuzzy logic integrated circuits fabricated by the standard CMOS process. IEEE Trans. on Computers C-35 (1986),pp.161-167.

[7] B Schweizer; A Sklar: Probability metric spaces. North-Holland,1983.

[8] WJ Poppelbaum et. al.: Unary processing. Advances in computers, Vol.26 (1987),pp.47-92.

[9] AF Murray; AV Smith: Asynchronous arithmetic for VLSI neural systems. Electronics Letters,23 (1987),12,pp.642-643.

[10] M Peschel; W Mende: Leben wir in einer Volterra-Welt? Akademie-Verlag, Berlin,1983.

[11] P Maragos; RW Schafer: Morphological Filters. IEEE Trans. ASSP-35 (1987),8,pp.1153-1170.

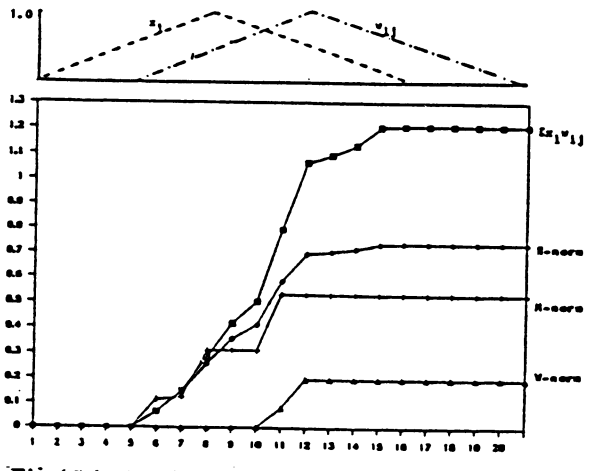


Fig.10 t-norm simulation results; inputs and weights as random distributions.

AN ANALOG SYNAPTIC WEIGHT SYSTEM

W.E. Mattis
Department of Electrical Engineering
Villanova University
Villanova, PA 19085

V. Uzunoglu
Synchtrack
Gaithersburg, MD

ABSTRACT

The synapse of a neural system is analyzed, and a hardware model is presented for so-called "weight" storage. This unique hardware model consists of a bipolar transistor with an S_iO_2 layer applied to the base-emitter junction of the transistor. It is postulated how charge, representing a weight value, is stored beneath the oxide layer. Model equations are formulated, with parameters including insulating (S_iO_2) capacitance, depletion layer capacitance, trapped charge capacitance, resistivity of the oxide layer, etc. From these equations, the storage time for this element is determined and compared to an actual measurement of a fabricated device. This element, and its placement and function in an analog model of a neural system, is then discussed.

PROPOSED MODEL

For neural systems, it is desired to store the changing synaptic weight, and at the same time, have this weight affect the output signal of the modeled postsynaptic membrane. In order to do this, a semiconductor weight storage and active device needs to be designed.

Consider an NPN bipolar transistor (Fig. 1). By changing the bias,, the quiescent point (Q point) can be changed. Changing the Q point will result in a corresponding change in the current gain of the device, given the semiconductor characteristics of the device. Thus, if this property (changing the Q point) can be utilized, it will meet the requirements, namely, that the output signal from the modeled postsynaptic membrane can be influenced by another variable.

It is recalled that for a MOSFET, the conductivity of the channel is affected by the voltage on the gate. In fact, for an N channel MOSFET, the effective area where carriers reside can be increased by changing the gate voltage, increasing the conductivity of the channel.

Given this property, it is proposed that an S_iO_2 layer be added between the base and emitter of a bipolar transistor. Connected to this is a metal contact, a so-called gate. Care must be taken that the insulating layer and contact are appropriately aligned, with regard to the contact, such that the device operates properly and fringing fields are minimized (1). From Fig. 2, it can be seen that for proper operation, the S_iO_2 layer must extend from near the boundary of the emitter to a portion of the base region.

MODEL OPERATION

By applying the proper voltage to this new terminal, a portion of the base region becomes an N region. Thus, the emitter area has enlarged, and therefore, the distance for electrons to travel from the emitter to the collector has been shortened. This implies less recombination of carriers and thus, increased gain. Positive voltage on the gate attracts more electrons beneath the surface of the insulating layer, increasing conductivity (2).

For proper operation of this device, the appropriate connections must be made as shown in Fig. 3. Here a resistor is added to the socalled gate terminal of the transistor to allow for "memory loss". Further, since the neural signals are of low frequency, bias is

obtained by using a modified DC amplifier configuration (1,4). For proper phasing, an additional stage is added. The operation is such that, from a previous circuit, (3) the signals are filtered and accumulated at the presynaptic junction. Then, this signal is routed to the base lead of the memory transistor for amplification, in addition to the new terminal, the so-called gate. Amplification, however, is a direct function of the gate bias, and thus, the charge retained near the insulating S_iO_2 layer. This charge deposition is a direct function of the filtering that takes place at the terminal of the axons or dendrites, which determined whether the synapses were excitatory or inhibitory. Thus, the output of the modeled postsynaptic membrane varys, depending upon whether the activity at the synapse was mostly excitatory or inhibitory.

THEORETICAL FOUNDATION

The storage element functions because of the deposition of S_iO_2 on the semiconductor surface. Charge storage exists because this layer functions as a capacitance. In fact, the insulating layer is composed of a number of layers of various forms of S_iO_2 (5). These include, at the semiconductor surface, a layer of incompletely oxidized silicon, S_iO_x. Then, a strained region of S_iO_2, followed by a strain free layer of S_iO_2. Thus, interface traps and oxide charges will exist. Specifically, interface trapped charges are located at the S_i - S_iO_2 interface, and fixed oxide charges are located near the interface, in addition to mobile ionic charges within the SiO_2. The equivalent capacitance of this configuration is given in Fig. 4. C_i and C_B are the equivalent insulating capacitance, and semiconductor depletion layer capacitance which depends on voltage, respectively. These capacitances exist even in the ideal metal insulator semiconductor model. What is added is the capacitance of the interface traps, C_S, which is a function of surface potential.

Given the S_iO_2 layer, there can, in actuality, be some current flow due to ionic conduction. When the electric field is removed, some ions can flow back to their equilibrium position. Thus, the equivalent circuit for the new "gate" system is shown in Fig. 5. Therefore, the charge on the gate can leak off over time, even without the resistor for so-called "memory loss."

For the equivalent system,

$$C_{MIN} = \frac{C_i\left(C_{BEQ}\right)}{C_i + C_{BEQ}} \tag{1}$$

where $C_{BEQ} = C_B + C_S$

and the minimum value of capacitance is that where there is no further change in the depletion region.

Now the capacitance per square centimeter, in general, is given by (5)

$$C_i = \frac{\epsilon}{d} \tag{2}$$

where ϵ is the permittivity and d is the S_iO_2 layer thickness. C_{BEQ} is a function of the geometry and the width of the depletion layer, as alluded to above.

Given a width of .7 μm, a depth 33.3 μm, with an S_iO_2 layer of 1000 A°

$$C_i = .0101 \text{ pf}$$

$$C_{BEQ} = .0236 \text{ pf}$$

and

$$C_{MIN} = .0071 \text{ pf}$$

For the resistance of the insulating layer,

$$R = \frac{\rho \ell}{A} \tag{3}$$

where ρ is the resistivity, and ℓ and A are the thickness of the insulating layer and the cross-sectional area, respectively.

For the previous geometry with a resistivity of 11×10^{16} ohms-centimeter for the S_iO_2-S_iO_x insulator (5)

$$R = 4.92(10^{18}) \text{ ohms}$$

Now for the circuit of Fig. 5, the charge Q, at any time t, is related to the initial charge on the gate Q_0, and the parameters R and C_{MIN} by

$$Q = Q_0 EXP[-t/RC_{MIN}] \tag{4}$$

Assuming no real degradation of charge after time t', when $Q/Q_0 = .9$, it is found, using the values of R and C_{MIN}

$$t' = 3496 \text{ sec}$$

EXPERIMENTAL VERIFICATION

The analog storage device described was fabricated, and the charge retention noted (2). It was found that the charge was retained for approximately 1 hour, or 3600 seconds, before significant degradation began to take place. This value compares favorably with that derived in the previous section.

CONCLUSION

From the desired characteristics for a neural synapse, especially the excitatory and inhibitory behavior, the design of a transistor system for modification of amplification, as a function of analog weight storage, was postulated. In addition to the requirement for passing all frequencies, D.C and above, and having high input resistance, necessitating the use of a D.C amplifier configuration, the system had to retain charge for a significant period. Thus, the semiconductor physics of this new device was analyzed, a model proposed, and the charge storage time found. This time compared favorably to the storage time for a similar transistor without the DC configuration, originally fabricated for use as a correlator or adaptive equalizer weight storage device (2).

This new element is significant, in that it permits on-line learning to take place in real time, with the option of monitoring weight values, stored as charge on the so-called gate.

REFERENCES

1. Millman, J., MICROELECTRONICS, McGraw-Hill, New York, 1979.

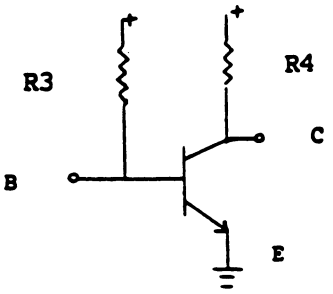
2. Uzunoglu, V., "An Analog Memory Device", INTERNATIONAL TELEMETERING CONFERENCE PROCEEDINGS, Vol 11, Washington, DC, October 1985, pp. 574-578.

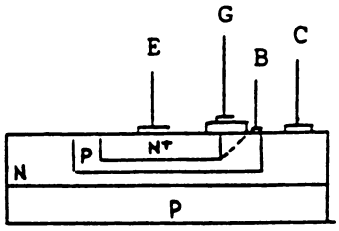
2. Mattis, W., "A Charge-Coupled (CCD) Organ Pipe Architecture For Neural Systems: Modeling And Simulation:, PROCEEDINGS OF THE TWENTIETH PITTSBURGH MODELING AND SIMULATION CONFERENCE, Pittsburgh, PA, May 1989.

4. Gregorian, R. and G. C. Temes, ANALOG MOS INTEGRATED CIRCUITS FOR SIGNAL PROCESSING, John Wiley and Sons, New York, 1986.

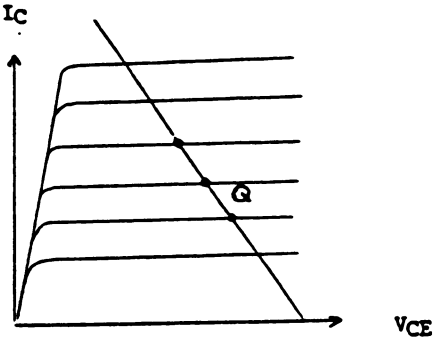
5. Sze, S. M., PHYSICS OF SEMICONDUCTOR DEVICES, John Wiley and Sons, New York, 1981.

6. Shearer, J. L., Murphy, A. T., and H. T. Richardson, INTRODUCTION TO SYSTEM DYNAMICS, Addison-Wesley, Reading, Mass., 1967.

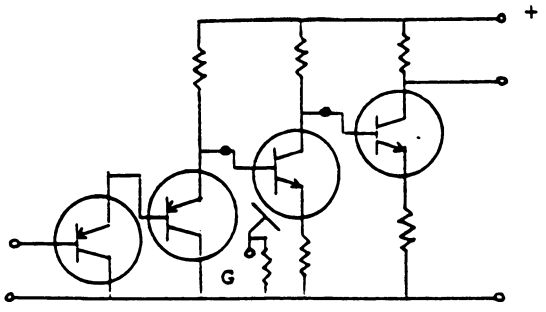




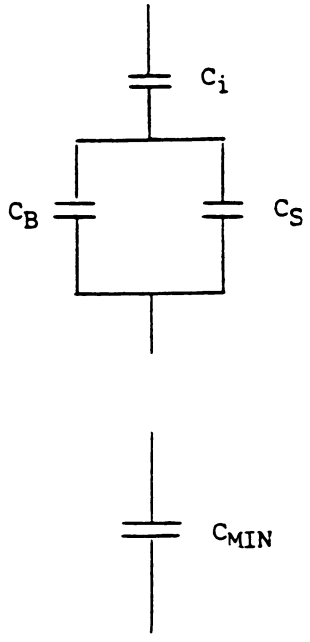
"Figure 2 - Analog Weight Storage Transistor"



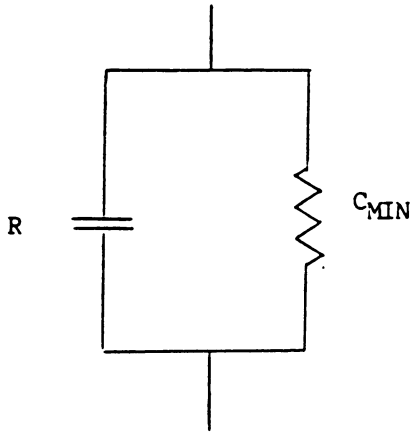
"Figure 1 - NPN Bipolar Transistor"



"Figure 3 - Synaptic Circuit"



'Figure 4 - Capacitance Model Of S_iO_2"



"Figure 5 - Equivalent Circuit For S_iO_2 Layer"

A Hybrid Architecture for the ART2 Neural Model

William R. Michalson and Peter Heldt

Raytheon Company
1001 Boston Post Road
Marlborough, MA 01752

1. Introduction

This paper describes a hardware implementation of the ART 2 neural network model[1]. This implementation addresses the problems associated with massive interconnect, variable synaptic weights, and learning within a hardware architecture capable of supporting many thousands of artificial neurons. The hardware architecture described retains the ability of the underlying ART neural model to use top-down learned expectations to focus attention on bottom-up information in a way that protects previously learned memories from being destroyed by new learning. Thus, at a hardware level, learning may proceed without compromising the synaptic plasticity necessary to admit new learning.

Figure 1 shows a block diagram of ART 1, the predecessor of the ART 2 neural network model[2,3]. Both models contain four basic functional blocks: an F1 feature representation layer, an F2 classification layer, bottom-up and top-down adaptive filters (labled LTM in the figure), and a self-regulating control structure. In operation, data is received at the inputs of the F1 layer where it is noise supressed and contrast enhanced. This data is then presented, via the bottom up filters to the F2 layer where a cooperative-competitive network selects a classification for the input data. This classification is then passed back to the F1 layer via the top-down adaptive filter where it is compared with the input to the F1 layer. If the top-down and bottom-up data are a poor match (meaning that the distance between the top-down and bottom-up vectors exceeds the distance dictated by the vigilance parameter ρ), the control inactivates the current classification of the F2 layer and the process begins again. If the match is good, "resonance" commences.

ART 2 is an extension of the basic ART 1 model, the main difference between these models being in the F1 layer. In ART 2, this layer assumes much of the functionality of the gain control paths shown as part of the attentional subsystem shown in figure 1. By making these gains continuous, based on the degree of mismatch between the F2 top-down feedback and F1 layer input, as opposed to simply being dependent on the presence or absence of signals, the ART 2 model becomes capable of producing responses to arbitrary sequences of analog rather than just binary valued input patterns.

2. A Hardware Architecture for Supporting ART 2

In this section, a hardware architecture which supports the ART 2 neural network model is described. This architecture, shown in figure 2, consists of four functional blocks: a set of Bottom-Up Adaptive Filters (BUAFs), a category representation module (the F2 layer), a set of Top-Down Adaptive Filters (TDAFs), and a feature representation module (the F1 layer). Both digital and analog components are combined to implement various components of the architecture, thus allowing the efficiencies of each type of technology to be exploited within the framework of a single hybrid system.

2.1. The Bottom-Up Adaptive Filter Hardware Module - BUAF

The Bottom-Up Adaptive Filter (BUAF), shown in figure 3, consists of i independent entities which individually multiply the ith element of the feature vector S_i, received from F1, by the ith element in each classifying weight vector Zc_j, where j implies the jth virtual classifying node in F2. The product terms $S_i Zc_{ij}$ are then summed to form the vector inner products, $T_j = S \cdot Zc_j$ $(j = 1..N)$, which are sent to F2.

Once, all the dot products T_j, $(j = 1..N)$, have been computed by the BUAF and received by F2, F2 selects the maximum inner product and sends the corresponding index j_{MAX} to a memory address controller and, in addition, applies the transfer function $G(T_j)$ to the selected maximum vector inner

product $T_{j_{MAX}}$ to form the classifying signal U_j which is sent to both the BUAF and the TDAF. F2 always makes a single $MAX[T_j]$ selection resulting in a single classifying signal $U_{j_{MAX}}$.

In a single ART 2 system, there are a total of M BUAF$_i$s, (i = 1..M). Each of which contains N modifiable-weight synaptic connections to F2. In the hardware architecture, these modifiable-weight connections are implemented as a 1xN vector in digital memory where N is the number of inputs presented by the F1 module. Using digital storage in this manner provides a cost-effective way to store many connection weights over long periods of time.

The outputs of each BUAF$_i$ are connected to a single output conductor which uses current summing to implement the summation needed to form a dot product. Thus, only a single connection between the M BUAF$_i$s and F2 is required. Since every node in the F1 module must effectively connect to every node in the F2 layer, there are a total of O(MxN) inter-layer connections needed. By using current summing to form dot products, the total number of required connections between F1 and F2 is reduced to O(M).

Operation of the BUAF is as follows: During the hypothesis-testing phase, the external memory address controller steps through all N memory locations in each of the M 1xN memory arrays, at each step, simultaneously selecting the same memory location j in each memory array. As shown in figure 4, each BUAF$_i$ uses its currently selected digital synaptic weight zc_{ij}, where j is the same for all the BUAF$_i$s, together with its analog input signal s_i from the corresponding output node in F1, to control a programmable current source, the analog output current of which is proportional to $s_i zc_{ij}$. During the filtering phase, the analog output current from the programmable current source is connected to the output of the BUAF$_i$ through a switch set to "Filter", by the external controller. The analog output currents from all M BUAF$_i$s are collected (summed), at the junction where the output wires from all the BUAF$_i$s are connected to form the resultant vector inner product $T_j = S \cdot Zc_j$ which is passed to F2.

During the bottom-up learning of classifying vectors, each BUAF$_i$ applies its currently selected synaptic weight $zc_{ij_{MAX}}$, converted from digital to analog by the DAC, together with the analog input signal s_i from its corresponding output node in F1, to a variable-gain amplifier, the gain of which is controlled by the classifying signal $U_{j_{MAX}}$ from F2, to derive a new synaptic weight $zc'_{ij_{MAX}}$ in accordance with the specified learning law. During the learning phase, the output of the variable-gain amplifier is connected by the external controller to the ADC which performs the conversion of the new analog synaptic weight z_{ij} to digital form for storage in memory location ij. The contents of the currently selected memory location (ij) are thus updated with the new synaptic weight $zc'_{ij_{MAX}}$.

2.2. The Category Representation Module - F2

The F2 module converts the sum of currents received from the BUAF$_i$s, to a digital form which represents the vector inner product T_j of a feature vector S and a classifying weight vector Zc_j, and compares T_j with T_k ($k \neq j$) in the register. If $T_j > T_{k \neq j}$ then F2 stores T_j in the register and signals the external memory address controller, which controls both the BUAF and the TDAF, to select the next address in the memory arrays which hold the synaptic weight values.

After the memory address controller has completed a scan of N memory locations (which is equivalent to completing the processing of N F2 classifying nodes), it signals F2 to pass it the index j associated with the T_j currently stored in the register. This j becomes j_{MAX}, used by the memory controller to tell the BUAF$_i$s and the TDAF$_i$s which weights to select for modification during their respective learning phases. In the case of the TDAF$_i$s, j_{MAX} also serves to select the top-down expectation weight vector $Ze_{j_{MAX}}$.

2.3. The Top-Down Adaptive Filter Hardware Module - TDAF

Similarly to the Bottom-Up case, there are a total of M TDAF$_i$s, (i = 1..M) which constitute a top-down adaptive filter. These TDAF$_i$ hardware modules are similar in form and function to those which constitute the BUAF. However, unlike the BUAF$_i$s, the analog current outputs from the individual TDAF$_i$s are not wired together, instead, the output of each TDAF$_i$ is connected to its corresponding output node s_i in F1.

As can be seen in figures 2 and 4, during the derivation of a top-down expectation, each $TDAF_i$ uses its currently selected synaptic weight $ze_{ij_{MAX}}$, where j is the same for all the $TDAF_i$s, together with the classifying signal $U_{j_{MAX}}$, to control its programmable current source, the output current of which is proportional to $ze_{ij_{MAX}} U_{j_{MAX}}$. The output from each $TDAF_i$ is connected to its corresponding output node in F1. Collectively, the $TDAF_i$ generate the top-down expectation vector $V_{j_{MAX}} = Ze_{j_{MAX}} \cdot U_{j_{MAX}}$ which is presented to F1.

In figure 2, F1 matches the top-down expectation vector $V_{j_{MAX}}$ with the feature vector S. Depending on the degree of mismatch relative to the chosen vigilance level ρ, F1 signals the BUAF and F2 to remove the current j_{MAX} from their respective pools of valid node indices j.

The learning of top-down expectation weight vectors proceeds in a similar fashion as the learning of bottom-up classifying weight vectors.

2.4. The Feature Representation Module - F1

F1 accepts an analog input pattern vector I and separates signal from noise through subjecting I to a combination of normalization and nonlinear feedback processes to produce a contrast-enhanced corresponding feature vector S at the output nodes of F1.

For ART 2 to match and learn sequences of analog input patterns in a stable fashion, its feature representation field F1 includes several processing levels and gain control systems. The basic concepts for reducing the necessary numbers of interconnections between the F1 and F2 modules can be applied to the intralayer connectivity required in the F1 module. The main difference is that rather than implementing an underlying learning law equation, in the F1 module normalization constants are exchanged and filtering operations are performed. Therefore facilities, such as ROM lookup tables, must be provided to provide the correct transforms on the intralayer normalization constants.

3. Conclusion

The basic framework of a hybrid hardware system which implements the ART 2 neural network model has been described. Based on this framework, ART 2 systems containing many thousands of artificial neurons may be constructed, the main limitation being the resolution of the analog to digital converters. As is the case with the total number of artificial neurons, the primary limitation to processing speed is also the analog to digital conversion since this conversion will tend to be quite slow compared to the time needed to access memory. Even with these limitations, however, it appears to be possible to implement systems containing thousands of neurons which perform classification at rates on the order of tens of milliseconds using current technology.

In summary, the proposed architecture provides the features needed to implement variable-weight connections while maintaining the stability and plasticity needed to support learning. By using current summing techniques, the number of interconnections has been reduced from O(NxM) to O(M) with a minimal impact on the parallelism inherent in the ART 2 model thus making the system a viable alternative for implementing systems containing several thousand neurons.

References

1. G. A. Carpenter and S. Grossberg, "ART 2: Self Organization of Stable Category Recognition Codes for Analog Input Patterns," *Applied Optics*, vol. 26, no. 23, pp. 4919-30, Dec 1, 1987.
2. S. Grossberg, "Adaptive Pattern Classification and Universal Recoding: II. Feedback, Expectation, Olfaction, Illusions," *Biological Cybernetics*, vol. 23, pp. 187-202, 1976.
3. G. A. Carpenter and S. Grossberg, "A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine," *Computer Vision, Graphics, and Image Processing*, no. 37, pp. 54-115, 1987.

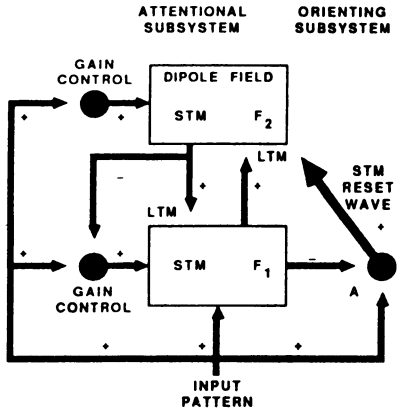


Figure 1

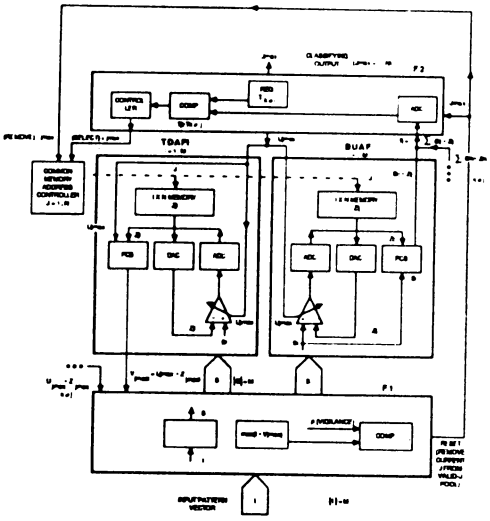


Figure 2

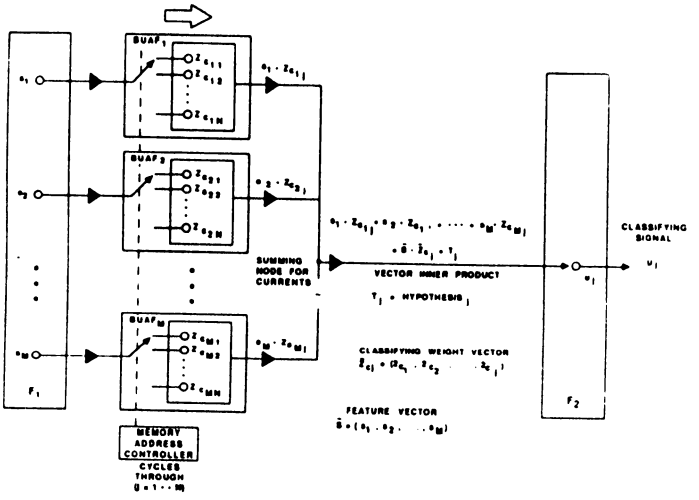
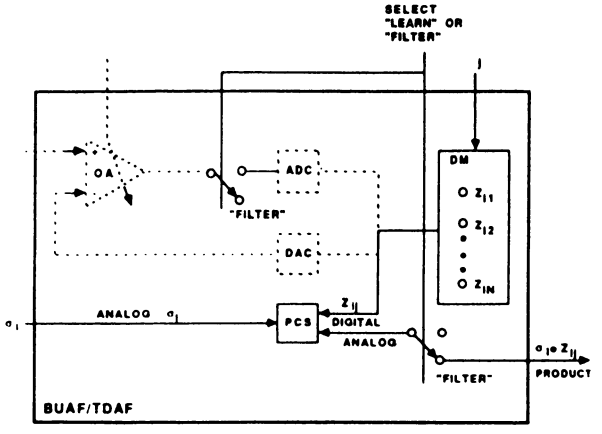


Figure 3



LEGEND

(OA VARIABLE-GAIN OP AMP)
(ADC ANALOG-TO-DIGITAL CONVERTER)
(DAC DIGITAL-TO-ANALOG CONVERTER)
PCS PROGRAMMABLE CURRENT SOURCE
DM DIGITAL MEMORY
σ_i ANALOG INPUT SIGNAL
j ADDRESS OF MEMORY LOCATION FOR Z_{ij}
DASHED LINES INDICATE COMPONENTS NOT RELEVANT TO FILTERING

Figure 4

INTRODUCTION OF NEW ANGLE MODULATED ARCHITECTURES FOR THE REALIZATION OF LARGE SCALE NEURAL NETWORK HARDWARE

By Patrick Nunally and Brian Hallse

General Dynamics Corporation, Pomona Division
P.O. Box 2507 MZ 50-28 Pomona, CA 91769

Abstract:

This paper introduces new principals of 'implicit' hardware structures which operate under the constraints of neural mathematical theory but do not rely on direct correlation between function and structure. This paper discusses specific research, innovations and results in the development of advanced hardware structures (Analog,Digital & Optical) utilizing the new principals introduced.

Introduction:

Developments in neural science have been primarily focused in the development of mathematical theory and its verification/validation in software. The software developed to execute neural structures has been, and continues to be ingenious in its utilization of Von Neuman serial/parallel processors. However, it has become clear that neural processing must begin to directly compete with classical algorithmic methodologies. The lack of neural hardware has been well documented as the major stumbling block in neural technology [1-3] and its importance has been well documented and is not included here for brevity.The design of large scale neural network hardware is a massively complex problem which has currently been addressed with electrical and optical devices which implement mathematical functions directly (i.e. one synaptic weight one dendritic interconnection).

The two most basic requirements of neural hardware structures is a bounded function of thresholding and an ideally unbounded function of weighted (synaptic) interconnect. Currently in analog hardware a voltage or current (voltage/current) threshold, of a sigmoid form, is used as a bounded threshold and an analog potential voltage/current is used as the semi-continuous synaptic weighting mechanism. The advantage of analog hardware is clearly the direct implementation of sigmoidal model theory and straitforward continuous potential weights. However, the disadvantages of 'direct' analog hardware have recently begun to emerge [4]. The structure of weighted interconnection is generally the dominant limitation in the development of large scale analog, digital or optical neural devices. These interconnect structures tend to be area intensive, unexpandable and limit the feasibility of the large neural hardware structures needed in modern systems.

This research addresses new and innovative neural architectures appropriate for future compact, low-power systems. These architectures accommodate the high fan-out/high fan-in properties characteristic of artificial neural network systems with high density interconnects, and have the high throughput capability to achieve rapid processing of large volumes of data.

Theory to Hardware:

Neural hardware structures currently fall into three main categories: analog, digital and optical. Analog structures use voltage/current as the weighted variable between neurons and multiple interconnects for routing potential to the neurons of the next layer in the system. Digital architectures use counting structures as the weight variable between neurons and multiple interconnects busses for routing values to the neurons of the next layer in the system. Optical structures use holographic refraction as the weight variable between neurons and multiple light interconnects for routing values to the neurons of the next layer in the system. This new frequency based neural architecture is independent of these technologies yet generally applies to them all. The architecture uses offsets of frequency as the weighted variable between neurons and a single collective interconnect for routing neuron contributions to the neurons of the next layer in the system.

This technology is a combined analog/digital device architecture, using silicon or other materials, and is based on developments currently in work as part of this research. The architecture is designed to be robust to manufacturing and environmental variability. The goal of this new architecture is to optimize usage of interconnect structures which account for a majority of device area consumption and processing "bottlenecking". The architecture is

designed to be flexible and modular to accommodate evolving neural network system architectures and allows for scale-up to large-sized systems through assembly/interconnection of smaller subsystems.

Neuron Structure:

The structure of the neuron processing unit is design such that the summation of the synaptic weights over time is performed, instead of the summation of voltage/current typically used in analog neuron structures. The neuron is driven by two signals, an excite and inhibit. These signals are driven by the synaptic weights were the synaptic weights are represented as frequencies, with a smaller weight being represented by a larger frequency offset from the carrier and a larger weight represented by a smaller frequency offset. The purpose of the excite and inhibit signals is to increase or decrease the internal sum of the neuron. The internal sum of neurons are thresholded using bandpass thresholding method. The correspondence between the theoretical synaptic summation and the summation performed within the neuron is:

Theory

$$S_{ij} = \sum_{j=1}^{N} W_{ij} X_j$$

Hardware

$$S_{ij} = \frac{1}{C}\int_{0}^{t} (I_e + I_i)\,d(t) + V(0)$$

$$I_e = \sin[\tan^{-1}(E(t))]\ \frac{\mu_p \varepsilon}{t_{ox}}\left(\frac{W}{L}\right)\left[(V_{gs}-V_{tp})V_{ds} - \frac{V^2_{ds}}{2}\right]$$

$$I_i = \sin[\tan^{-1}(I(t))]\ \frac{\mu_n \varepsilon}{t_{ox}}\left(\frac{W}{L}\right)\left[(V_{gs}-V_{tn})V_{ds} - \frac{V^2_{ds}}{2}\right]$$

Where,

$$E(t) = \delta\left[\sum_{j=1}^{N} \Omega_{Eij}(t)\right] \begin{cases} \Omega_{Eij}(t) = \cos\left(W_C(t) + \frac{\Omega X_j}{W_{ij}}(t)\right) & W_{ij} X_j < 0 \\ \Omega_{Eij}(t) = \cos\left(\frac{\pi}{2}\right) & W_{ij} X_j \geq 0 \end{cases}$$

$$I(t) = \delta\left[\sum_{j=1}^{N} \Omega_{Iij}(t)\right] \begin{cases} \Omega_{Iij}(t) = \cos\left(W_C(t) + \frac{\Omega X_j}{W_{ij}}(t)\right) & W_{ij} X_j > 0 \\ \Omega_{Iij}(t) = \cos\left(\frac{\pi}{2}\right) & W_{ij} X_j \leq 0 \end{cases}$$

Synaptic Structure:

The use of analog voltage/current for weighted interconnect has been explored as a primary focus of much of today's current research. This storage of variable weights in analog formats has proved to be a superior structure for data storage. The development of initial functional devices has been completed and the development of expandable concepts must now be initiated. The critical aspect of system structure expandability is the development of weighted interconnect structures which will allow for a maximization of area utilization and flexibility.

By interconnecting multiple synapses on single interconnects they would typically interfere with each other because all signals would interact directly. By utilizing frequency directly as a synaptic weight they would still interfere with each because the spectra of all the signals occupy more or less the same bandwidth. By utilizing synaptic weight to modulate different carrier frequencies, it is possible to translate each signal to a different frequency range. This principle is fundamental to the high density interconnect and expandability requirements of all analog, digital and optical neural systems. Figure 1 shows the power of interconnect scheme versus current methods.

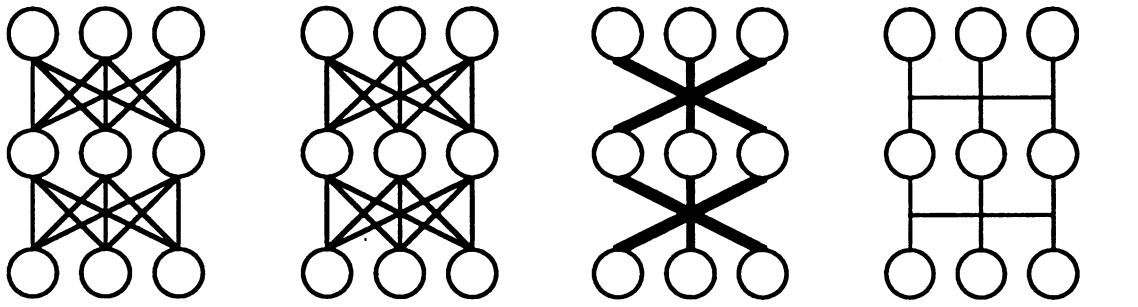


Figure 1. - Comparison of conventional neural structure's connectivity to frequency based neural networks

Frequency is a continuous function which can equate to a large continuous variation of weighting. This weighting function can be achieved by varying the excite carrier frequency in proportion to a modulating signal. The frequency of a particular exponentially modulated frequency can be shifted by this variation. A frequency shift of w_0 in the frequency domain is equivalent to multiplication by e^{jw0t} in the time domain. The significance of this variation is that the demodulation of multiple frequencies on a single line is performed with bandpass filters. These filters can be wall filters which transition from a signal of full amplitude to no transmission emulating a step function. This is of course an ideal condition; real filters tend to role off at some given rate. Using a typical filter represented by the following transfer function:

$$|H(\omega)| = \frac{1}{\sqrt{1+\left(\frac{\omega}{2\pi B}\right)^{2n}}} \qquad \text{[Equation 1]}$$

we can represent a number of different transfer functions through controlling the order of the filter (n) as shown in Figure 2.

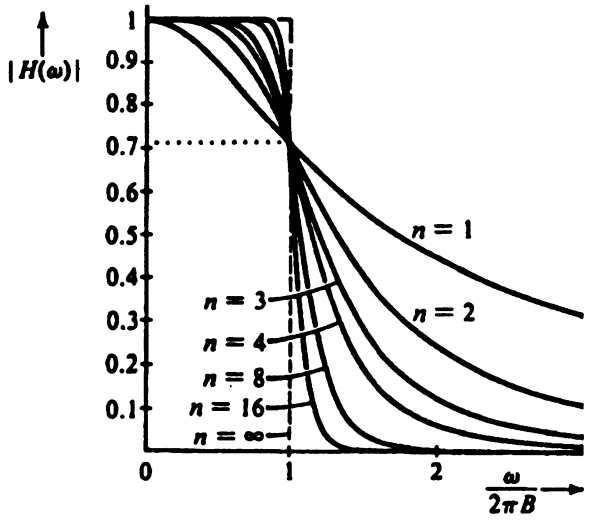


Figure 2 - Typical filter characteristics

By using this principle we can vary a particular neurons contribution to the next layer in the network by simple varying its modulation slightly. The role of of the switch capacitor filters constitute the threshold response of the

previous layer. These filters can emulate common activation functions typically utilized in neural network paradigms [5] shown in Figure 3.

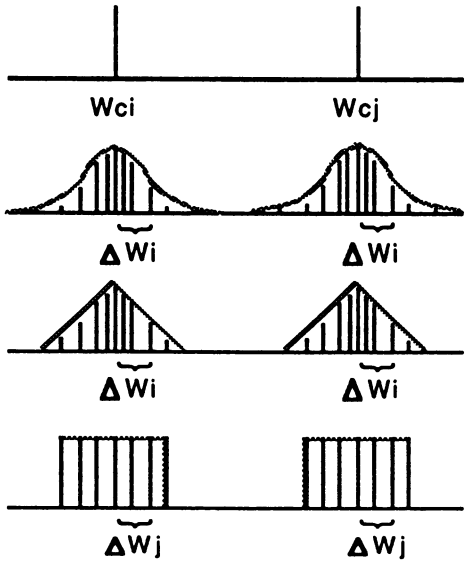


Figure 3 - Neuron contribution to the next layer as a function of filter structure.

The modulation variation is a function if the switch capacitor filter network and the constant amplitude of angle modulation make this system less susceptible to nonlinearities.

Results And Future Work:

This research has resulted in the development of technologies and integrated circuit structures for frequency based neural networks. The primary focus of this research has been geared towards the design and fabrication of a 1.6 micron CMOS silicon analog/digital 40 neuron integrated neural network. This is not meant (nor should it be perceived as) a limitation of this architecture's application to other implementation technologies or larger neural networks (it is merely an economical test vehicle). Higher end limits of device bandwidth have not been reached however, ground work has been established for the expansion of future bandwidth capability. The positive results of this research has led to the initiation of a 256 neuron dual layer fully interconnected analog/digital self configuring neural network project scheduled for fabrication late in 1990.

References:

[1] DARPA Neural Network Study Lincoln Labratory, MIT, 1988

[2] Mead, C., *Analog VLSI and Neural Systems*. Menlo Park CA. Addison-Wesley, 1989. pp. 6-8

[3] Widrow, B. & M. E. Hoff,"Adaptive Switching Circuits,"1960 WESCON Convention,Record Part IV,pp96-104.

[4] VanderBout, D. E. & T. K. Miller, "TInMANN: The Integer Markovian Artificial Neural Network,: in Proceedings of the International Joint Conference on Neural Networks, Vol II, pp 205-211, Washington DC, 1989

[5] Rumelhart, D. E., G. E. Hinton & L. J. McClelland, "A General Framework for Parallel Distributed Processing,: in *Parallel Distributed Processing-Explorations in the Microstructures of Cognition, Vol. 1: Foundations*, D. E. Rumelhart, J. L. McClelland, the PDP Research Group (eds), 45-76, MIT Press (1986).

A Multiple-Bus Network for Implementing Very-Large Neural Networks with Back-Propagation Learning

D. K. Panda and K. Hwang
Dept of EE-Systems, Univ of Southern California
Los Angeles, CA 90089-0781

Abstract: A multiple-bus network incorporating only two types of building blocks is proposed. The architecture, with its built-in features, provides an integrated solution to the ever-demanding high-speed, modular, reconfigurable and VLSI implementable hardware platform to implement back-propagation neural networks(BPNNs). The programmability features provided by software simulators are embedded into the architecture and hence, make it suitable for high-speed real-time implementation of very-large BPNNs.

1. Parallelism in Back-Propagation Neural Networks

Multilayer neural networks using back propagation learning [2,3] are extensively used by neural network researchers. The back-propagation learning algorithm(BPLA) dictates data dependencies in computations (computing outputs of neurons in forward pass, error vectors and weight updates in backward pass) between different layers of neurons. Hence, irrespective of serial or parallel implementation, the parallelism is restricted only to computations involving adjacent layers of neurons.

First, we analyze the computational overhead associated with scaling up a BPNN. Figure 1 (a) shows a three layer BPNN with m, n, and p neurons in input, hidden, and output layers. Assuming full connectivity, the number of computations involved in adjacent layers are $O(mn)$ and $O(np)$ respectively. By scaling up the problem x times (figure 1(b)), the number of computations increases to $O(mnx^2)$ and $O(npx^2)$. This $O(x^2)$ overhead slows down the serial implementation time by $O(x^2)$ and makes the software simulation scheme unrealistic for real-time large scale BPNNs.

The best parallel implementation is to have a hardware featuring dedicated point-to-point(PTP) interconnections, where the computations between adjacent layers can be carried out in $O(1)$ time step. But, the present-day technologies (digital/analog/optics/analog-hybrid) are not matured to implement full PTP connectivity between large number of nodes. Hence, a technologically feasible scheme, demonstrating maximum parallelism with limited PTP connectivity, is ideal for efficient implementation of large scale BPNNs.

The back-propagation learning algorithm uses the following computational steps:

$$u_i(k,l+1) = \sum_j w_{ij}(k) \cdot a_j(k,l) + \theta(l+1) \quad (1)$$

$$a_i(k,l+1) = f(u_i(k,l+1)) \quad (2)$$

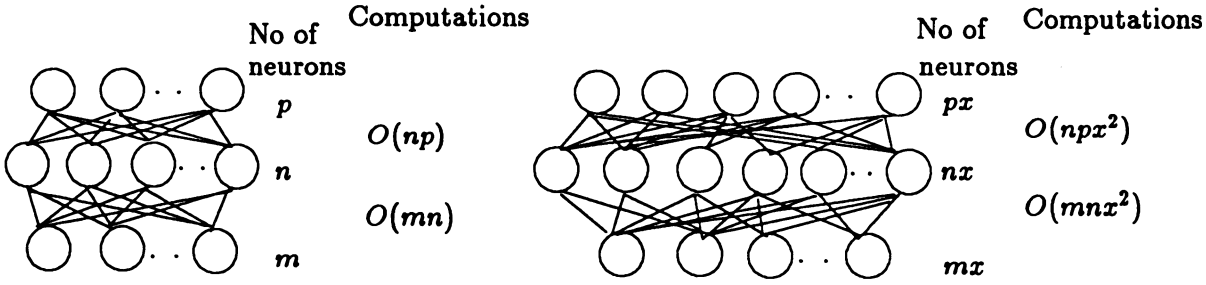
$$w_{ij}(k+1) = w_{ij}(k) + \delta_i(k,l+1) \cdot f'(u_i(k,l+1)) \cdot \eta \cdot a_j(k,l) \quad (3)$$

$$\delta_j(k,l) = \sum_j \delta_i(k,l+1) \cdot w_{ji}(k) \quad if \ \ l \neq L \quad (4)$$

$$= O_j(k,l) - a_j(k,l) \quad if \ \ l = L$$

where k = iteration number, l = lth layer ($1 \leq l \leq L$), L = total number of layers, $1 \leq i \leq N_l$ and N_l = number of neurons in the lth layer.

Steps (1) and (4) are of matrix-vector multiplication type and hence, can be implemented in parallel by assigning a processing element (PE) for each neuron. Efficient distribution of synaptic weights among the PEs is required to minimize interprocessor communications. Step (3) is of array-updation type and hence, can be implemented in parallel. Step (2) can also be implemented in parallel by using a table-broadcast approach instead of a table-lookup approach as mentioned



(a) A back-propagation network

(b) x times scaled back-propagation network

Figure 1: Computational overhead in scaling up a back-propagation network.

in [4]. So, overall, the computational steps in a BPLA exhibit inherent parallelism. In the next section, we propose a multiple-bus network to exploit this parallelism.

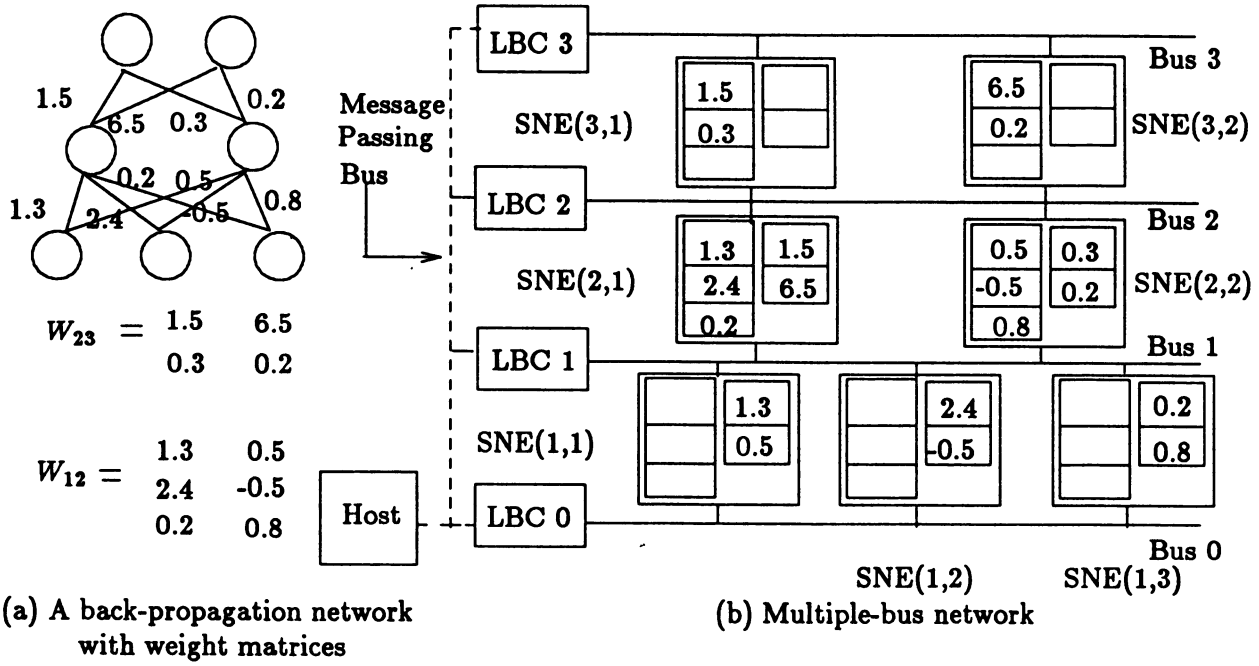
2. The Multiple-Bus Network Architecture

Steps (1) and (4) of BPLA belong to different passes(forward and backward) and the matrix-vector operations are carried out on weight matrices W_{ij}(weight matrix between layers i and j) and W_{ij}^T respectively. To exploit full parallelism, we designate one PE (Synaptic Neuron Element or SNE) per neuron. The synaptic weights are duplicated among these SNEs to provide full parallelism in both passes. SNE (j,k) (kth SNE of the jth layer) stores the kth column of the weight matrix $W_{j-1,j}$ in a ***column memory*** and kth row of the weight matrix $W_{j,j+1}$ in a ***row memory***. Figure 2 shows the mapping. The group of SNEs in adjacent layers are connected to a common bus controlled by a Layer Bus Controller (LBC). The SNEs support dual-ported control and addressing. A (m) layer BPNN is mapped to a (m+1) layered bus structure. LBC(j) is capable of addressing the row(column) memories of SNEs in jth ((j+1)th)) layer. The SNEs in layer j operate in SIMD mode under the control of either LBC(j) or LBC(j-1). All LBCs together with the host share a common message passing communication bus and implement the computational steps of the BPLA by message exchanges.

Initially, the host distributes the synaptic weights of SNEs to the respective LBCs using high-speed message passing. All LBCs work in parallel to store these weights in respective SNEs. During the forward pass, LBC(0) commands the layer 1 SNEs (from left to right sequentially) to broadcast their output values on Bus(1) in consecutive clocks. During this broadcast, all SNEs on layer 2 listen to the broadcasted value and perform a multiply-accumulate(MAC) operation each using the desired weight from the column memory. The weights in column memory are serially addressed in consecutive clocks. After all SNEs of layer 1 have broadcasted their values, the SNEs of layer 2 are ready with the step (1) operation. A table-broadcast approach is used to carry out step (2) in parallel. The non-linear activation function f, stored inside LBC (1) by the host, is broadcasted on Bus 1. All SNEs of layer 2 compare the broadcasted value with the step (1) result in parallel and determine the output of the neurons in $O(1)$ time step.

Steps (1) and (2) are repeated for subsequent layers. At the end of the forward pass, each SNE of the output layer computes the error and broadcasts it to lower layer SNEs. During the backward pass, the SNEs in destination layer use the weights stored in their row memories for computing the back-propagation error and updating the weights. While the backward pass operation between layer j and layer (j-1) SNEs is carried out, LBC(j) copies the weights from row memories of layer j SNEs to column memories of layer (j+1) SNEs. This ensures that the modified weights are used during the forward pass of the next iteration.

The present architecture exploits full parallelism of a BPNN in each of the 4 computational steps. It takes $O(m+n)$ and $O(n+p)$ time steps respectively for forward and backward pass for the



(a) A back-propagation network with weight matrices

(b) Multiple-bus network

Figure 2: Mapping of a back-propagation network onto a multiple-bus network (LBC=Layer Bus Controller and SNE=Synaptic Neuron Element).

BPNN shown in fig 1(a). Since a shared bus is used, scaling the problem size by x times has only linear effect ($O(x)$) on the computing time. Since the bus technology allows only a finite number of nodes to be connected to it, we provide a multilayer bus structure with repeaters (Fig 3) for implementing very large scale BPNNs. The LBCs on a horizontal bus perform themselves the bus repeater operation by receiving a broadcasted value from left(right) bus and re-broadcasting it on right(left) bus with 1 clock delay.

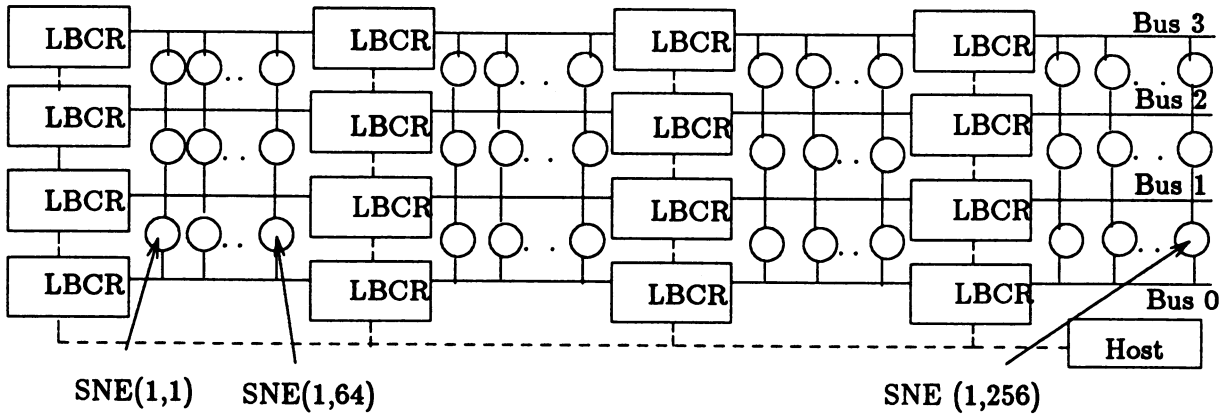


Figure 3: Multilayer-bus network with repeaters for a 256 neurons 3 layer back-propagation network(LBCR=Layer Bus Controller with Repeaters, SNE=Synaptic Neuron Element).

3. Programmability, Modularity and Expandability

The multiple-bus network, providing flexibility at the hardware and firmware level, incorporates the following programmability features:

1. Reconfigurable Interconnections: Any arbitrary interconnection between adjacent layers of neurons is supported. A tag bit of 1(0), associated with each weight, indicates the presence(absence) of an interconnection. The MAC hardware of SNE uses this tag bit to bypass

a MAC operation in the absence of an interconnection. This tagged memory feature allows a smaller size problem to be mapped to the architecture. By disabling selected repeaters, multiple smaller size problems or different instances of a smaller size problem can also be implemented on a common hardware in parallel. This *multitasking* feature results in full resource utilization and fast turnaround time for algorithm development.

2. Neuron parameters: The table-broadcast approach allows dynamic allocation of non-linear function to a group of neurons. The neurons of a single layer can also have mixed parameters.
3. BPNN parameters: Different parameters of a BPNN like number of hidden layers, number of neurons in a hidden layer, gain constants, etc are easily programmable on the architecture by downloading appropriate control program to the LBCs by the host.
4. Downloading/Uploading synaptic weights: Precomputed weights can be downloaded to the memory to run the BPNN with/without learning. The whole computation in the system can also be frozen after a predetermined passes to debug/alter the synaptic weights. This *internal visibility* feature [1] makes BPNN algorithm development much easier.

Besides programmability, the architecture uses only two types of building block VLSI chips: SNE and LBC. Due to the bus oriented structure and message passing communication between LBCs, the architecture is expandable and provides modular growth in both dimensions: 1) increasing the number of neuron layers and 2) increasing the number of neurons in each layer. In the present design, the size of the column and row memory restricts the growth of the system in the second dimension beyond certain stage. This restriction can be alleviated by providing a SNE source address field against each synaptic weight [5].

4. VLSI Implementation and Technology Requirements

The multiple-bus architecture uses the technologically proven bus-oriented interconnections. Fully digital implementation using only two types of building block modules make the architecture suitable for VLSI implementation. The local bus controller with repeater (LBCR) is functionally equivalent to the currently available message passing coprocessor chips (MPC chip for MULTIBUS-II for example). A small portion of the synaptic memory element (SNE) is dedicated to simple processor logic. A major portion of the SNE is dedicated to memory cells for storing the synaptic weights. From the integration and functionality point of view, LBCR stands in par with commercially available medium-level microcontroller and SNE with high density memory chips.

The sophisticated processor technology and high density memory technology of the present-day VLSI world promises high density SNEs and high performance LBCRs. Similar to the available memory chips, SNEs can be fabricated with different word widths and storage capacities to implement different hardware implementation for different sizes of back-propagation network. The architecture, using the VLSI technology, allows implementation of very large BPNNs consisting of 10^5 neurons/layer. Due to the regular bus-oriented interconnections, wafer scale integration of this architecture is also technologically feasible [5].

References

[1] T. Kohonen. State of the art in neural computing. In *Proc ICNN*, pages 79–90, Jun 1987.

[2] R. P. Lippmann. An Intro to computing with neural nets. *IEEE ASSP*, 3(4):4–22, 1987.

[3] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, chapter 8, pages 318–362. Volume 1, MIT Press, 1986.

[4] T. Watanabe, Y. Sugiyama, T. Kondo, and Y. Kitamura. Neural network simulation on a massively parallel cellular array processor. In *Proc IJCNN*, pages 155–161, Jun 1989.

[5] M. Yasunaga, N. Masuda, M. Asai, M. Yamada, A. Masaki, and Y. Hirai. A WSI neural network utilizing completely digital circuits. In *Proc IJCNN*, pages 213–217, Jun 1989.

Parallelized Back-Propagation Training and Its Effectiveness

Ken L. Parker
Allison L. Thornbrugh
Martin Marietta Space Systems, PO Box 179, Denver, CO 80201

Introduction

Training back-propagation neural networks for real-world, real-time problems involves managing large quantities of numeric examples and piping those data through a computationally intensive learning algorithm. In response to this need, a parallel back-propagation (BP) training tool has been developed. The approach used for parallelization is coarse-grained. Training patterns are split among the various processors, gradient estimations are computed in parallel, and then a central copy of the weights is updated. This approach results in a nearly linear relationship between computation speed-up and the number of processors (Chart 1). However, the reduction in training time (defined as time to reduce total sum of squared error to a specified level) is not linear. This is due to the fact that this parallel approach effectively computes a more accurate gradient estimation. It does not necessarily allow more weight updates to occur. Therefore a limit exists to the amount of actual training time speed-up achievable using this method of parallelization. The exact amount of parallel processing power that is useful depends heavily on the nature of the training data and the problem being solved. Two different back-propagation networks have been considered: a simple sum of sine functions and a target detection problem with actual imagery. Based upon evidence gathered from these examples to-date, a relationship exists between the epoch or batch size specified when running the parallel BP training tool and training performance.

Parallelization is only one option for improving training performance. Other options include new or modified learning algorithms [Park87, Simo89, Denn87, Sama87, Shep88], learning rate changes over time [Cate87, Hush88], momentum adjustments [Gogg89], and parallelization of the network model itself [Wata89]. Because the current range of application sizes (1,000 - 5,000 interconnections) and the inherent difficulty in parallelization of the network itself (due to 3-D interconnections on 2-D real estate) — a parallel implementation of the training phase only and not the entire simulator was selected as a starting point. An eighteen-processor Encore Multimax was used for this study. This machine is a bus-based parallel processor with 32 MB of shared main memory. Each processor has a floating point co-processor and 64 KB cache.

Tool Development and Test Results

There are several possibilities for segmenting the BP training problem. One of the simplest approaches is to divide the training patterns into 'n' sections, where 'n' is the number of processors. Each processor contains a complete copy of the neural network simulation and trains using its subset of the training patterns. The back-propagation algorithm that executes at each processor was adapted from work at the University of California San Diego [Rume86]. The issue of weight coherency quickly becomes a problem with this approach. In order for the individual processors to collectively work on a solution, they need to operate from the same weight matrix. Otherwise, each processor would work toward a solution for its peculiar subset of training patterns, not the global solution. This coherency requirement causes the training to become "epoch" training, i.e. - several training patterns are presented to the network before weight changes are made. Figure 1 shows an overview of the tool's operations.

Epoch training produces different effects on different training pattern sets. Each pattern presentation during BP training is used to compute an estimation of the steepest descent gradient on the error surface. By collecting multiple gradients and averaging them (epoch training), more time is spent computing the gradient, but it is theoretically more accurate because of the larger sample size. Thus, for a given amount of time, a trade-off exists between time spent computing the gradient and the number of weight updates possible. The two extremes are computing the gradient over a) the entire training pattern set, and b) a single training pattern.

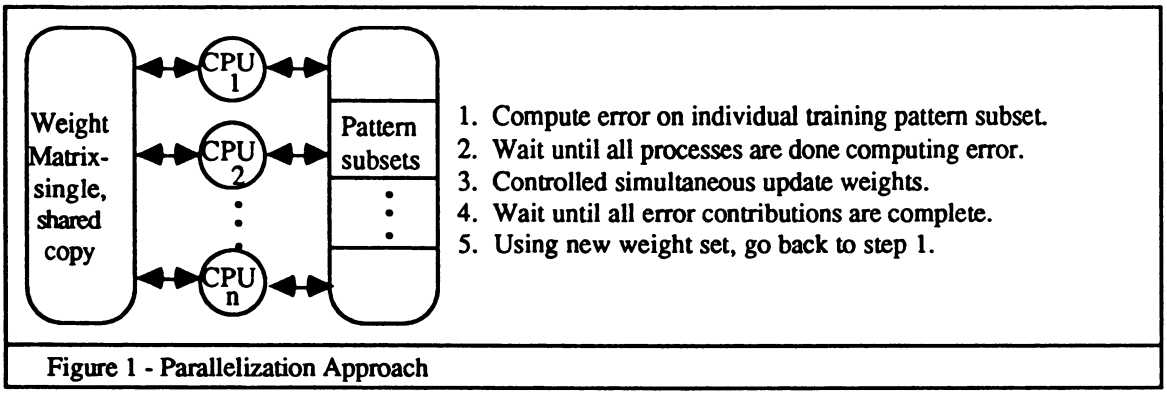


Figure 1 - Parallelization Approach

This trade-off is certainly not unique to the parallel processing paradigm, but it directly affects the parallel training tool's efficiency. If the quality of the gradient estimate improves only slightly using more training patterns, the parallelized training procedure described will not decrease the learning time. Parallelization is only helpful where gradient quality increases with respect to epoch size remain significant. In other words, on the hypothetical curve shown in Figure 2, useful work is done as long as the slope does not level off. After that point, one is only "polishing the stone." More processors can work on the problem, but the total error reduction over time will not increase.

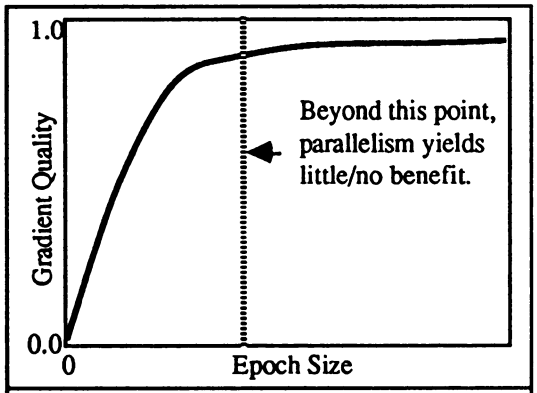


Figure 2 - Gradient quality as a function of epoch size (Hypothetical curve)

For optimal learning efficiency, parallelization among training patterns requires careful attention to the epoch size (ES). Bigger is not necessarily better. See Charts 2 and 3 for graphical summary of results for the object detection example problem. This training set contained 4156 samples and the network was 30x30x10. Table 1 summarizes the environment for each trial used as a data point. In order to generate certain epoch sizes, different numbers of processors were used as listed. The interesting point on Chart 2 is that increasing the epoch size produces lower error up to epoch size of 18. After that point, even though more efficient parallel processing (less synchronization, more number crunching) is occurring, error is not further reduced because the additional time refining the gradient estimation is wasted. Thus, fewer weight updates are made in a given time. Since the gradient estimates are not more accurate, total error is higher. Chart 3 summarizes the twenty test trainings examined. A single learning rate did not perform well for all cases. To negate any advantage, the learning rate is individually specified (Table 1) inversely proportional to the epoch size [Kung88, Juri88]. We discovered empirically that a learning rate of about "5 / epoch size" worked well for nearly all cases.

Discussion

This paper reports incremental results to-date. Several areas require more analysis: (1) the effects and size of parallelism overhead, (2) further evidence for predicting the gradient quality vs. epoch size curve, (3) optimizing the contribution of learning rate and momentum to reduced learning time, and (4) use of different training sets and types of data.

The results presented are specific to the data set studied. We fully expect different results for different problems, sample sets, and types of data. On some data sets, using even more than one

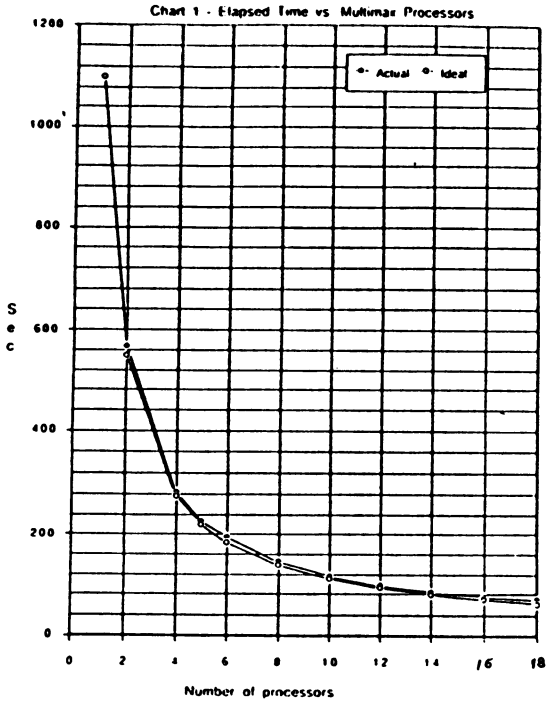
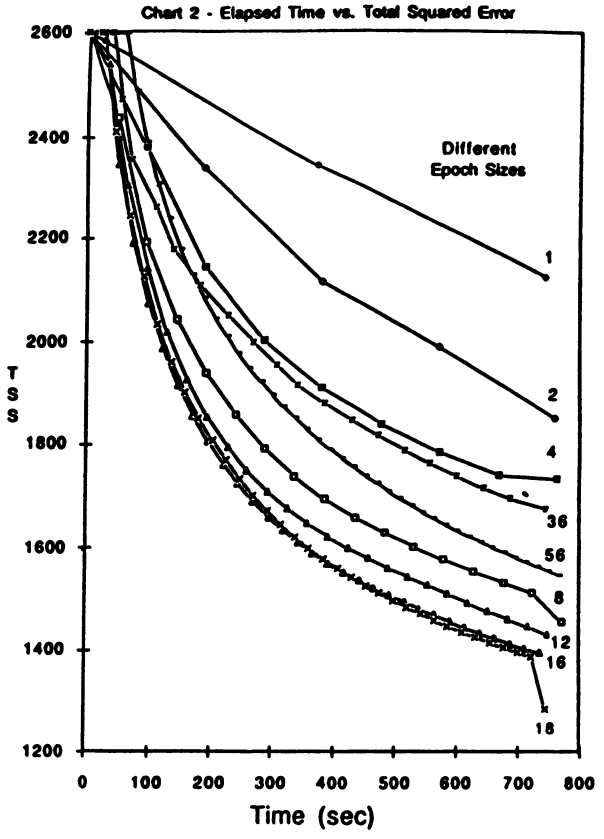
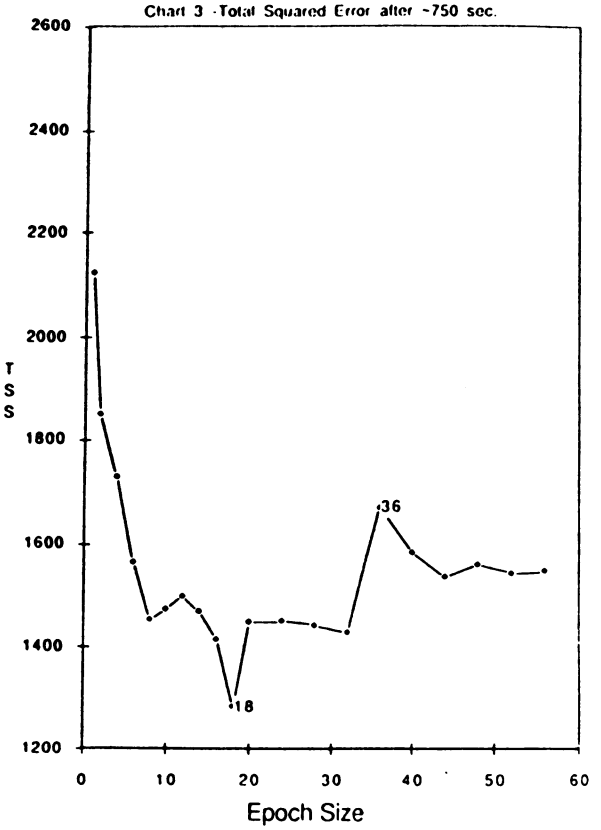


Table 1 - Test Environments and Results

npatterns	4156			tot_num_examples = npatterns * passes				
T_EXAMPLE	0.0028			Tot_ES = ES * nprocesses				
T_STEP	0.0866			steps = (npatterns * passes / Tot_ES)				
				lrate = 5.0 / Tot_ES				
Input File	Tot ES	ES	npasses	nprocesses	steps	act. time	lrate	error
car01.in	1	1	2	1	8312	743	5	2123
car02.in	2	1	4	2	8312	760	2.5	1848
car04.in	4	1	8	4	8312	763	1.25	1731
car06.in	6	1	11	6	7619	701	0.833	1564
car08.in	8	1	16	8	8312	772	0.625	1453
car10.in	10	1	19	10	7896	740	0.5	1473
car12.in	12	1	23	12	7966	749	0.417	1498
car14.in	14	1	27	14	8015	754	0.357	1467
car16.in	16	1	30	16	7793	736	0.313	1414
car18.in	18	1	33	18	7619	745	0.278	1285
car20.in	20	2	25	10	5195	757	0.25	1448
car24.in	24	2	29	12	5022	734	0.208	1450
car28.in	28	2	34	14	5047	740	0.179	1442
car32.in	32	2	39	16	5065	749	0.156	1428
car36.in	36	2	20	18	2309	747	0.139	1674
car40.in	40	4	29	10	3013	757	0.125	1583
car44.in	44	4	31	11	2928	732	0.114	1534
car48.in	48	4	34	12	2944	746	0.104	1559
car52.in	52	4	37	13	2957	747	0.096	1542
car56.in	56	4	41	14	3043	767	0.089	1545





pattern to compute the gradient may be a waste. Parallelism would not help reduce training time and would likely prolong it due to overhead. In other instances, the gradient quality curve might be linear. Linearity and increased utility of the parallel approach might occur where the samples constitute a spanning set for the total input space, and there is, thus, no redundancy in the training set. Other types of data sets need to be studied to allow any general conclusions to be drawn.

The simple empirical evidence presented herein shows the benefit of measuring the quality of the gradient estimate as a function of epoch size for a given problem. The analogy can be drawn between the use of gradient quality through multiple weight updates (time) and forces applied through distance to create work in mechanical systems. Conservation and flow of information from the sample set to the neural net weights is similar to conservation of energy between potential to kinetic forms. In creating a better weight update in parallel, one is increasing the potential "information" energy of the system. To do work and reduce errors, that potential must become active, i.e., iterative weight updates must be applied. The tool described can produce statistical results for finding the optimal epoch size for a given problem, in addition to increasing the efficiency of the training for larger back-propagation neural networks in complex domains of interest.

References

Cate87 "Successfully using peak learning rates of 10 (and greater) in back-propagation networks with the heuristic learning algorithm," J.P. Cater, *Proceedings of the IEEE First International Conference on Neural Networks*, 1987.

Denn87 "Accelerated learning using the generalized delta rule," E. Denning, *Proceedings of the IEEE First International Conference on Neural Networks*, 1987.

Gogg89 "Primacy and recency effects due to momentum in back-propagation," S.D.D. Goggin, K.M. Johnson, K. Gustafson, *Proceedings of the International Joint Conference on Neural Networks*, 1989

Hush88 "Improving the learning rate of back-propagation with the gradient reuse algorithm," D.R. Hush, J.M. Sala, *Proceedings of the IEEE International Conference on Neural Networks*, 1988.

Juri88 "Back error propagation - a critique," M. Jurik; *Digest of Papers: COMPCON*, 1988.

Kung88 "An algebraic projection analysis for optimal hidden units size and learning rates in back-propagation learning," S.Y. Kung, J.N. Hwang, *Proceedings of the IEEE International Conference on Neural Networks*, 1988.

Park87 "Optimal algorithms for adaptive networks: second order back propagation, second order direct propagation, and second order Hebbian learning," D.B. Parker, *Proceedings of the IEEE First International Conference on Neural Networks*, 1987.

Rume86 *Parallel Distributed Processing (PDP): Explorations in the Microstructure of Cognition (Vol. 1)*, D.E. Rumelhart, J.L. McClelland, and the PDP Research Group, MIT Press, 1986.

Sama87 "Refining and redefining the back-propagation learning rule for connectionist networks," T. Samad, *Proceedings of the 1987 International Conference on Systems, Man, and Cybernetics*, 1987.

Shep88 "Fast learning in artificial neural systems: multilayer perceptron training using optimal estimation," J.F. Shepanski, *Proceedings of the IEEE International Conference on Neural Networks*, 1988.

Simo89 "Back propagation learning equations from the minimization of recursive error," W.E. Simon, J.R. Carter, (to be presented) *IEEE International Conference on Systems Engineering*, August 24-26, 1989.

Wata89 "Neural network simulation on a massively parallel cellular array processor: AAP-2," T. Watanabe, Y. Sugiyama, T. Kondo, Y. Kitamura, *Proceedings of the International Joint Conference on Neural Networks*, 1989.

IMPROVEMENT OF AUTOASSOCIATIVE MEMORY MODELS BASED ON PROPERTIES OF BAM'S

C.J. Pérez, J. Carrabina, E. Valderrama and N. Avellana
Centro Nacional de Microelectrónica
Universitat Autònoma de Barcelona, 08193 Bellaterra, Barcelona

Bidirectional associative memories (BAM) [1] are mainly designed to allow heteroassociation between pairs of patterns. The first studies of these systems were very optimistic because they seemed to show excellent results not only because of the high number of patterns that could be encoded but also by the rather easy implementation in VLSI. However, more accurate analysis showed that the performance of a BAM cannot improve that of other well known models of associative memory [2,3,4]. This fact caused an oblivion of the BAM.

The goal of these notes is to renew the interest on BAM by presenting two different results. From one side we present a new form of the matrix of connections which has the advantage that together an architecture suitable for VLSI purposes, allows to increase the effective capacity of the system respect to a general autoassociative network. The point is the reduction of the number of physical connections. As a complement we introduce a method able to codify very efficiently patterns with low levels of spatial activity, i.e, sparse coded. This method is particularly interesting for the architecture presented in this paper.

A BAM is composed by two layers with n and m neurons respectively. Both layers are completely interconnected but there are not connections between neurons of the same layer. The goal of the system is to codify a set of pairs of patterns $((X^1,Y^1).....(X^p,Y^p))$ through of the following matrix of connections $M = \sum_i X^{iT}Y^i$. The dynamics of the system is simple. Taking as initial condition a corrupted version of a pattern, for instance X^k, one evaluates the product X^kM obtaining as a result a version of Y^k. Then, one evaluates M^TY^k leading to X'^k and so on till one arrives to a stable state.

The learning rule that we propose is aimed for autoassociative purposes. Its adaptation needs a modification of the basic structure of the system in order to make profitable the features of the standard BAM. Let's suppose that a set of vectors $Z = (X_1...X_n, Y_1...Y_m)$ of length n+m, built up from the composition of vectors X and Y, must be codified. We construct a matrix of connections M' through the following learning rule

$$\begin{aligned} M'_{ij} &= 0 \quad 1 \le i \le n, \;\; 1 \le j \le m \\ &= M^T \quad 1 \le i \le n, \;\; m+1 \le j \le m+n \\ &= M \quad n+1 \le i \le n+m, \;\; 1 \le j \le m \\ &= 0 \quad n+1 \le i \le n+m, \;\; m+1 \le j \le m+n \end{aligned} \tag{1}$$

where M is given by (1). With such structure our system behaves as a one layer autoassociative network for vectors Z. The dynamics of the model is the usual one. Each element of Z evaluates its state in the next time step from the product ZM', however the structure of M' causes some interesting effects which deserve special attention. The components of the vector Z after one iteration are

$$
\begin{aligned}
X_1 &= Y_1 M_{11} + ... + Y_m M_{m1} \\
&\cdot \qquad\qquad \cdot \\
&\cdot \qquad\qquad \cdot \\
Y_m &= X_1 M_{1m}^T + ... + X_n M_{nm}^T
\end{aligned} \tag{2}
$$

Therefore, the dynamics of the standard BAM is preserved. The subset X of Z is recalled through M whereas Y is recalled through M^T. Notice that the new dynamic process is composed by two independent steps. First, we evaluate X(t+1) from the initial values of Y(t). If the initial version of Y is , for instance, very corrupted, then the retrieval of X(t+1) will be very poor even if we begin the process with a correct version of X. The same reason leads to a very good retrieval of Y(t+1) because it comes from a good version of X(t). In the second step we find opposite behaviours. If the initial X was good the retrieval of X(t+2) is expected to be good (if there are not problems of saturation) and consequently bad for Y(t+2). Therefore, we have to difference two cycles, each with its own features which depend on the initial overlap of the two subsets of Z, X and Y. The reason for this effect born in the structure of M'. One observes two submatrix with no links between them which causes two independent processes. In this way we have introduced partitioning.

VLSI analog neural networks implementing associative memories, have been built following different design methodologies [5,6], mainly with architectures based on the Hopfield-model (Fig 1). The results obtained in terms of silicon area are highly dependent on the number on interconnections, n^2 in a Hopfield net, rather than on the number of the neurons n. Our system behaves as an autoassociative one-layer network for vectors of length n+m. Because of the symmetry and of the dilution our matrix is composed by two zero sub-matrix plus two sub-matrix, one the traspose of the other. This structure can be profitable from an architectural point of view, to reduce one half the number of implemented synapses.

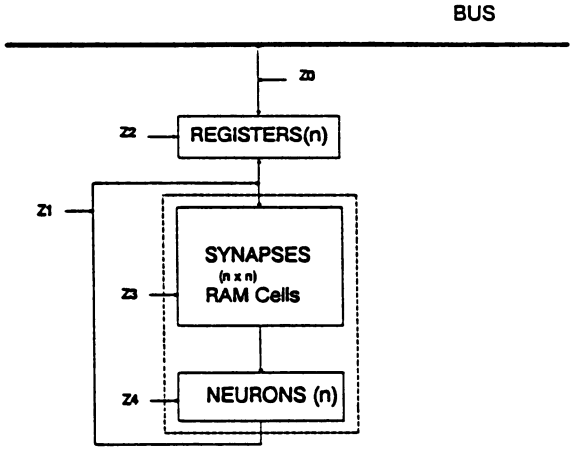
One can obtain autoassociativity on vectors Z=(X,Y) by multiplexing the calculation. To achieve that, we use a "programmable combinational resource", composed by the interconnection matrix and the neurons. In this sense programmable means not a new load for RAM but a control of the direction of the process or, in other words, to transpose the matrix M. In odd cycles, the resource implements M matrix, so Y' is computed from the product of X by M, whereas in even cycles, M^T is implemented and X' is computed from the product of Y by M^T. So every two clock cycles we have an updated value of Z.

In terms of architecture one needs, for each interconnection:

- A number of RAM cells which depends on the values that weights can take, and its associated circuitry.
- One device simulating the synapse.
- One device and I/O and control lines, for bidirectionality.

Additionally, two layers of neurons with programmable threshold for sparse coding, and two registers per neuron, to store X, X', Y, Y'.

The design (Fig.2) is composed by an external bus (an usual eight or sixteen bits bus), used to load the weights into the RAM cells of the interconnection matrix and also for data, through a register in a multiplexed way. Internally each register is split in two parts of



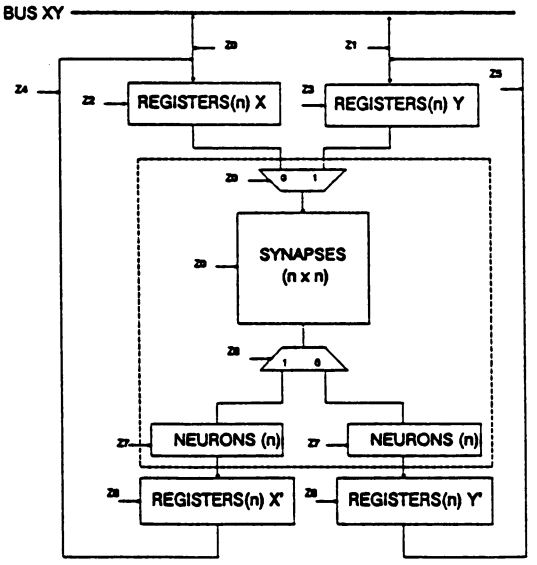


Fig 1:Architecture based in the Hopfield model. Fig 2:Architecture based in our BAM.

lenghts n and m. BAM is showed as a programmable (X to Y, Y to X) resource, controlled by the same line that controls internal multiplexers.

Evaluating our architecture in terms of devices we have the same number of neurons n+m, whereas the number of interconnections is nm ,instead of the expected 2nm for this type of system without multiplexing. The stable state condition not ensures an error-free recall becuase of the existence of two independent processes which produces two different values for each register part, X and Y, for even and odd iterations. This additional condition can be used as an error-free test, for fault tolerant computing purposes. Otherwise one must take a decision about taking values associated with even or odd cycles for vector Z.

This process can be expanded towards a great number of parts, using more BAM resources, and using different fault tolerant conditions (two of three, two of four, etc.). It is a very powerful method of partitioning, not possible with autoassociative memories because of the independence of different parts.

We also present a BAM capable of codifying in a very efficient way sparse coded patterns. The strategy that one has to adopt goes through the study of the non-homogeneous BAM whose most relevant feature respect to the homogeneous BAM is the effect caused by non-zero thresholds. Let's consider a set of p pairs of patterns (X,Y) with n and m neurons respectively. We assume that $\frac{(1+a)n}{2}$ neurons are active in each X and $\frac{(1+b)m}{2}$ neurons are active in each Y. This new BAM has the following energy function

$$E = -XMY^T + XS^T + TY \tag{3}$$

where $S = (S_1...S_n)$ and $T = (T_1...T_n)$ are the thresholds. We propose for each M in (1), S and T the following expressions [7]

$$M = \sum_i (X^{iT} - a)(Y^i - b)$$

$$\begin{aligned} S^T &= aM + S'^T \\ T &= bM + T' \end{aligned} \tag{4}$$

We observe that the first term of S is correlated with the pairs of patterns (X,Y) learned by the network. We also observe a prefactor (a or b) related with their level of activity. Both elements are fundamental in our analysis. Moreover, we introduce another factors S', T' that we will consider as a constant thresholds. We are now interested in the evaluation of the number of patterns p which can be stored perfectly (no errors in the retrieval process) by our BAM. We consider as a simplification that the number of neurons in both layers is the same n=m, and the level of activity is also identic (a=b). The basic idea is to analyze the stability of a pattern through the study of the dynamics of the system. Following a standard process [4] (no explained here because of the brevity) we have found

$$p \approx \frac{N}{2(1-a^2)[2ln(N) - ln(1-a^2)]} \tag{5}$$

We observe that if the patterns are highly correlated,($a \rightarrow 1$), p increases notably. It is of interest the study of some limit cases. For instance, when $a = 0$ the expression (5) reduces to $p \approx \frac{N}{4ln(N)}$. This well known result [4] shows that the BAM cannot overreach, for unbiased patterns, the capacity of the Hopfield model. If a scales with N as $N \approx (1-a^2)^{-1}$

$$\frac{p}{N} = \alpha \approx \frac{1}{6(1-a^2)ln(1-a^2)} \tag{6}$$

This is also a well known result found by Gardner [8]. It is interesting to observe that the process followed in this notes causes the performance of the BAM to be nearly optimum since the functional form of (6) is an upper limit for associative networks. Moreover, we have found a relationship between the sparseness and the size of the network, which can be of interest for finite systems. From expression (5) we extract additional properties of the BAM. The storage capacity of a standard BAM is determined by only an element, the number of neurons of the smaller layer. However, when sparse coding appears on the scene, there is a competition between two elements, the size of the layer and the level of activity of the patterns encoded. This fact could lead to interesting effects useful for applications.

References

[1] Kosko B., IEEE Trans. on Systems, Man and Cyber. **18** (1988) 49

[2] Hopfield, J.J., Proc. Natl. Acad. Sci USA **79** (1982) 2554

[3] Amit D.J., Gutfreund H. and Sompolinsky H., Ann. Phys. (NY), **173** (1987) 30

[4] McEliece R.J., Posner E.C., Rodemich E.R. and Venkatesh S.S., IEEE Trans. on Information theory **33** (1987) 461

[5] H.P. Graf and P. de Vergar, Proc. of the 1987 Standford Conf. on Advanced Research in VLSI, MIT Press 1987, pp 351-369

[6] M.A. Sivilotti, M.R. Emerling and C.A. Mead, AIP Conf. Proc. 151, pp 408-413, 1986.

[7] Perez-Vicente C.J. and Amit D.J.,J. Phys A,**22** (1989) 559

[8] Gardner, E., J. Phys. A, **21** (1988) 257

DIGITAL IMPLEMENTATION ISSUES OF STOCHASTIC NEURAL NETWORKS

E. E. Pesulima*, A. S. Pandya§*, and R. Shankar*

***Department of Computer Engineering**

§Center for Complex Systems

Florida Atlantic University, Boca Raton, FL 33431

I. ABSTRACT.

Recent reports have shown the feasibility and advantages of implementing stochastic nets in digital hardware. We propose a simple digitally implementable formulation of a smooth sigmoidal transfer function for the probability distribution function. Simulation results are briefly described for the implementation of the sigmoid in a Boltzmann machine architecture. The Traveling Salesman Problem was selected as a benchmark simulation problem.

II. INTRODUCTION.

Several attempts have been made to implement the neural network computational paradigm in hardware. Analog implementations suffer from the problems of the implementation of variable resistances and the interconnection complexity. This has led researchers to find alternative approaches, such as implementing stochastic nets in digital hardware [2]. An added benefit of the stochastic approach is that it provides an avenue for simulated annealing [1]. We propose a stochastic approach to digital hardware implementation of neural networks. Essential to the implementation of stochastic digital nets is the availability of a simple, effective, hardware implementable random number generator that can generate random numbers with a Gaussian probability distribution, which in turn will give the needed sigmoidal cumulative distribution function. We will now describe a digital architecture for neural networks that addresses these issues.

III. GENERATING A USEFUL SIGMOIDAL TRANSFER FUNCTION.

The most widely used sigmoid function is the Boltzmann distribution function. The method used to get a cumulative distribution function of that form is to generate a uniform random number and compare it with the output of the Boltzmann distribution function and pulse the output if the random number is smaller. It is, however, difficult to approximate the smooth sigmoidal Boltzmann distribution function in digital hardware. An alternative approach is to generate random numbers with a Gaussian distribution and compare that directly with the input. Simple digital hardware designs, however, can only generate uniform pseudo-random numbers. Obtaining a Gaussian from the uniform distribution usually involves a large sample of uniformly distributed random numbers and application of the Central Limit Theorem to get the Gaussian. Both methods thus require complex random number generators, implemented with a special unit or with software methods via a host computer connected to the digital hardware. We propose here a different sigmoidal function, g(x), that will be much easier to implement in digital hardware:

$$g(x) = 2^{(x-t)/T} = 2^{(x/T)}/2 = 1/(2*2^{(|x|/T)}) \qquad x \leq 0$$

$$g(x) = 1 - 1/(2*2^{(x/T)}) \qquad x > 0$$

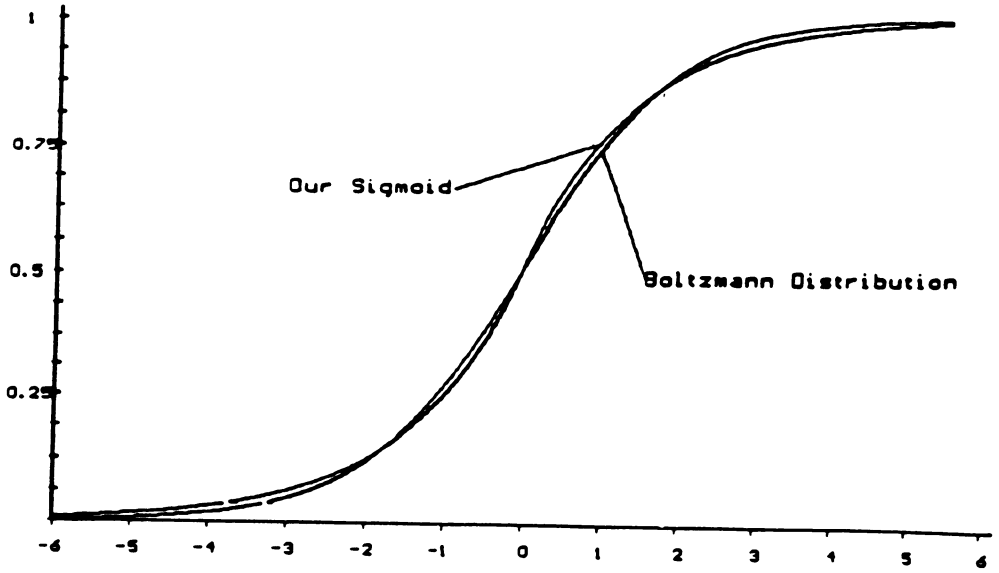


Figure 1

where T is the temperature and x is the input. Figure 1 shows our sigmoid in comparison with the Boltzmann distribution function. Both were drawn with T = 1. T would be varied to incorporate simulated annealing in the network.

The function is easily implemented in hardware by using a Pseudo Random Binary Sequence Generator(PRBSG) implemented using a Linear Feedback Shift Register(LFSR). Each bit in the LFSR of a PRBSG has a probability of 0.5 of being a 1. Logic gates can be used to generate different probability values; two signals with probabilities p_1 and p_2 of being a 1 would, for example, result in a signal with probability $p_1 * p_2$ of 1 if put through an AND gate and a probability $p_1 + p_2 - p_1 * p_2$ if put through an OR gate. Similarly, for an inverter, if the input has probability p_1 the output has a probability value of $1 - p_1$ [3]. The characteristic of the inverter allows us to generate only half of the sigmoid function, say [0,0.5], and put that output through the inverter to get the other half. In our sigmoid above, the sign bit will determine which half to use. If the input ($|x|/T$) = 0 the probability value is 0.5; to get this we could just use the value of one LFSR bit. In figure 2 the register shown contains the magnitude of the input after

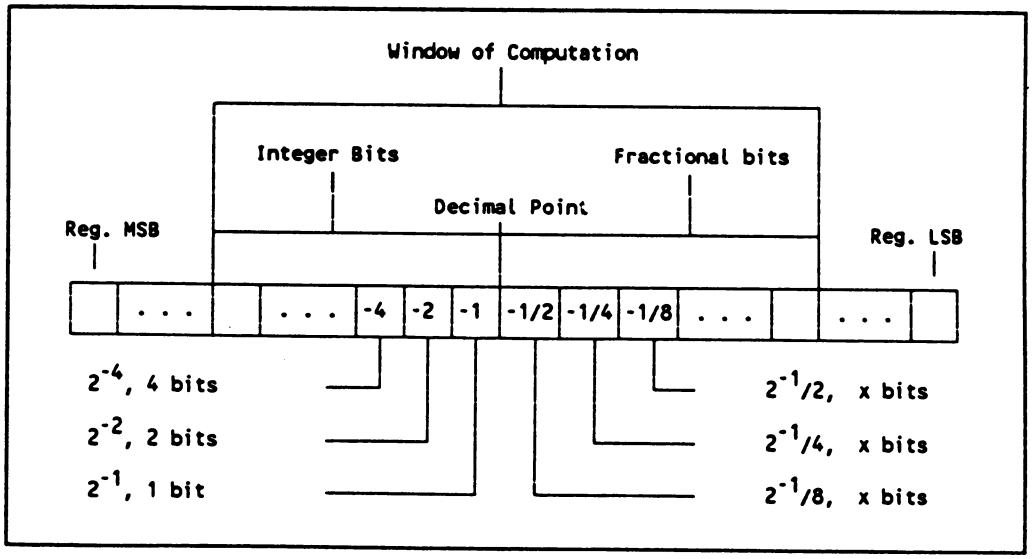


Figure 2

division by T. Each bit represents powers of two, integer and fractional, of the sigmoid. Also shown are the number of bits used for each power. For integer powers, the bits are simply ANDed together. For fractional powers, the number of bits could be varied according to the desired accuracy. Calculating $2^{-1/2}$ with 4 bits would approximate it with 11/16; this could be generated by the use of AND and OR gates. To obtain the sigmoid simply AND the probability values of all the bits that are on together with an extra LFSR bit. We observe that for large negative integer powers of 2 the probability is close to zero, while for small negative fractional powers of 2 the probability is close to 1/2. This would suggest using some cut off point implementing a "window" of computation that includes both integer and fractional powers of 2. Figure 2 illustrated a six-bit window. The wider the window the better the approximation. This sigmoid is part of a digital architecture that will operate wholly in integer arithmetic, thus resulting in a faster and much more space efficient implementation in hardware.

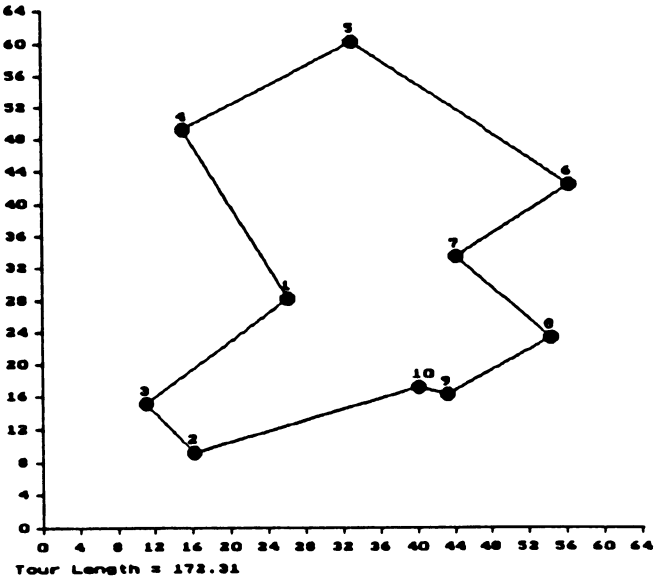


Figure 3

IV. SIMULATION RESULTS.

We implemented the sigmoid in a Boltzmann machine architecture and performed simulation on the Traveling Salesman Problem. We chose to use the original 10 city problem used by Hopfield and Tank [4] transposed to a 64 X 64 square area from the original 1 X 1 square area, a necessary step forced by the use of integer arithmetic. Figure 3 shows the minimum path solution that was found by the network on several occasions. The performance of the network was as expected: at high temperatures there is a large degree of randomness, and as the temperature is reduced the network converges to a stable state. The cost function proposed by Hopfield [4] has parameters that are not easy to establish and their values do not scale easily with the size or the spatial configuration of the problem. We propose a cost function with parameters that are easily established for a given problem and do scale well with the size of the problem. We define the connection weights between the neurons i and j in a matrix configuration for the neurons [4] as follows:

$$G_{ij} = \begin{cases} D_x(D - d_{ij}) & \text{for neighboring cities} \\ -D_{max} & \text{for row/column inhibition} \end{cases}$$

Parameter d_{ij} is the distance between the neighbors. Parameter D plays an important role similar to the biasing current in Hopfield's formulation. Larger values of D give longer tours but more valid convergences; while smaller D values give better tours at the cost of more invalid convergences. Parameter D_x could be varied to control the level of the critical temperature during annealing; while D_{max} should be set large enough to prevent more than one neuron on in a row or column. Note that in this formulation the scaling related to the spatial configuration is dealt with in a straightforward manner. Also, scaling the number of cities does not complicate matters. Gutzmann proposed a similar cost function that also scales well but often leads to less optimum and more invalid solutions [5]. The scaling properties of the Hopfield cost function compared to ours is analogous to that of the a portable versus a nonportable program in conventional computers. Table 1 summarizes the results of preliminary simulation runs on the 10 city TSP.

Table 1

Exp. #	# of Tries	# of Valid Conv.	D	Dx	Tour Length	
					Avg.	Min.
1	10	9	45	2	209.54	188.21
2	20	20	45	2	203.97	172.31
3	10	7	35	8	199.39	176.96
4	50	20	30	4	191.45	172.31

The results were obtained with the following annealing schedule:

$$T(i) = 2^{To - (i/30)}$$

where 2^{To} is the initial temperature, with To initially set to 5, and i is the iteration number; i/30 was obtained in integer arithmetic. Note that even for this very rough schedule (chosen for its simplicity, since changing temperature just involves a shift operation in hardware) the network still performed well. The minimum tour length of 172.31 shown in figure 3 and table 1 is the same one found by Hopfield (172.31/64 = 2.69 $\cong$ 2.71, their optimum tour)[4].

In conclusion, we have proposed a different sigmoid and showed how that sigmoidal cumulative probability distribution function could be realized with simple digital hardware. This would in turn allow the placement of these PRBSG in each VLSI chip or several PRBSG's on each VLSI chip that implements the network instead of the one random number generator per circuit board to be shared by several chips as suggested by [2], thereby resulting in a reduction of the complexity and an enhancement of the speed.

V. REFERENCES.

[1] D. H. Ackley, G. E. Hinton and T. J. Sejnowski, "A Learning Algorithm for Boltzmann Machines," *Cognitive Science*, vol. 9, 147-169, 1985.

[2] D. E. Van den Bout and T. K. Miller, "A Digital Architecture Employing Stochasticism for the Simulation of Hopfield Neural Nets," *IEEE Trans. Circuit and Systems*, vol.36, no. 5, 732-738, May 1989.

[3] P. H. Bardell, W. H. McAnney and J. Savir, *Built-In Test for VLSI: Pseudorandom Techniques,* John Wiley & Sons, New York, NY, 1987.

[4] J. J. Hopfield and D. W. Tank, " "Neural" Computation of Decisions in Optimization Problems," *Biological Cybernetics*, vol. 52, 141-152, 1985.

[5] K. M. Gutzmann, "Combinatorial Optimization Using a Continuous State Boltzmann Machine," *IEEE Int'l Conf. on Neural Networks, I,* III-721 to III-734, 1987.

INTEGRATING DIGITAL AND ARTIFICIAL NEURAL NETWORKS USING NEUROCONTROLLERS: AN INTERMEDIATE STEP TOWARD THE UNIVERSAL COMPUTER

Luis Rabelo*, Doo Kim*, and Temel Erdogan**
*Computer Integrated Manufacturing Laboratory
Engineering Management Department, University of Missouri-Rolla
Rolla, Missouri 65401
**Boeing Aerospace Research and Technology
Kennedy Space Center, Florida 32899

ABSTRACT

This paper presents a computer architecture based on the integration of digital and artificial neural networks. The goal is to demonstrate the opportunities that an effective connection between neurocomputing and conventional computing might offer. A small instruction set computer (SISC) was designed using a hardwired controller and A Hardware Programming Language (AHPL). The hardwired controller was replaced by a neurocontroller that utilizes several artificial neural networks in order to achieve speed and possibilities of modifiable behavior.

1. INTRODUCTION

Computer architecture is a universal concept that integrates hardware and software to perform a variety of processing activities. These processing activities usually have different natures that affect the strategy utilized in order to provide an efficient computing process. Consequently, functional models of computational systems should be developed.[4]

This research studies a computer architecture based on the integration of digital and neural networks to provide an effective connection between neurocomputing and numerical computing environments. In order to accomplish our goals a small instruction set computer (SISC) was designed. Originally, this computer was designed using a hardwired controller and AHPL.[3] This computer was redesigned, replacing the hardwired controller by a neurocontroller that utilizes several artificial neural networks (ANNs) to achieve various characteristics including the speed of hardwired controllers and possible modifications in the processor's behavior as microprogramming does.

This computational scheme will provide a powerful interface to other computing environments such as neurocomputing and symbolic processing. This scheme also provides the opportunity for a futuristic hierarchichal parallel distributed processing architecture (HPDP). The high level elements of a HPDP architecture will distribute the tasks to perform to the appropriate computing elements. HPDP could take advantage of the natural characteristics of ANNs in order to utilize metalearning[8] procedures and become an evolvent computer.

2. DESIGN OF A SMALL INSTRUCTION SET COMPUTER USING AHPL

For this research, a 16-bit small instruction set computer (SISC) was designed and simulated.[2] This design created a simple architecture, with the required functional capabilities. These capabilities should demonstrate the adequacy of a general purpose digital computer for certain kinds of computing environments (e.g. numerical computing). The system configuration of this particular design has a collection of registers and busses to allow the necessary data transformations (See Figure 1). The set of instructions is classified by the following

groups:

a. Fifteen memory reference instructions with four addressing modes;
b. Thirty operate instructions;
c. Eight input/output (I/O) instructions.

The nature of the operations in this computer is cyclical, and consists of the sequential fetching and execution of instructions --the degree of parallelism was kept to a minimum at the instruction level to reduce the complexity of design. Consequently, a control strategy was incorporated to perform these operations in the registers. This control strategy was implemented using a hardwired controller.

The design phase involved the utilization of AHPL. The AHPL sequences provided a complete description of the sequential digital network required to control the computer. The detail of this hardware description language allowed the simulation of the computer and a clear picture of the digital network realization.

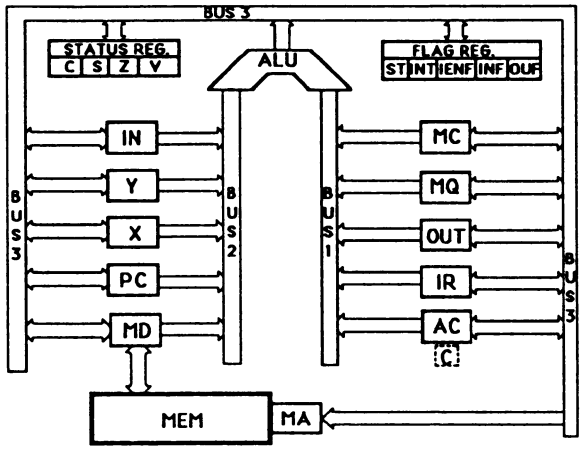


FIGURE 1. SMALL INSTRUCTION SET COMPUTER.

3. DESIGN OF A NEUROCONTROLLER FOR A DIGITAL COMPUTER

ANNs have commonly been applied to control the behavior of robotics systems and manufacturing processes.[9] Their characteristics of graceful degradation, real-time learning, and massively parallel-distributed processing make ANNs candidates for controlling the sequential actions required for certain computing tasks. ANNs might provide a control with a fast decoding of machine language instructions and flexibility to be modified.

3.1 Architecture

Based on the factors mentioned above, the hardwired controller was replaced by a neurocontroller in the computer designed. The neurocontroller utilized is composed of several ANNs(see Figure 2). These ANNs decode instructions and generate time sequences. The ANN A is a three-layer feed-forward network that was trained using the Back-Propagation learning algorithm. This network decodes the instruction stored in the instruction register(IR) and feeds the ANNs B, C, and D.

The ANN B is a three-layer network that assisted by the ANN C reproduces the Jordan's network architecture.[5] The structure formed by the ANNs B and C allows the generation of time sequences and inclusive loops and jumps in the sequence of patterns of a specific plan. These loops and jumps are needed to implement certain hardware algorithms such as multiplication. The output of ANN B is fed to the ANNs C and D.

The ANN C is a three-layer feed-forward network that was trained using the Back-Propagation algorithm. This network receives the output of the ANN B, fed by the decoded contents of the IR, and the thresholded output of several

registers and status flags. ANN C is responsible to copy back the output of the ANN B to the current-state units of the ANN B. This process is modified if the contents of a register (involved in the execution of a instruction) trigger a different response to feed to the current-state units of the ANN B. This "new" output pattern will force a change for subsequent patterns in the sequence of the current plan.

The ANN D is a three-layer feed-forward architecture that was trained using the Back-Propagation paradigm. This network receives an input pattern from the ouput from ANN B and the decoded contents of the IR, and then generates the corresponding control signals.

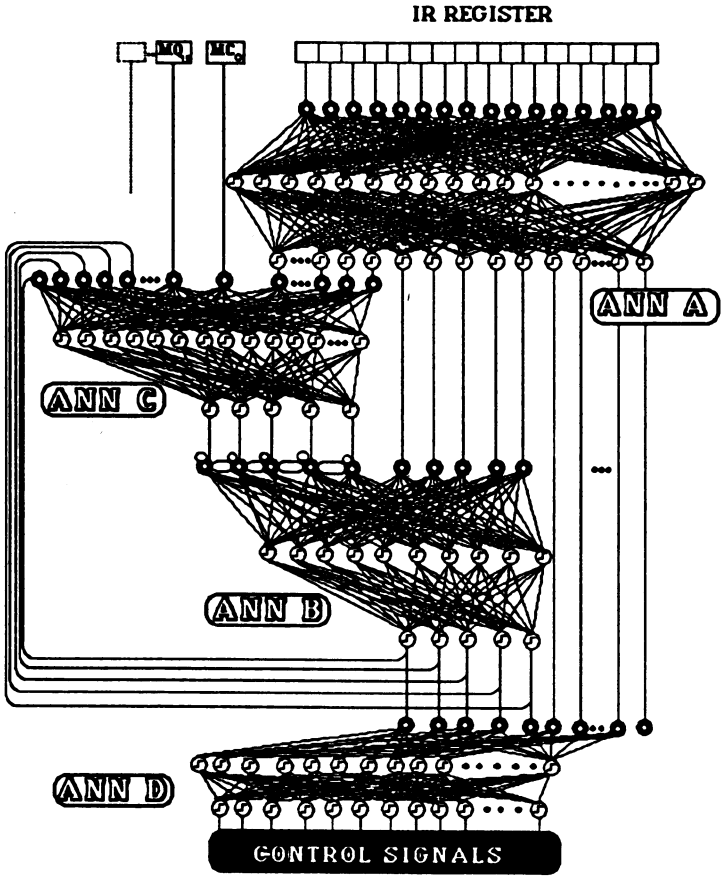


FIGURE 2. NEUROCONTROLLER ARCHITECTURE.

3.2 Training

The Back-Propagation algorithm was utilized. Dynamic node creation,[1] dynamic learning rate adjustments, and combined subset training[7] were used to assure an accelerated and efficient learning process in each of the networks described.

Dynamic node creation (DNC) was utilized to achieve accuracy in the input to output mapping and an efficient size for the hidden layer.

The learning rate and the momentum were adjusted dynamically. The learning rate was sometimes incremented to accelerate convergence and decreased to avoid oscillations, and divergence.

To accelerate the learning and increase the efficiency of the learning process, combined subset training (CST) was utilized in some of the learning sessions.

ISZ (USING DIRECT ADDRESSING)

DESCRIPTION: Increment the contents of the memory location by 1 and skip the next instruction in sequence if the result is zero.

ANN A OUTPUT	ANN B OUTPUT	ANN B CURRENT U.	ANN C OUTPUT	ANN D AHPL STATEMENT
11001	00001	00000	00001	BUS2=MD; BUS3[1:16] = INC1[0:15](BUS2); MD <= BUS3[1:16].
11001	00010	00001	00010	M*DCD(MA) <= MD; BUS2[6:15]=PC*(~(+/MD)); BUS3[7:16]=INC2[0:9](BUS2[6:15])*(~(+/MD)); PC*(~(+/MD)) <= BUS3[7:16].
11001	00011	00010	00011	BUS2[6:15]=PC; BUS3[7:16]=INC2[0:9] (BUS2[6:15]); PC <=BUS3[7:16].
11001	10000	00011	10000	BUS3[7:16]=PC;MA<=BUS3[7:16].
11001	11000	10000	11000	MD<=BUSFN(M;DCD(MA)).
11001	11100	11000	00000	BUS3[1:16]=MD;IR<=BUS3[1:16].

TABLE 1. EXECUTION OF ISZ.

3.3 Simulator

A simulator of the computer using the neurocontroller was made using C programming language in an IBM 6152 workstation. As an example, the execution of an instruction is explained in Table 1.

4. CONCLUSIONS AND FURTHER RESEARCH

The design and simulation of a SISC computer using a neurocontroller has demonstrated the possibility of an effective connection between neurocomputing and conventional computing environments (when the necessary hardware becomes available). Also, more powerful ANN architectures are required, specifically those needed for sequential formalisms.[6] Further research is needed to develop hybrid hardware description languages. These hybrid hardware description languages should have primitives to identify parallel and sequential hardware patterns. This development will lead to optimum hardware implementations.

REFERENCES

1. T. Ash, "Dynamic Node Creation in Backpropagation Networks," ICS Report 8901, Institute for Cognitive Science, University of California, San Diego, 1989.
2. T. Erdogan, "Design and Simulation of a Small Instruction Set Computer by Using A Hardware Programming Language," Master Thesis, FIT.
3. F. Hill, and G. Peterson, DIGITAL SYSTEMS: HARDWARE ORGANIZATION AND DESIGN, John Wiley & Sons, 1987.
4. D. Hillis, THE CONNECTION MACHINE, The MIT Press, 1985.
5. J. McClelland and D. Rumelhart, EXPLORATION IN PARALLEL DISTRIBUTED PROCESSING, The MIT press, 1988.
6. J. McClelland and D. Rumelhart, PARALLEL DISTRIBUTED PROCESSING, vol. 1 and vol. 2., The MIT Press, 1986.
7. F. Tsung and G. Cottrell, "A Sequential Adder using Recurrent Networks," Proceedings of the IJCNN 1989 Conference, pp. II133-II140.
8. D. Stubbs, "Neurocomputers," SAIC publication, 1989.
9. P. Werbos, "Backpropagation and Neurocontrollers: A Review and Prospectus," Proceedings of the IJCNN 1989 Conference, pp. I209-I216.

A Dataflow-Based Neural Net Multiprocessor*

Behrooz Shirazi and Chungching Wang

Department of Computer Science and Engineering
Southern Methodist University
Dallas, TX 75275

Abstract

Neural nets have been shown to have the capability to learn and to remember. As more and more supporting theory and applications for neural nets are being explored, the hardware implementation of neural nets becomes more important. However, most of the existing hardware implementations of neural nets suffer from various weaknesses. This has led to the design of a dataflow-based multiprocessor system for efficiently simulating asynchronous, highly parallel neural nets.

This paper presents the architectural requirements for a suitable neural net implementation. The organization of the proposed design is outlined. A neural net-dataflow graph transformation technique is introduced which guarantees the executability of neural nets on the proposed dataflow-based system. Simulation results show the validity of this approach. A speedup method is also presented for increasing the parallelism in neural net operations. The scheme is based on dataflow's coloring mechanism which allows parallel execution of network iterations.

1. Introduction

Contrast to advances and developments in theory and applications of neural nets, hardware support for these networks has been sporadic. Although analog VLSI and optical approaches are attractive means for implementing neural nets and a number of circuits for this purpose have been developed, current microelectronic techniques face severe technological restrictions. Therefore, almost all current neural nets implementations are still emulated by either software simulation on conventional computers or using special-purpose neurocomputer coprocessors [1-4]. However, these approaches suffer from inadequate parallelism and/or high communication overhead and/or restriction to only small-to-medium neural nets, and being unable to perform truly asynchronous neural operations. Based on these observations, a different machine architecture and technology suitable for neural nets becomes demanding.

The dataflow model of computation offers an effective approach for achieving parallel computation and may serve as a good engine in the neural net implementation. The basic idea of dataflow model is that data flows from instruction to instruction directly rather than via a shared variable and rather than a program counter or other central control mechanisms, instructions may execute ("fire") any time after arrival of the data they require [5]. We found that neural nets and dataflow model have many characteristics in common: i) they are represented by directed graphs with nodes and links; ii) in both models each node functions as a simple processing element and operates asynchronously with other nodes; and iii) both models execute value-passing computations. These findings have motivated us to design a dataflow-based multiprocessor as an effective engine for neural nets implementation.

2. The Proposed Approach

The ideal architecture for neural nets must i) contain many simple, yet massively parallel, processing elements which are able to be asynchronously operating; ii) have adequate local memory in each processing element to hold the states of neurons ,the weights , and the program itself; iii) consist of processing element(s) capable of handling I/O activities; iv) provide a high communication bandwidth between

* This research is in part supported by a grant from DARPA.

processing elements; and v) be cost effective.

To satisfy the above requirements, we propose a fine-grained, loosely-coupled, dataflow-based multiprocessor design where each computing node is a dataflow processor (in current design they are static architecture) connected through a point-to-point link interconnection network. A noticeable point is that the loosely-coupled processors can have a separate I/O system from the interface of host computer. This feature enables a particular sub-set in a large neural net to take specific I/O data, which is of no concern to other portions of the net, without overloading the host interface and routing the message through the interconnection network. In such an architecture, the communication between neurons in different processors is asynchronous. In addition, neurons within the same processor also communicate in an asynchronous fashion due to the dataflow operations.

2.1. Neural Net - Dataflow Graph Transformation

A dataflow program is described by a dataflow graph, and a specific neural net model is also represented by a topology graph plus its operation algorithm. Though both graphs constructed by nodes and links, they stand for different meaning. Hence, a method to transform a neural net to a corresponding dataflow graph is required such that neural operations can be executed on a data flow-based machine.

Our transformation method is essentially a disassembling procedure which disassembles complicated operations in the neural net into several simpler computations that are executable on the proposed dataflow machine. For example, in a typical neural operation, a node sums N weighted inputs and passes the result through a nonlinearity. We can disassemble this operation into N multiplication nodes, one node which sums these N products, and follow it by the activation function f_T. Figure 1 presents this transformation for node N4 with three inputs ($N = 3$). Here, circles denote nodes and boxes denote input pools. Note that N additional nodes are generated in order to hold the corresponding weights. The transformation for those nodes which reside in the same level and perform the same operation, such as N1, N2, and N3 in the example, are identical. The relative positions among these nodes in the transformed graph also remain the same. Thus, adding or deleting nodes on the original graph later does not necessarily require an overall retransformation. In fact, to reflect the modification, only attaching or removing the corresponding subgraph on the transformed graph is sufficient. For a given learning algorithm, the transformation can be modularly constructed on a neural function to neural function basis. This is because only a few neural functions are involved in any artificial neural net and there is no dependency among those functions except the data dependency.

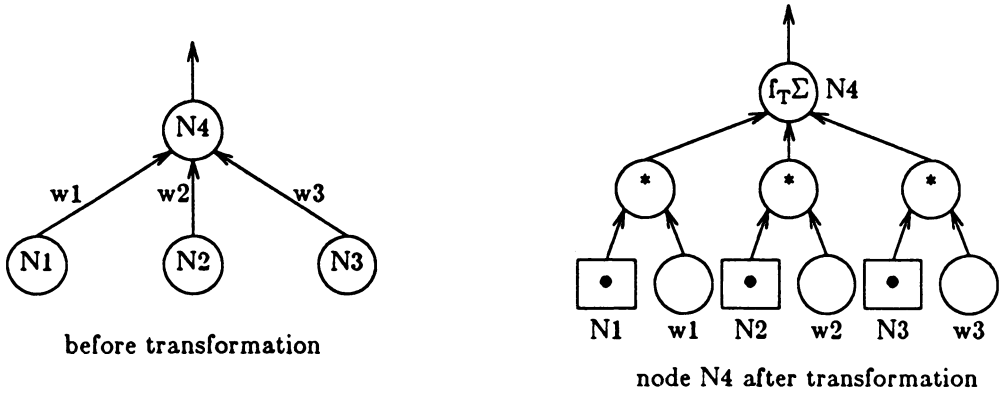
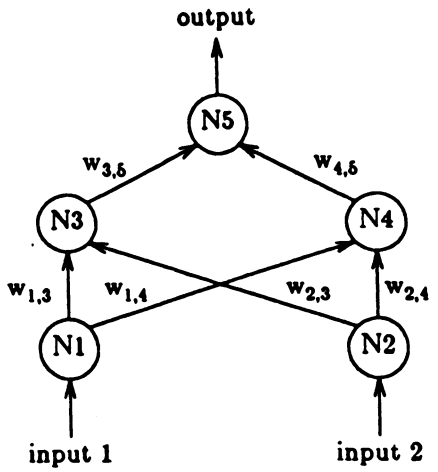
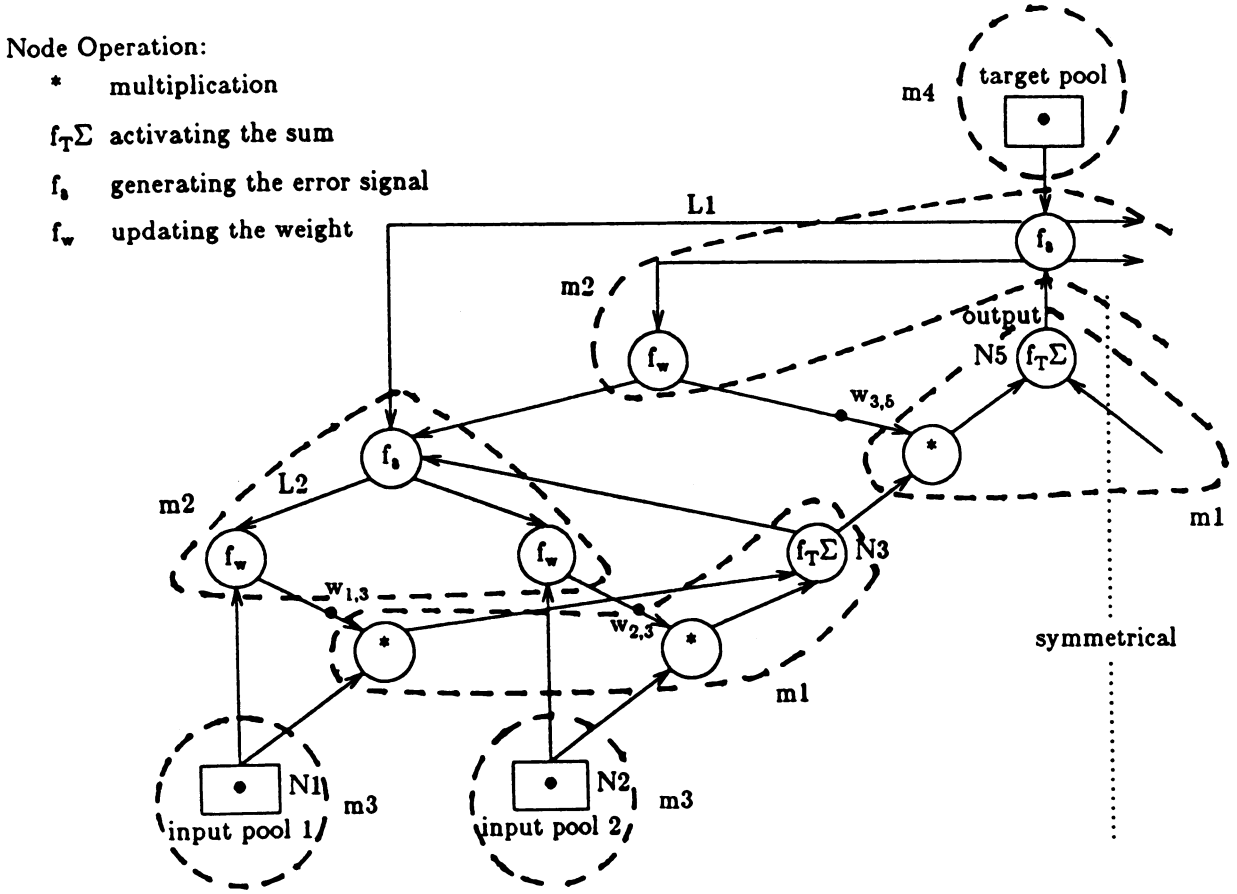


Figure 1. A simple neural net-dataflow graph transformation example

Figure 2(a) presents a topology of a simple three-layered back propagation [6] neural net where the backward links used for propagating errors are not always shown. The neural functions in this neural net can be grouped into four classes of modules. They are (m1) summing the weighted inputs, (m2) updating the weights according to the error signal, (m3) supplying input data, and (m4) supplying target data, respectively. By employing the modular transformation technique, the corresponding dataflow graph can



(a) the original neural net



(b) the transformed dataflow graph

Figure 2. A transformation example of a complete neural net with back propagation algorithm

be obtained as shown in Figure 2(b) in which only the left half is shown. Note that the right half would be identical to Figure 2(b) since the network is symmetric. In the transformed graph, the backward error links appear downward and explicitly, such as L1 and L2. Clearly, the whole transformed graph is composed of several subgraphs circled by broken lines in which each subgraph stands for a module and performs a specific neural function. In a similar way, this transformation can be applied to any neural net model with any number of neurons.

The modular transformation technique has been verified through software simulations based on several different learning algorithms. Simulation results show that the transformed network indeed performs the functions of the original network and does converge with exactly the same learning performance as we expect from a conventional implementation of such neural net.

3. Dynamic Dataflow

Based on the transformed dataflow graph, we found that some nodes are unable to fire until the tokens representing the updated weights and/or the outputs of certain nodes generated from the previous iteration are present. This fact which occurs in most neural nets would reduce the parallelism to a good extent and hence results in a speed bottleneck. Those tokens whose absences would cause a bottleneck are called bottleneck tokens.

In order to remove this bottleneck, we slightly modify the firing rule in the static dataflow model such that those bottleneck tokens will not be consumed even after their destination nodes fire. In this way, a bottleneck token will always be there until a new corresponding bottleneck token arrives and replaces it. This modification allows immediate firing of a node with a new set of inputs. Thus, parallelism is highly increased. However, we must keep track of each set of weights as they relate to a new set of input values. This can be easily achieved through the coloring mechanism of a dynamic dataflow machine. We have also simulated this environment using several neural networks. The simulation results indicate that this deviation makes the modified network keep the same learning capability but with a much smaller number of iterations. In the example of Figure 2, which is used to solve a two-input exclusive-or problem with 500 presentations of target data , the unit time elapsed is 12000 against 20000 in the unmodified network. This is equivalent to a factor of 1.7 speedup.

4. Conclusion

Due to the fact that both current analog VLSI technology and existing emulation approaches suffer respective weaknesses, we have presented a dataflow-based multiprocessor prototype system for efficiently implementing neural nets. A modular transformation technique was also presented to embody the executability of neural nets on such a dataflow machine. Until the technology for direct analog implementation is fully developed, this dataflow approach provides an efficient alternative to realize neural nets.

References

[1] *ANZA Plus VME*, Hecht-Nielsen Neurocomputers, Inc., San Diego, CA 92121, 1988.

[2] *Σ -Series: Workstations for Artificial Neural Systems*, Science Applications International Corp., San Diego, CA 92121, 1987.

[3] DARPA, *Executive Summary of DARPA Neural Network Study*, Lincoln Lab., MIT, Lexington, MA, July 1988.

[4] Toborg, S. and Hwang, K, "Exploring Neural Network and Optical Computing Technologies," in *Parallel Processing for Supercomputing and Artificial Intelligence*, McGraw Hill, 1988

[5] Dennis, J. B. and Misunas, D. P. "A computer architecture for highly parallel signal processing," in *Proc. 1974 Nat. Computer Conf.*, AFIPS Press, Arlington, VA, 1974, pp. 402-409.

[6] Rumelhart, D. E. and McClelland, J. L., *Distributed Parallel Processing: Explorations in the Microstructure of Cognition, vol. 1*, MIT Press, Cambridge, MA, 1986.

Connectionist Production Systems in Local Representation*

Andrew Sohn and Jean-Luc Gaudiot
Computer Research Institute,
Dept. of Electrical Engineering - Systems
University of Southern California, Los Angeles, CA 90089-0781

Abstract: *The importance of production systems (PS) in artificial intelligence has been repeatedly recognized in a number of expert systems. We demonstrate in this paper that the connectionist approach can be applied to the PS paradigm. The three-layers of a ring-structured feedback network with associative memories is considered as an architecture. Characteristics of production systems are identified, based on which mapping PS onto neural networks are done along the local representation. Simulation results show that the proposed approach can be efficient for PS processing.*

1. Problems and Approaches in Production Systems

A production system consists of *production memory* (PM), *working memory* (WM), and *inference engine* (IE). PM (or rulebase) is composed of *productions* (or rules), each of which performs predefined actions if all the necessary conditions are satisfied. The left-hand side (LHS), the condition part, and the right-hand side (RHS), the action part consist of *patterns*. The productions operate on WM which is a database of assertions, called *working memory elements* (WMEs). Both patterns and WMEs have a list of elements, called *attribute-value pairs* (AVPs). The value to an attribute can be either *constant* in lower case or *variable* in upper case. The inference engine repeatedly executes an inference cycle which consists of three steps: (1) *matching* condition patterns of all the productions against the WMEs to determine the satisfied productions, (2) *resolving conflicts* to select one among satisfied productions, and (3) *rule firing* to perform action patterns of the selected production. IE will halt the PS either when there are no satisfied productions or when the desired solution is found.

The PS paradigm described above presents *inefficiencies*. The time taken to match patterns over WMEs can reach 90% of the total computation time [3]. Another weakness is in heavy memory dependency, i.e., all the patterns to be matched must be repeatedly stored and recalled for a new inference cycle. This in turn gives less-adaptability to complex problems and new environments where robustness and self-learning are essential.

To solve the above problems, neural network [6,7] and data-driven multiprocessor [4] approaches have been proposed. The architecture of a neural network proposed in this study has three layers of neurons which form a ring-structured feedback network. A hardware approach is to reduce the matching step, which takes 90% of the total computation time. Insertion of Bidirectional Associative Memories (BAMs) [5] between layers substantially reduces the matching time. At the same time, the memory dependency can be flattened by the proposed architecture and mapping strategy.

2. The Three Layers of Ring-Structured Architecture

The architecture proposed for the PS paradigm consists of three layers, (1) *condition*, (2) *rule*, and (3) *action-layers*, as well as three associative memories, BAM1, BAM2, and BAM3. Fig.1(a) shows the logical organization of the architecture. Neurons in the three layers connected through a ring structure feed information forward. Upon completion of an inference cycle, the newly created state of WM is fed backward for another inference cycle.

* This research was supported in part by AFOSR #88-0274.

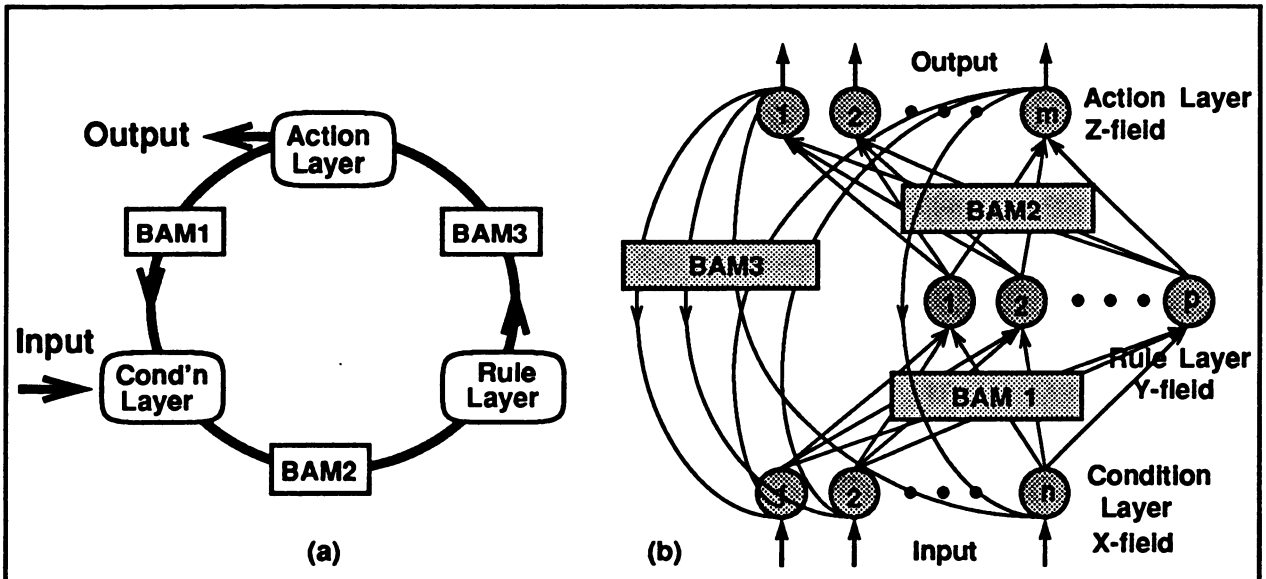


Fig.1: *Three layers ring-structured architecture.* (a) Logical organization, (b) Detailed architecture.

As depicted in Fig.1(b), the condition-layer (*X-field*) which has n neurons $x_1,...,x_n$, $x_i \in \{0,1\}$, reflects the state of WM. If $x_i=1$, the corresponding WME is present at WM. Otherwise, the WME is absent. An initial state of a PS is presented to this layer in the form of a vector X with n elements. The rule-layer (*Y-field*) has p neurons $y_1,...,y_p$, each of which represents one of several instantiations of the rules. All the possible instantiated rules are represented in this layer. The activation of a neuron indicates that the corresponding rule is to instantiate. The action-layer (*Z-field*) holds a set of WMEs indicating *how* and *what* operations it will enforce on WM.

BAM1 holds dependence relations between WMEs and conditions of a particular rule. Given a set of WMEs, rule(s) will be instantiated, thereby turning on/off the corresponding neurons in the rule layer. BAM2 holds information between WMEs and actions of a particular rule. Upon selection and firing of a rule, WMEs in the action layer will be changed by turning on/off the corresponding neurons. BAM3 holds information between the previous and next states of WM. When WMEs are changed due to the rule firing, the neurons in the action layer will reflect the current state to the previous state of the condition layer.

3. Representation of Working and Production Memories

WM at any point in time holds a subset of all the possible instantiation of condition patterns. Let $\rho_i=[(a_1\ v_1),...,(a_n\ v_n)]$ be a pattern consisting of n AVPs. Suppose that ρ_i has in its n value-parts m variables, each of which can be bound to d different values. There would be a maximum of md different WMEs, resulting in md neurons in each X- and Z-layers. A straightforward way of representing the state of WM is assigning a WME to a neuron. Assignment of neurons in the Y-layer for PM is based on *how* the rule be instantiated, i.e., information regarding as which variables bind what values will establish a basis. An example will clarify the assignment process.

Consider below a production system with RULE1 and four WMEs. RULE1 has two condition patterns u_1, u_2 and two action patterns v_1, v_2. The condition pattern u_1 consisting of two

Rule 1	WM
u_1: [(a 1) (b F)]	w_1: [(a 1) (b 2)]
u_2: [(p F) (q G) (r 3)]	w_2: [(a 1) (b 3)]
→	w_3: [(p 1) (q 2) (r 3)]
v_1: -[(a 1) (b G)]	w_4: [(p 2) (q 3) (r 3)]
v_2: +[(p G) (q F) (r 3)]	

AVPs has a variable F. The condition pattern u_2 with three AVPS has variables F and G. Action patterns v_i's are proceeded by '+' or '–' indicating adding to and deleting from WM. Assuming that the variables F and G can be bound to 1,..,3, each of the X and Z-layers can be modeled with 12 neurons since there can be a maximum of 12 different WMEs for RULE1. Table 1 shown below lists all 12 possible instantiations of RULE1 and their assignments to neurons $x_1,...,x_{12}$ of X-layer and $z_1,...,z_{12}$ of Z-layer.

Neurons	WMEs assigned	Neurons	WMEs assigned	Neurons	WMEs assigned
x_1,z_1:	[(a 1) (b 1)]	x_5,z_5:	[(p 1) (q 2) (r 3)]	x_9,z_9:	[(p 2) (q 3) (r 3)]
x_2,z_2:	[(a 1) (b 2)]	x_6,z_6:	[(p 1) (q 3) (r 3)]	x_{10},z_{10}:	[(p 3) (q 1) (r 3)]
x_3,z_3:	[(a 1) (b 3)]	x_7,z_7:	[(p 2) (q 1) (r 3)]	x_{11},z_{11}:	[(p 3) (q 2) (r 3)]
x_4,z_4:	[(p 1) (q 1)(r 3)]	x_8,z_8:	[(p 2) (q 2) (r 3)]	x_{12},z_{12}:	[(p 3) (q 3) (r 3)]

Table 1: *Assignment of all 12 instantiations of Rule1 to neurons of X-layer and Z-layer.*

An initial state of the WM shown above can then be represented in the X-layer as: **X** =(011010001000). Given the initial state, RULE1 is instantiated by w_1 and w_4 with F any G bound respectively to 2 and 3. Selection and firing of RULE1 affects WMEs [(a 1) (b 3)] and [(p 3) (q 2) (r 3)] according to v_1 and v_2. The fact that 2 WMEs are influenced by the rule firing is represented in the Z-layer by turning z_3 off and z_{11} on, i.e., **Z**=(00∅000000010), where z_3 is marked ∅(*null*) to indicate its deletion from WM.

To satisfy u_1 and u_2 of RULE1, the variables F of u_1 and F of u_2 must be bound to the same value v. G can be bound to any value v' such that $v \neq v'$. For the given four WMEs $w_1,...,w_4$, the variables F and G are bound respectively to 2 and 3 to satisfy RULE1. The fact that F and G are bound to 2 and 3 constitutes a particular state of the Y-layer. For RULE1 there can be a maximum of *nine* different combinations of variable bindings, requiring *nine* neurons in the Y-layer. Assigning variable bindings to the neurons $y_1,...,y_9$ is made as follows: y_1:(F=1,G=1), y_2:(F=1,G=2), y_3:(F=1,G=3), y_4:(F=2,G=1), y_5:(F=2,G=2),..., y_9:(F=3,G=3). When the rule is instantiated with (F=2,G=1), the Y-layer is represented as: **Y**=(000100000).

4. Encoding the Three Associative Memories

We shall continue using the above example to show the encoding of three BAMs. Associations for BAM1 is derived from the X and Y-layers. The given four WMEs is represented in the X-layer as **X1**=(011010001000). The corresponding instantiation of the rule for the given WMEs is represented in the Y-layer as **Y1**=(000100000). An association for BAM1 is then obtained as: $A(\mathbf{X1},\mathbf{Y1})_{BAM1}$ = {(011010001000),(000100000)}. Associations for BAM2 is derived from the Y and the Z-layers. Given RULE1 and $w_1,...,w_4$, the Y-layer will have y_4 turned on, resulting in **Y1**=(000100000). According to the action patterns, the Z-layer will become: **Z1**=(00100000∅010). An association for BAM2 is obtained as: $A(\mathbf{Y1},\mathbf{Z1})_{BAM2}$={(000100000), (00100000∅010)}.

After firing of RULE1, WM contains the following 4 WMEs: w_1, w_3, w_4, and w_{new}=[(p 3) (q 2) (r 3)]. This new state is (011010000010), which must be subsequently represented in the X-layer. A simple way of doing this update is a combination of AND/OR operations on two states of X_{prev}= (011010001000) and Z_{prev}= (00100000∅010). A combination of AND operations on the plain binary values and OR operations on the null values gives a new state X_{new}=(011010000010). An association for BAM3 is, therefore, obtained as $A(\mathbf{Z1},\mathbf{X1})_{BAM3}$ = {(00100000∅010), (011010001000)}. Applying the same procedures, remaining associations for the three BAMs can be easily obtained.

5. Simulations and Conclusions

Our simulation is on the blocks world STRIPS [2] with *two* blocks. Assigning binary {0,1} or bipolar {−1,+1} values to associations seems straightforward. However, the actual procedure for the encoding of BAMs is a rather difficult task for two reasons: First, the continuity assumption [5] must be satisfied, i.e., $H(X_i,X_j)/n \approx H(Y_i,Y_j)/p \approx H(Z_i,Z_j)/m$, where $H(*,*)$ is a Hamming distance between two vectors. Second, $H(*,*)$ for any two vectors of the *same* layer need be large enough to distinguish each other from the crowd. Enforcing the following condition, $H(X_i,X_j)>n/2$, $H(Y_i,Y_j)>p/2$, $H(Z_i,Z_j)>m/2$ for $i,j=1,...,8$, and $i \neq j$, we were able to successfully implement STRIPS with 2 blocks in our architecture.

Three BAMs are not listed in this paper due to the space constraint. For details, see [6]. Using our architecture and mapping strategy, matching condition patterns over WMEs reduces to O(1). When the correct initial state is presented to the network, our architecture can find the goal state in four iterations for the problem with two blocks. When a disturbed initial state is presented to the network, irrelevant operators are selected and fired, resulting in an incorrect solution. This is due to the facts that the heuristics which prune the irrelevant search paths are not considered in this study and that our study is mainly in reducing matching time in production systems. In summary, the architecture and mapping strategy we developed in this work substantially reduces the pattern matching time and assimilate the memory dependency of production systems in parallel processing environment by flattening it into layers of neurons.

References

1. Fahlman, S., and Hinton, G., "Connectionist Architecture for Artificial Intelligence" in *IEEE Computer*, January 1987, pp.100-109.
2. Fikes, R., and Nilsson, N., "STRIPS: a new approach to the application of theorem probing to problem solfing," *Artificial Intelligence 2*, 1971, pp.189-208.
3. Forgy, C.L., "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," in *Artificial Intelligence 19*, September 1982, pp.17-37.
4. Gaudiot, J.-L., and Sohn, A., "Data-driven Multiprocessor Implementation of the Rete Match Algorithm," To appear in *IEEE Transactions on Software Engineering*, 1989.
5. Kosko, B., "Bidirectional Associative Memories," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 18, no. 1, 1988, pp.49-60.
6. Sohn, A., and Gaudiot, J.-L., "Multilayer of Ring-Structured Network for Production Systems Processing" in *Proc. IEEE Int'l Workshop on Tools for AI*, Oct., 1989.
7. Touretzky, D.S. and Hinton, G.E., "Symbols Among the Neurons: Details of a Connectionist Inference Architecture," in *Proc. Int'l Joint Conf. on AI*, Aug.,1985, pp.238-243.

Recognition of 26 character-alphabet using a dynamic opto-electronic neural network

Shuichi TAI, Masaya OITA,
Masanobu TAKAHASHI, Keisuke KOJIMA, and Kazuo KYUMA

Central Research Laboratory, Mitsubishi Electric Corporation
8-1-1, Tsukaguchi-Honmachi,Amagasaki, Hyogo 661,JAPAN

ABSTRACT

We demonstrate an opto-electronic neural network that can recognize the 26 characters of the alphabet for the first time. This success is owing to the adoption of the proposed quantized learning rule suitable for optical implementation and the development of the dynamic opto-electronic neural network.

1. Introduction

Recently, there has been a strong requirement for the high-speed, low-cost and minituarized neural networks in order to use them in the practical fields. Among several approaches including Si-LSI neuro-chips, the opto-electronic approach is expected to play an important role in the implementation of neural networks because it allows the use of the innate parallelism and dense interconnection capabilities of optics in conjunction with the nonlinear processing ability of electronics.[1)]

Up to date,[2)-4)] many opto-electronic neural networks have been reported. Psaltis and Farhat pioneered this field by demonstrating the Hopfield associative memory using discrete opto-electronic components.[2)] We have reported the optical associative neuro-chip in which light emitting diodes, synaptic interconnection matrix and photodiodes are integrated in a layered structure on GaAs substrate.[3)]

One of the important features of neural network is that it automatically acquires knowledge through learning. However, most of the reported opto-electronic neural networks are, with a few exceptions,[5)] based on the Hopfield models without learning mechanism. This is due to the lack of a suitable analogue spatial light modulator (SLM) as a synaptic connection device.

In the first part of this paper, we propose a quantized learning rule which is basically the back-propagation learning rule but modified for utilizing the binary-operating SLM. In the second part, we demonstrate the recognition of 26 characters of the alphabet using the proposed learning rule and the dynamic opto-electronic neural network.

2. Quantized learning rule

The back-propagation learning rule is an error-correction-type learning rule using a set of supervised signals and training signals.[6)] Although the usefulness of this learning rule is verified by computer simulations, the weights of the synaptic connection must be continuously varied in the

learning process. This requirement makes it difficult to use the reliable SLMs such as magneto-optic SLMs and liquid crystal SLMs that operate in a binary digital value. To solve this problem, we propose the quantized learning rule. The procedure of the learning rule is summarized as follows.

(1)Start from random-distributed continuous weight W_{ij}. W_{ij} is a connection strength between ith and jth neurons.
(2)Convert the continuous weight W_{ij} into three discrete levels (-W,0,W). We call this discrete level W_{ij}^{q} quantized level.
(3)Choose one training signal and the corresponding supervised signal that the network is required to learn, and present them to the network. These signals are unipolar binary vectors.
(4)Calculate the correction signal ΔW_{ij} according to the conventional back-propagation rule.
(5)Correct the original continuous signal W_{ij} by adding ΔW_{ij}.
(6)Repeat steps (2)-(5) for all pairs of training and supervised signals.
(7)Repeat steps (2)-(6) until the connection strength pattern of the network is converged.
(8)Convert the final continuous weight into the quantized level if the connection strength pattern is converged.

The computer simulation results for the recognition of 26 characters of the alphabet are shown in Fig.1, by plotting the recognition rate as a function of the Hamming distance. The simulation was done using the three layered network where the number of neurons of the input, hidden and output layers are 30, 32 and 26, respectively. 26 unipolar binary vectors with 30 bit length were rearranged two-dimensionally as characters "a" to "z". The learning was performed so that one of the 26 output neurons corresponding to the input character was excited. In the simulation, the connection strength pattern was converged by 13000 times of learning (500 times for each character). As shown in Fig.1, there is little difference between the conventional continuous back-propagation learning and the proposed quantized learning. Therefore, it is confirmed that the quantized learning rule is very useful for opto-electronic neural networks using the binary-operating SLMs.

3. Experimental set-up

A schematic diagram of the dynamic opto-electronic neural network developed for the alphabet recognition is shown in Fig.2. In this system, two optical multipliers corresponding to the positive and negative synaptic connections were used to calculate the product of the unipolar binary vector with the bipolar synaptic matrix. In Fig.2, only one optical multiplier is shown to simplify. Each multiplier is constructed of an array of 32 light emitting diodes (LEDs), a binary-operating liquid crystal SLM with 32x32 pixels, an array of 32 photodiodes (PDs). The time-division multiplexing technique[7] was used to implement the three layered network.

The operation procedure is described as follows. At first,

the input vector $\mathbf{v}^{(1)}$ and the connection matrix W(I,H) between the input and hidden layers are addressed to the LED array and the SLM, respectively. The output vector $\mathbf{u}^{(2)}$ from the PD array is thresholded by the comparators to obtain the neuron state vector of the hidden layer $\mathbf{v}^{(2)}$. Next, $\mathbf{v}^{(2)}$ and the connection matrix W(H,O) between the hidden and output layers are addressed to the LED array and the SLM. The output vector $\mathbf{u}^{(3)}$ from the PD array is transmitted to a maximum-value selector. The output neuron which takes the maximum-value among 26 neurons corresponds to the desired answer to the input incomplete character.

4. Experimental results

For all training set of 26 characters, perfect recognition was successfully achieved by using the dynamic network described above. In Fig.1, the experimental recognition rate is also shown as a function of the Hamming distance d. The recognition rate was averaged over the randomly-selected signals more than 100 patterns for each Hamming distance. In response to the almost all incomplete input patterns of d=2, the correct character could be retrieved. These results agree with the computer simulations as shown in the figure.

The processing time of the constructed network was a few seconds and it was limitted by the slow response time of the SLM utilized in this work. However, it is not essential at present stage. The processing time can be dramatically improved by separating the optical vector-matrix multipliers between the input and hidden layers, and the hidden and output layers.

5. Conclusion

We proposed the quantized learning rule that allows the use of the binary-operating SLM. Furthermore, we demonstrated the recognition of alphabet 26 characters using actual dynamic opto-electronic neural network for the first time. The experimental results agreed with the computer simulation results. The optical neural networks reported in this work is quite suitable for integrating the system in a chip. The possibility of such optical neuro-chip will be also discussed at the conference.

References

1)For example, Appl. Opt., **20**, No.23 (1987)
2)N. H. Farhat, et al.:Appl. Opt.,**24**, 1496 (1985)
3)J. Ohta, et al.:IEEE IJCNN II-477 (1989)
4)K.Kyuma, et al.:SPIE **963**, 475 (1988)
5)N. H. Farhat, et al.:IEEE IJCNN II-365 (1988)
6)D. E. Rumelhart, et al:"Parallel Distributed Processing", Vol.I, (1986)
7)M. Oita, et al:IOOC '89, 19B3-17 (1989)

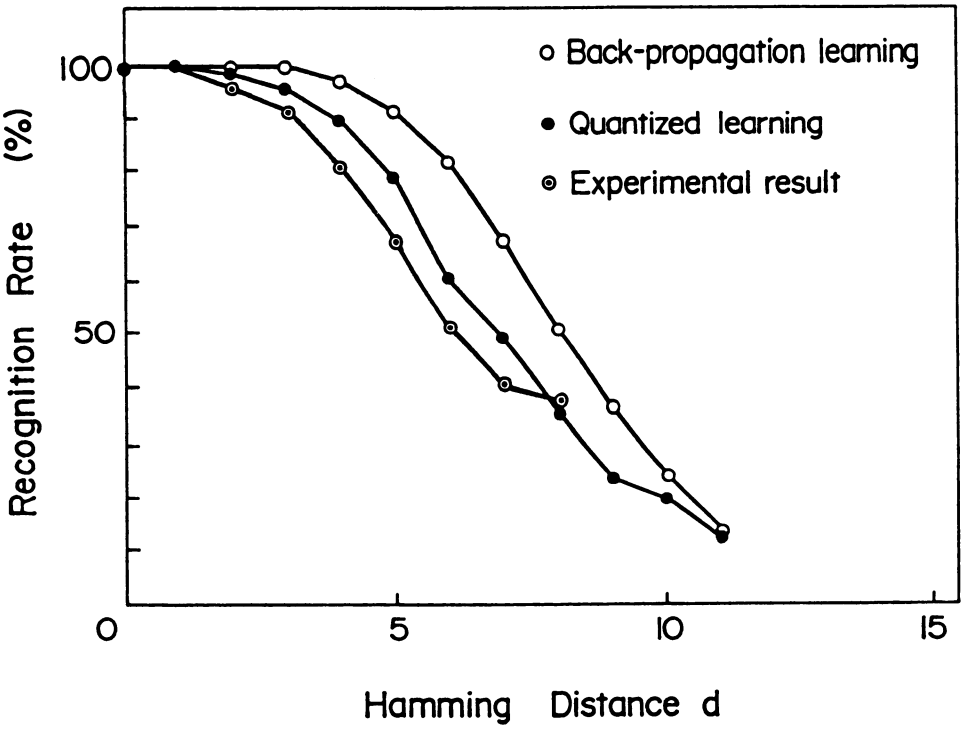


Fig.1 Computer simulation and experimental results of recognition rate for the 26 characters of alphabet

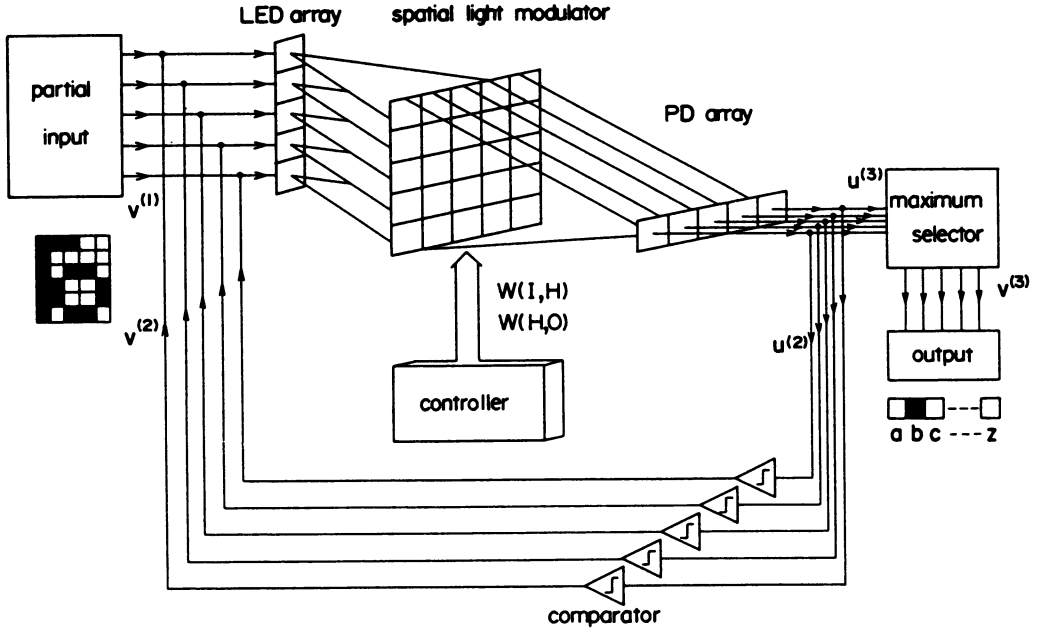


Fig.2 Schematic diagram of dynamic opto-electronic neural network developed for the alphabet recognition

Robotics, Speech, Signal Processing, and Vision

FLETE: An Opponent Neuromuscular Design for Factorization of Length and Tension

Daniel Bullock† and Stephen Grossberg‡
Cognitive & Neural Systems Program, Boston University
Boston, MA 02215, USA

How does the nervous system ensure independent control, or factorization, of the length and tension of muscles controlling a moving limb? We report simulations that show how an opponently organized spino-muscular system (Figure 1) may use co-contraction to vary limb compliance over a large range without causing joint rotations by inadvertently changing the lengths of opponent muscles. Consider the forces, F_i, $i = 1, 2$ developed by two muscles operating on different sides of a joint. In a springy tissue like muscle, force depends on the amount of stretch beyond the resting length. Because muscle can actively contract, muscle has a *variable* threshold length for force development. Thus we have

$$F_i = g\big([L_i - \Gamma_i + C_i]^+\big) \tag{1}$$

where L_i is muscle length, Γ_i is the resting muscle length, C_i is degree of contraction, and $g(w)$ is monotone increasing. Notation $[w]^+$ means $\max(w, 0)$. Because contracted fibers *yield* when the force acting to stretch them is sufficiently large, a simple law for C_i is

$$\frac{d}{dt}C_i = \beta_i\big[(B_i - C_i)M_i - \delta C_i\big] - [F_i - \Gamma_F]^+ \tag{2}$$

where $0 < \beta_i < 1$ and M_i is the output signal of the i^{th} alpha-motoneuron pool. When force F_i exceeds threshold Γ_F, it reduces contraction. By constraint β_i, contraction caused by neural input M_i is slow relative to decontraction by external forces.

At equilibrium, $\frac{d}{dt}C_i = 0$ in (2), so the equilibrium value of C_i is

$$C_i = \frac{M_i B_i - \dfrac{[F_i - \Gamma_F]^+}{\beta_i}}{M_i + \delta}\ . \tag{3}$$

Given (3), how is it possible to generate and sustain forces much larger than Γ_F at a fixed muscle length? By (1), greater force at a fixed length L_i can be generated only by increasing C_i. However, if β_i is constant and less than 1, then (3) shows that the negative force feedback will cancel the effects of increasing M_i, and C_i will not grow large. To overcome this deficiency, let the contraction rate parameter β_i and the number of sites B_i increase with M_i. Such a relation is called the *size principle* [3]: As total excitatory input to the alpha motoneuron population grows, it recruits additional, progressively larger motoneurons which have faster conducting axons, whose collaterals reach many more motor fibers and whose potentials evoke more rapid muscle contractions. Equation (3) provides a new, functional, perspective on the size principle.

However, the size principle can pose a threat to stable position coding. If a limb segment is initially at equilibrium, such that $F_1 = F_2$, then by (1),

$$g\big([L_1 - \Gamma_1 + C_1(A_1)]^+\big) = g\big([L_2 - \Gamma_2 + C_2(A_2)]^+\big), \tag{4}$$

† Supported in part by the National Science Foundation (NSF IRI-87-16960).

‡ Supported in part by the National Science Foundation (NSF IRI-87-16960) and the Air Force Office of Scientific Research (AFOSR F49620-87-C-0018).

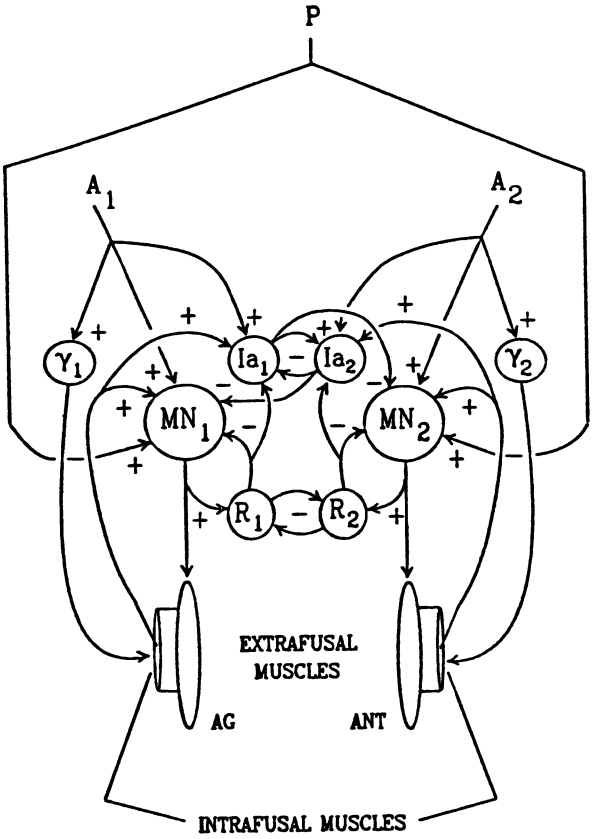


Figure 1. FLETE model components: Neuron populations comprising two channels control opponent muscles (AG for agonist, ANT for antagonist) acting on a joint. Descending signal P to both channels allows co-contraction and joint stiffening. Adjusting the balance between descending signals A_1 and A_2 allows reciprocal contractions and joint repositioning. For clarity, subpopulations of neurons and some signal pathways are not depicted. Key: $Ia_i = Ia$ interneuron population in channel $i, i = 1, 2$; $\gamma_i =$ gamma motoneurons; $MN_i =$ alpha motoneurons; $R_i =$ Renshaw cells; $+ =$ excitatory input; $- =$ inhibitory input.

where $C_1(A_1)$ denotes the equilibrium value of C_1 when $M_1 = f(A_1)$ in (2). Now try to hold the limb at the same position, but more rigidly, by increasing the level of muscle contraction on both sides of the joint. To do this, add constant P to each motoneuron input [4]. Thus $M_1 = f(A_1 + P)$ and $M_2 = f(A_2 + P)$. However, by the size principle, (4) implies

$$g\big([L_1 - \Gamma_1 + C_1(A_1 + P)]^+\big) = g\big([L_2 - \Gamma_2 + C_2(A_2 + P)]^+\big) \tag{5}$$

for arbitrary P and the same initial values of L_i *only if* $A_1 = A_2$ (Figure 2). Thus a co-contractive input P aimed at stabilizing limb position could instead cause a limb rotation. This is a failure to factorize length and tension.

Renshaw cells are well situated to play a compensatory role: Opponent Renshaw populations R_1 and R_2 measure the output of their respective alpha-motoneuron populations, α-MN_1, and α-MN_2, and compare those outputs via mutually inhibitory signals (Figure 1). A consensus emerges regarding which MN channel to inhibit via Renshaw feedback, and which to disinhibit via feedback along the Ia interneuron (IaIN) pathway. Suppose that a co-contractive input, P, to α-MN_1 and α-MN_2 occurs when input A_1 exceeds A_2 and that the activity of α-MN_1 is consequently multiplied by a larger factor than that of α-MN_2 due to the size principle (Figure 2). Then R_1 also becomes much more active due

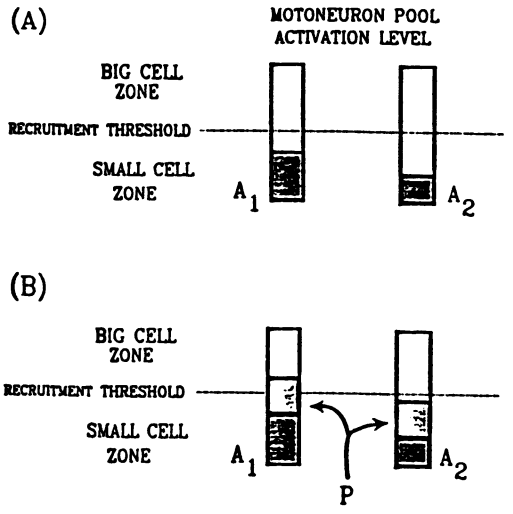


Figure 2. When opponent motoneuron populations obey the size principle, a co-contractive signal P sent in parallel to both populations can disrupt the joint position code. (A) Signals A_1 and A_2 supraliminally activate only small cells in opposing channels and their relative sizes determine the balance of muscular forces and thus the equilibrium joint position. (B) With $A_1 > A_2$, co-contractive signal P causes the total input $A_1 + P$ to exceed the big cell threshold while input $A_2 + P$ remains below the big cell threshold. Thus part of the signal P is subjected to greater amplification in channel 1 than in channel 2. Unless compensated, this would create a new balance of forces and cause an unwanted joint rotation.

to a size-correlated synaptic weighting on α-MN_1 axon collaterals to R_1 [2],[5]. Because the opposing R_2 has not experienced as large an input increment, R_1 will transiently become more active than R_2 by an amount that scales with the *difference* between the α-MN output increments due to the change in P. Thus this system calculates an *error* due to unequal amplifications of co-contractive inputs. This error signal then directly inhibits α-MN_1 and, by inhibiting $IaIN_1$, indirectly activates α-MN_2. Both actions work to zero the error without negating either the shared increment in α-MN_i activation required to increase joint stiffness, or the joint angle setting determined by the difference in descending inputs, exclusive of P, to opponent α-MN and IaIN populations.

This conjecture has been supported by our computer simulations, which assumed a rotary joint affected by two opponent muscles, each of which is inserted in the moving segment one unit from the axis of rotation. The distance from muscle origin to the axis of rotation was 20 units, and the midpoint of the limb's 180° excursion was stipulated to be at joint angle $\Theta = 0°$. Origin-to-insertion muscle lengths, L_i, were thus functions of Θ:

$$L_1 = \sqrt{(\cos\Theta)^2 + (20 - \sin\Theta)^2}, \quad L_2 = \sqrt{(\cos\Theta)^2 + (20 + \sin\Theta)^2} \tag{6}$$

Because these simulations concerned only large-scale effects on equilibrium joint angle, we ignored moment-arm and force-velocity effects and chose the simple force law

$$F_i = k[L_i - \Gamma_i + C_i]^+ \tag{7}$$

where $k = .5, \Gamma_i = 20.9$ and $i = 1, 2$. Limb dynamics were governed by equation

$$\frac{d^2}{dt^2}\Theta = \frac{1}{m}(F_1 - F_2 - n\frac{d\Theta}{dt}) \tag{8}$$

where m represents mass and n is a damping coefficient.

Contractile state C_i was governed by (3). Variables β_i and B_i were defined by:

$$\beta_i = .05 + .02(A_i + P), \quad B_i = 2 + 20(A_i + P) \tag{9}$$

Both variables grow as a function of total descending input $A_i + P$ to the MN pools in channel i, but β_i grows with a smaller slope. Use of β_i and B_i in Equations (2) and (9) approximates α-MN recruitment effects that occur *in vivo*.

Recruitment of larger motoneurons causes larger inputs to the Renshaw cells. In our lumped model, this effect was absorbed into a single variable, z_i, which was a function of recruitment extent, approximated by $A_i + P$. The equations for opponent Renshaw populations were thus

$$\frac{d}{dt}R_i = (\lambda B_i - R_i)z_i M_i - R_i(1 + R_j) \tag{10}$$

$$z_i = .2 + .8(A_i + P) \tag{11}$$

where $\{i, j\} = \{1, 2\}$ and $\lambda = 5$.

We modeled the opponent alpha-motoneuron populations via

$$\frac{d}{dt}M_i = \phi\Big[(\lambda B_i - M_i)(A_i + P + \chi E_i)\Big] - M_i(1 + \Omega R_i + I_j) \tag{12}$$

where $\{i, j\} = \{1, 2\}, \phi = .2$, and $\Omega = 0$ or 1. Inhibitory inputs I_j come from the IaINs (Figure 1) and excitatory inputs E_i from the muscle spindles. IaIN dynamics were modeled without direct dependence on B_i, and without a co-activating input P:

$$\frac{d}{dt}I_i = \phi(10 - I_i)(A_i + \chi E_i) - I_i(1 + \Omega R_i + I_j) \tag{13}$$

Results. In our simulations, variables A_1, A_2, and P were constant inputs and variables L_i, F_i, and Θ were dependent variables. Composite spindle feedback signals E_1 and E_2 in Equations (12) and (13) were gated off in our simulations by setting $\chi = 0$. This allowed us to test the ability of the Renshaw-Ia-MN feedback circuit to achieve position code invariance without assistance from stretch reflexes.

When Renshaw feedback was absent ($\Omega = 0$), changing P while A_1 and A_2 remained fixed led to large rotations. When Renshaw feedback was present ($\Omega = 1$), rotations due to changing P with fixed A_1 and A_2 were $< 1°$. This was the invariance property we sought. A more complete discussion of relevant physiology, of our modeling results, and their relation to our model of variable-speed trajectory formation will appear in [1].

REFERENCES

[1] Bullock, D. and Grossberg, S. (in press). VITE and FLETE: Neural modules for trajectory formation and postural control. In W.A. Hershberger (Ed.), **Volitional action**. Amsterdam: North-Holland/Elsevier.

[2] Cullheim, S. and Kellerth, J.O. (1978). A morphological study of the axons and recurrent axon collaterals of cat α-motoneurones supplying different functional types of muscle unit. *Journal of Physiology* (London), **281**, 301–313.

[3] Henneman, E. (1985). The size-principle: A deterministic output emerges from a set of probabilistic connections. *Journal of Experimental Biology*, **115**, 105–112.

[4] Humphrey, D.R. and Reed, D.J. (1983). Separate cortical systems for control of joint movement and joint stiffness: Reciprocal activation and coactivation of antagonist muscles. In J.E. Desmedt (Ed.), **Motor control mechanisms in health and disease**. New York: Raven Press, 347–372.

[5] Pompeiano, O. (1984). Recurrent inhibition. In R.A. Davidoff (Ed.), **Handbook of the spinal cord, Vols. 2 and 3. Anatomy and physiology**. New York: Marcel Dekker.

A Self-Regulating Generator of Sample-and-Hold Random Training Vectors

Paolo Gaudiano and Stephen Grossberg
Boston University
Center for Adaptive Systems
111 Cummington street
Boston, MA 02215

The Role of Circular Reactions in Self-Organization

How does a child learn to produce the sounds that he hears? How does he learn to reach for objects that he sees? Piaget[1] has provided a useful insight with his concept of a *circular reaction*. When an infant makes internally generated movements of his hand, the eyes automatically follow this motion. As he fixates the hand at a variety of positions, a transformation is learned between the eye-head system and the hand-arm system. As learning progresses, the reverse transformation is also learned, eventually enabling the child to touch what he sees. Thus the circular reaction is based on endogenously generated actions whose commands are correlated with sensory feedback by means of an associative transformation.

This article describes a self-regulating endogenous generator of random vectors that can be used to learn correct sensory-motor or motor-motor transformations during a circular reaction. The generator is biphasic: The generation of each vector is followed by a complementary quiescent phase to allow learning, after which a new vector is generated. Such a sample-and-hold mechanism allows the adaptive sensory-motor system to learn the correct transformations in a self-regulated periodic environment. Here randomness provides the substrate for adaptive control of deterministic motor behaviour.

A Self-Regulating Generator of Random Trainable Vectors

The Vector Integration To Endpoint (VITE) model for generation of synchronous multi-joint trajectories has been described by Bullock and Grossberg[2]. A self-organizing adaptive VITE model that incorporates the random vector generator is described in Gaudiano and Grossberg[3].

The following equations describe one generator channel, which is depicted in Figure 1:

Random Input to the Generator

$$I_i^- = I_i^+ = I(\text{constant}); \qquad J_i^+ = \begin{cases} J \in [\mu_J - \sigma_J, \mu_J + \sigma_J] & \text{with probability } p_J \\ \mu_J & \text{with probability } (1 - p_J) \end{cases} \tag{1}$$

Where μ_J is the average input level and σ_J is the maximum possible deviation from μ_J.

Pauser Gate

$$\gamma_P = \begin{cases} 1 & \text{if } \sum_i Y_i^- > \Gamma_P \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where Γ_P is a fixed threshold, and Y_i^- is described below (eq. 5).

On and Off Channel Activations

$$\frac{dX_i^+}{dt} = -AX_i^+ + (B - X_i^+)\,(I + J_i^+\gamma_P); \quad \frac{dX_i^-}{dt} = -AX_i^- + (B - X_i^-)\,I \tag{3}$$

Habituating Transmitter Gates

$$\frac{dZ_i^+}{dt} = C\,(D - Z_i^+) - Ef(X_i^+)Z_i^+; \quad \frac{dZ_i^-}{dt} = C\,(D - Z_i^-) - Ef(X_i^-)Z_i^- \tag{4}$$

where $f(X) = \alpha X + \beta X^2$ represents a nonlinear habituation law.

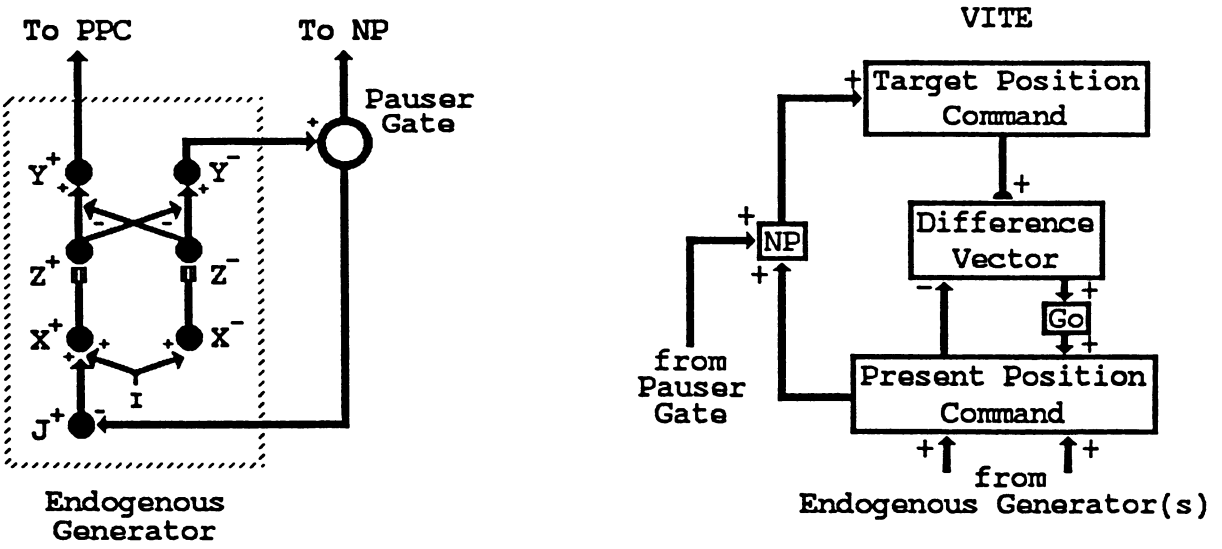


Figure 1: The endogenous generator circuit (left diagram). The On channels (+ superscript) provide the random unbiased test vectors, while the Off channels (-) regulate the pauser gate and the phasic input J^+. The right diagram shows the basic layout of the VITE model and its inputs from the generator.

Opponent Output Signals

$$Y_i^+(t) = [X_i^+ Z_i^+ - X_i^- Z_i^-]^+; \quad Y_i^-(t) = [X_i^- Z_i^- - X_i^+ Z_i^+]^+ \tag{5}$$

where $[W]^+ = max(W, 0)$.

The above equations describe a gated dipole (Grossberg, [4,5,6]) with nonlinear transmitter habituation (the "+" and "-" superscripts refer to the "On" and "Off" channels, respectively, as shown in Figure 1). With this formulation, the net signal through the gate exhibits a transient response for sufficiently large input channel activations. As a result, when the On channel receives random inputs J^+ it produces a transient response. As the transmitter Z^+ habituates to lower levels, the On channel activation decays, disinhibiting the Off channel. Once enough Off channels are active the pauser gate γ_P exceeds the threshold Γ_P, becoming active. This shuts off the random input J^+, leading to a vigorous rebound in the Off channel due to the habituated On transmitter gates. When the transmitter in the On channel is replenished, the pauser gate is inactivated. This restores random input to the On channel, and a new cycle begins.

Fig. 2 illustrates computer simulations of the various levels of a one-channel generator. The bottom row shows a trace of the phasic input (dashed line) and the X_i activations. The middle row represents the available transmitter levels[1] Z_i, and the top row depicts the resulting Y_i^+ (left) and Y_i^- outputs. Note the complementarity of the On and Off outputs, and the quasi-periodic nature of the alternating phases.

Adaptive Tuning of PPC and TPC Coordinates

The VITE model generates synchronous multi-joint trajectory commands [2] by integrating the Difference Vector (DV) between the limb's Present Position Command (PPC) and its Target Position Command (TPC). The present work shows how signals from TPC to DV are adaptively calibrated during the circular reaction to be in the same measurement scale as signals from PPC to DV.

The generator output signals are integrated at the PPC. This causes a limb movement which ends when the pauser gate turns on. Then a copy of the PPC is instated at the TPC through the Now Print (NP) gate, and learning in the TPC→DV pathways drives the DV toward zero until TPC→DV signals and PPC→DV signals are in the same measurement scale. PPC integration of endogenously generated signals obeys the equation:

[1] It should be noted that Z_i^- remains constant throughout the simulation because it receives no phasic input.

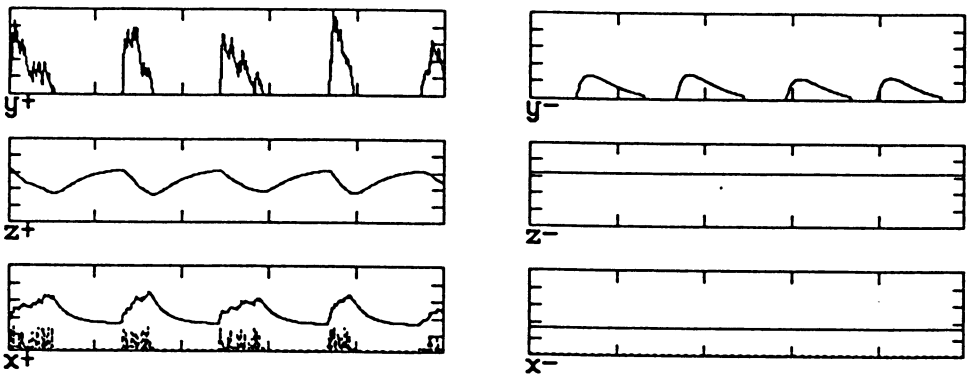


Figure 2: The behaviour of the various layers in the generator. Parameters for this simulation: A=0.1, B=1.0 C=0.1, D=10.0, E=0.5, I=0.05, μ_J=0.05, σ_J=1.0, p_J=0.5, Γ_P=0.08, $\alpha = 0$, $\beta = 1$. The Z values are plotted between 0.0 and 10.0, while all the other values are in the range [0,1].

Present Position Command[2]

$$\frac{dP_i^+}{dt} = \left(1 - P_i^+\right) Y_{2i-1}^+ - P_i^+ Y_{2i}^+; \quad \frac{dP_i^-}{dt} = \left(1 - P_i^-\right) Y_{2i}^+ - P_i^- Y_{2i-1}^+; \tag{6}$$

Each VITE channel consists of an agonist-antagonist pair, thus requiring input from two generator On channels. This is reflected in the notation Y_{2i-1}^+ and Y_{2i}^+, which shows the existence of two generator channels for each VITE channel.

System 1-6 is capable of generating an unbiased distribution of PPC vectors, as illustrated in Figure 3a. The abscissa represents the difference in amplitude between the agonist and antagonist side of the first PPC, and the ordinate that of the other PPC. Each 2-D vector was calculated at the moment when the PPC is copied into the TPC, where it remains unchanged until the next On phase. All four generator modules were coupled to a single pauser gate, which guarantees synchronous On and Off phases across generator modules.

Figures 3b and 3c show histograms of the number of vectors within each of sixteen evenly-spaced angles $(-\pi, \pi]$ and magnitudes $[0, 1]$ with respect to the origin. The curves confirm the unbiased distribution of the random vectors. Note that the distribution of output vector magnitudes can be modulated by generator parameter choices without changing the random input J^+ characteristics.

Concluding Remarks

We have presented a model that generates unbiased random vectors interleaved with quiescent phases that allow for complementary tasks–such as learning–which require periodically stationary inputs. The random vector generator provides the VITE model with a self-regulating mechanism for learning circular reactions, such that input stationarity occurs when the PPC and the TPC encode the same vector. The model is robust across a wide range of parameters, and allows modulation of the distribution and temporal characteristics of its output vectors.

The behaviour of the VITE model when the generator is active is functionally similar to the babbling phase observed in an infant's speech acquisition, during which the infant produces a series of endogenously generated sounds, which are used in learning the transformation between speech perception and production. An analogous babbling phase may be used to control self-tuning of parameters used in trajectory formation for execution of planned motor tasks.

[2]Terms not related to the generator have been omitted for clarity. See[3] for additional details.

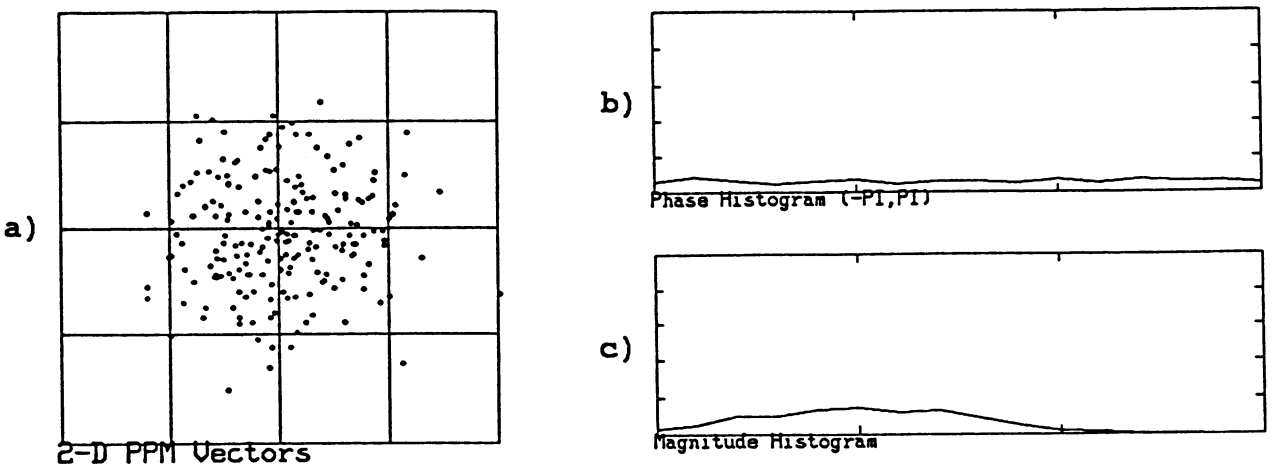


Figure 3: Random PPC vectors generated by four generator modules: (a) Each dot represents the pairwise difference of the four Y_i^+ responses; (b) Vector distribution at each of sixteen evenly-spaced angles between $-\pi$ and π; (c) Vector distribution at each of sixteen evenly-spaced magnitudes in the interval $[0, 1]$.

References

[1] Piaget, J. (1963). *The origins of intelligence in children.* New York: Norton.

[2] D. Bullock and S. Grossberg (1988) "Neural Dynamics of Planned Arm Movements: Emergent Invariants and Speed-Accuracy Properties During Trajectory Formation." *Psychological Review.* 95(1), 49-90.

[3] P. Gaudiano and S. Grossberg (1990) "A Self-Organizing Neural Circuit for Control of Planned Movement Trajectories." (In Preparation).

[4] Grossberg, S. (1972) "A Neural Theory of Punishment and Avoidance, II. Quantitative Theory." *Mathematical Biosciences* 15, 39-67.

[5] Grossberg, S. (1982) *Studies of mind and brain: Neural principles of learning, perception, development, cognition, and motor control.* Boston: Reidel Press.

[6] Grossberg, S. (1984) "Some Normal and Abnormal Behavioral Syndromes Due to Transmitter Gating of Opponent Processes." *Biological Psychiatry,* 19(7), 1075-1118.

MANIPULATOR CONTROL USING LAYERED NEURAL NETWORK MODEL WITH SELF-ORGANIZING MECHANISM

Shinya HOSOGI
International Institute for Advanced Study of Social Information Science
FUJITSU LIMITED
1-17-25 Shinkamata, Ohta-ku, Tokyo 144, Japan

ABSTRACT: A layered neural network model is proposed for controlling robotic manipulators. The model has self-organizing mechanism in the second layer, where the function of cerebellar Golgi cell system is attached. Connections are formed automatically by modifying both excitatory and inhibitory weights according to Hebb-type learning rule. It is shown that cells in the second layer self-organize into representatives of similar inputs. The model has been applied to learn to control a manipulator with dynamic characteristics. The ability of generalization and adaptation to failures of cells (redundancy) is demonstrated by computer experiment.

INTRODUCTION

It is well known that neural networks in cerebellar cortex play important roles in executing movement smoothly and accurately based on a great amount of information stored in modifiable synapses [1,2]. Hence, the cerebellum can be regarded as a sophisticated controller which has great ability of learning and of adapting itself to external circumstances.

In the cerebellar neural networks, the layer of granule cell is equipped with the Golgi cell. This layer corresponds to the second layer of the perceptron model of the cerebellum proposed by Marr [3] and Albus [4]. According to their theory, the role of the Golgi cell is considered to keep firing rate of parallel fibers constant by monitoring activity of both mossy fibers and parallel fibers. It is also shown that the second layer is essential to the ability of pattern separation of inputs. By taking the Golgi cell into account, Fujita [5] proposed the adaptive filter model of the cerebellum. But these models do not have modifiability in the related synapses.

For an artificial multi-layered network model, architecture of intermediate layer is substantial not only to pattern separability but also to ability of generalization and redundancy. For an extreme case, if each of input patterns is separated completely and is represented by a single cell, the ability of generalization and redundancy is not expected. Learning must be done for all inputs in this case. In order to obtain the response properties adequate for the above ability, we assume that the weights in the second layer are modifiable in our perceptron-based model.

In this paper, we propose the artificial network model by taking into account the above findings on the cerebellar neural networks. The model is constructed based on the theory of self-organization [6,7]. Preliminary results have been given in [8]. The response properties of cells in the second layer and the ability of generalization and redundancy are studied for the learning control of a robotic manipulator.

MODEL

The layered neural network model and the learning procedures in the second and third layers are described in this section. The neural network model with one Golgi cell is shown schematically in Fig. 1. Mutual inhibitory interaction is not considered, since the cerebellum has no such connection in the second layer.

The first layer of the network receives outputs from filters, through which continuous valued inputs (e.g. angle and velocity of each joint of a manipulator) are transformed into a set of spatial patterns. Each filter has Gaussian response property. The width of the response is taken rather large and they are arranged to overlap each other. The filters and the cells in the first layer are of the same number (N_1), and are connected one to one by fixed weights $W_i^{(1)}$.

Cells in the second layer receive inputs $V_i^{(1)}$ through excitatory synaptic weights $W_{ij}^{(2)}$ from the first layer and an input V_g through inhibitory synaptic weights W_i^g from one Golgi cell. We assume only one Golgi cell in the network, since one Golgi cell inhibits a large number of granule cells in the cerebellum. At the start of learning, the cells in the first and second layer (the number of cells is N_2) are randomly interconnected by $W_{ij}^{(2)}$.

A cell in the first and second layers has a sigmoid transfer function. A cell in the third layer (the number of cells N_3 corresponds to the degree of freedom of a manipulator) is assumed to have linear characteristics, since outputs of the network are considered to be unlimited. The *i*- th cell in the third layer and the *j*- th cell in the second layer are connected by a modifiable synaptic weight $W_{ij}^{(3)}$.

The equations of the cells in the second and third layers are written as

$$V_i^{(2)} = \phi(\Sigma_j W_{ij}^{(2)} V_j^{(1)} - W_i^g V_g - \theta^{(2)}), \quad (1)$$

$$V_i^{(3)} = \Sigma_j W_{ij}^{(3)} V_j^{(2)}, \quad (2)$$

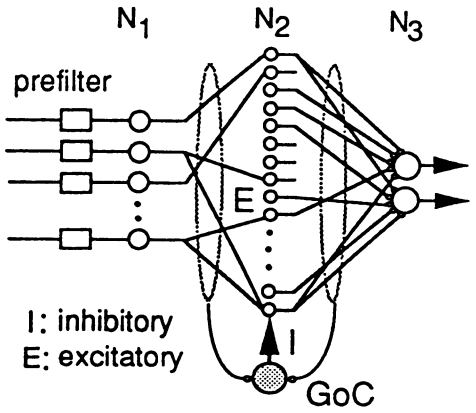


Fig. 1 Neural network model with Golgi cell.

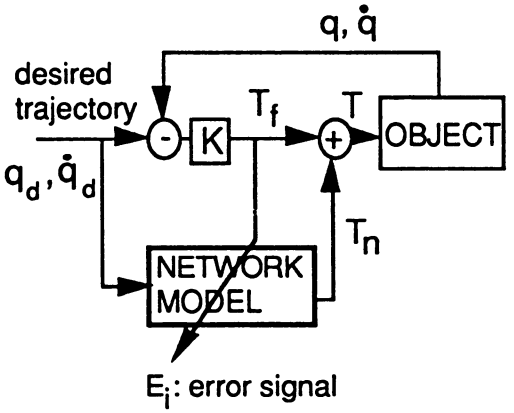


Fig. 2 Control system diagram.

where ϕ is a transfer function and $\theta^{(2)}$ represents a threshold.

Since the cerebellar Golgi cell receives outputs from both the first and the second layers, and inhibits the cells in the second layer through an inhibitory weight W_i^g, the output of the Golgi cell may be written as

$$V_g = \varphi(\Sigma_i W_{gi}^m V_i^{(1)} + \Sigma_j W_{gj}^p V_j^{(2)} - \theta_g), \quad (3)$$

where φ is a transfer function of Golgi cell, W_{gi}^m and W_{gj}^p are constant synaptic weights coming from mossy and parallel fibers to the Golgi cell respectively, and θ_g is a threshold.

The perceptron model has modifiable weights $W_{ij}^{(3)}$ only in the third layer. In the present model, it is assumed that the weights $W_{ij}^{(2)}$ and W_i^g are also modifiable. Therefore, the model has two learning procedures. Learning in the second layer is accomplished by the Hebb-type rule and these weights are modified according to

$$\tau_1 dW_{ij}^{(2)} / dt = (-W_{ij}^{(2)} + c_1 V_i^{(2)} V_j^{(1)}) V_i^{(2)}, \quad (4)$$

$$\tau_2 dW_i^g / dt = (-W_i^g + c_2 V_i^{(2)} V_g) V_i^{(2)}. \quad (5)$$

Here, τ_1 and τ_2 are time constants of learning, and c_1 and c_2 are parameters related to convergence and velocity of learning. Self-organizing properties of the above type of equations have been studied in Amari and Takeuchi [6], where the theory has been applied to the formation of category detecting cells and the analysis is performed under the conditions of $V_g = 1$ and $V_i^{(2)} = 1$ or 0. In the present case, V_g depends on the firing rate of input and output, by which response properties of the cells in the second layer have been improved. The term $V_i^{(2)}$ in the right-hand-side serves for the cells to fire when new patterns are given. The time constants and the parameters for learning are selected so as to fulfill the ability of pattern separation, generalization and redundancy. From Eqs. (1) and (3), it is found that the activity of the second layer cell can be regulated by the Golgi cell.

The learning in the third layer is accomplished by the usual error-correction procedure. The weights are modified by the following relation :

$$\Delta W_{ij}^{(3)} = \varepsilon E_i V_j^{(2)}, \quad (6)$$

where ε (<1) is a constant, and the error term E_i (i=1,..,N3) are given by feedback torques described below.

CONTROL SYSTEM

The present network model has been applied to the dynamic control of the robotic manipulator, which has two links operating in 2-dimensional vertical plane with equal length L=0.3(m) and mass M=5.0(kg). In general, a mechanical model of a manipulator is given by the non-linear differential equation:

$$M(q)\ddot{q} + C(q, \dot{q}) + G(q) = T, \quad (7)$$

where M(q) represents inertia terms, $C(q, \dot{q})$ Coriolis and centrifugal terms, G(q) gravitational terms, and T actuator torques. In the present considerations, joint angles and velocities of the first and second links are restricted to the regions of $-\pi/3 < q_1, q_2 < \pi/3$ and $-2.8\pi/s < \dot{q}_1, \dot{q}_2 < 2.8\pi/s$, respectively. If a desired trajectory is given in joint space coordinate, an inverse dynamic model of a controlled object can be obtained from the control system with two pathways as shown in Fig. 2. One path is the main path of feedback with a gain K and the other is the

side path with adaptive elements. Physiologically the lateral part of the cerebellum has the same side path structure for the major signal flow pathway [1]. In Eq. (6), the error-term E_i is given by the feedback torques:

$$T_{f,i} = K_p(q_i^d - q_i) + K_v(\dot{q}_i^d - \dot{q}_i), \qquad (i=1,..,N_3) \qquad (8)$$

where K_p and K_v are constant gains for angular positions and velocities, respectively. The manipulator is driven by the sum of two contributions: $T = T_f + T_n$. The path of feedback is necessary to sense unexpected changes of characteristics of the system or of external circumstances.

RESULTS

Four kinds of desired trajectories (T1-T4), each of which is made with a simple combination of trigonometrical functions and requires the period of 1 s, are used to see the response properties of the cells in the second layer and the performance in controlling the manipulator. For one trajectory 50 patterns are obtained (sampling time: 20 ms) by filtering the inputs.

Response property To see how the self-organizing process works and the connections to the cells in the second layer are formed, 200 patterns (correspond to four trajectories) are imposed on the network repeatedly. The parameter values used in Eqs. (4) and (5) are $\tau_1 = \tau_2 = 20$, $c_1 = 1.0$ and $c_2 = 30.0$, and the numbers of cells used in the simulation are $N_1 = 80$, $N_2 = 100$ and $N_3 = 2$. In calculating Eqs. (2) and (3), they are solved self-consistently, and the thresholds $\theta^{(2)}$ and θ_g are set to 0 for simplicity.

Initial values of weights, $W_{ij}^{(2)}$ and W_i^g, are chosen randomly between 0 and 1, which cause all the cells active at the start of learning. As the learning proceeds, the number of active cells decreases gradually and reaches an equilibrium state. One example of the response property for T1 is shown in Fig. 3, where the abscissa represents input patterns for one trajectory and the ordinate represents active cells in the second layer, which are rearranged in order to see the responsibility for a particular input pattern.

The figure shows some features adequate for the ability of generalization and redundancy. The cells in the second layer self-organize into representatives of similar inputs. An input pattern is represented by the plural number of active cells, the number of which is nearly constant over the sequential inputs.

Note that the cells in the second layer do not have mutual inhibitions. Even with this configuration, good quality of response properties has been obtained by taking the function of Golgi cell into account in the model.

Manipulator control The task of the network model is to learn to yield desired outputs (torques) so as to follow the track of desired trajectories expressed in joint space coordinates (angle and velocity). In the present case, the gains of the feedback are taken small ($K_p = 1.0$ and $K_v = 3.0$), so that the control of feedback alone is not enough to track the desired trajectories. At the start of learning, the weights $W_{ij}^{(3)}$ are taken zero for all the elements. The constant ε of learning in the third layer is 0.02.

In the early stages of learning, the contribution of the network is small and the object is controlled mainly with the contribution of the feedback. As the learning proceeds, the error due to the feedback decreases and the network contributes to most of the desired torques corresponding to the desired trajectory. After 100 trials the normalized root-mean-square-errors (RMSE) are reached within halves of the overlapping interval of the filter.

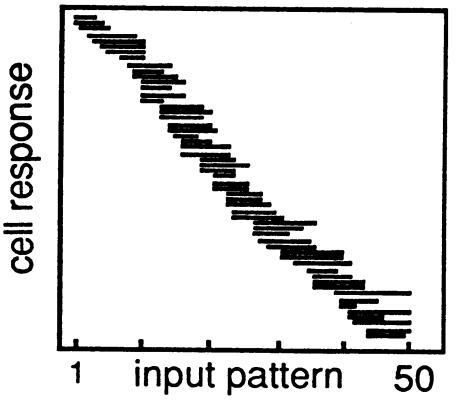


Fig. 3 Response property of cells in the second layer (T1).

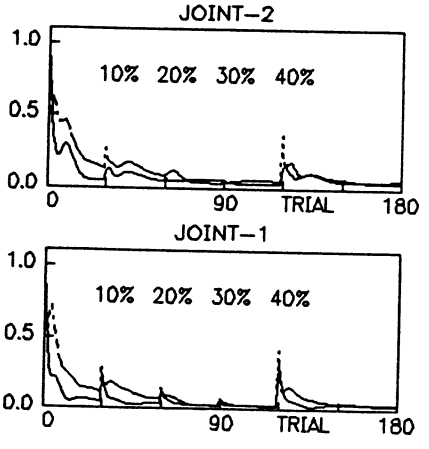


Fig. 4 Curves of normalized RMS errors vs. trials showing the ability of adaptation.

To see the adaptability (redundancy) to failures of the cells in the second layer, 10 % damages (outputs are forced to 0) are introduced additionally to the cells at every thirty trials. The results are shown in Fig. 4, where the upper and lower curves correspond to the normalized RMSE of angles and velocities, respectively. The learning of the second layer is not performed when the failures are introduced, since the purpose is to show that the connections formed by the self-organizing mechanism do have redundancy. The network shows adaptability up to about 30 % failures of the cells by the learning of the third layer.

To see the ability of generalization, learning of four trajectories has been performed by modifying $W_{ij}^{(3)}$ successively. Fig. 5 shows the curves of normalized RMSE of joint-angles versus trials (bold lines), where the RMSE of independent learning of each trajectory is superimposed on the same diagram as dotted lines. Each trajectory is learned repeatedly by 30 trials. The figure shows that the effect of generalization is not evident at the early stage of learnings. But, as the learning proceeds, the performance of T3 and T4 are improved (Fig. 5-a). The results after 600 trials are shown in Fig. 5-b, where the performance is markedly improved for all the trajectories.

Fig. 5 Curves of normalized RMSE of angle vs. trials showing generalization. Four trajectories (T1-T4) are learned successively.

CONCLUSION

We have proposed an artificial network model of the cerebellar neural networks and applied it to the control of a manipulator. The model is equipped with two learning mechanisms. Especially, the second layer of our model has self-organizing mechanism. The connections are formed automatically by input data. The ability of generalization and redundancy has been demonstrated. If there was such modifiability in the cerebellar neural networks, even at the early stage of growth, it would serve for the improvement of the ability. More complex trajectories than tested here can also be learned similarly by the method described above.

The author would like to thank K. Matsuo and K. Asakawa for their encouragement and K. Wada for his advice. This work was supported by NEDO under the management of FED.

REFERENCES

[1] M. Ito: "The Cerebellum and Neural Control," Raven Press, New York (1984).
[2] G. I. Allen and N. Tsukahara: "Cerebrocerebellar Communication Systems," Physiol.Rev.54 (1974) 957.
[3] D. Marr: "A Theory of Cerebellar Cortex", J. Physiol. 202 (1969) 437.
[4] J. S. Albus: "A Theory of Cerebellar Function", Math. Biosci. 10 (1971) 25.
[5] M. Fujita: "Adaptive Filter Model of the Cerebellum", Biol. Cybern. 45 (1982) 195.
[6] S. Amari and A. Takeuchi: "Mathematical Theory of Formation of Category Detecting Nerve Cells", Biol. Cybern. 29 (1978) 127.
[7] K. Fukushima: "Neocognitron : A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position", Biol. Cybern. 36 (1980) 193.
[8] S. Hosogi, K. Asakawa and K. Matsuo: "Cerebellar Neural Network Model for Manipulator Control", 2nd International Symposium on Bioelectronic and Molecular Electronic Devices, Dec. (1988) 105.

ONE-CLASS GENERALIZATION IN SECOND-ORDER BACKPROPAGATION NETWORKS FOR IMAGE CLASSIFICATION

Mary M. Moya
Larry D. Hostetler
Exploratory Systems Development Division 9133
Sandia National Laboratories
Albuquerque, NM 87185
E-mail: mmmoya@sandia.gov

Abstract: *In an earlier paper [5], we reported that it is possible to train a first-order multi-layer feedforward network with backpropagation to classify raw 8-bit images of vehicles. We concluded that a linear feedforward network is capable of within-class generalization when trained with perspective views taken every 10°, but is incapable of one-class generalization. This paper describes the results of a set of experiments to train a feedforward network with second-order inputs to perform one-class classification on image data. We compare the results of the first-order network and the second-order network and show that the second-order network is better able to generalize as a one-class classifier.*

1. Introduction In our previous paper [5], we implemented a multi-layer feedforward network with first-order inputs to process a 32 x 32 receptive field of raw grey-level data within a 256 x 256 image. Invariance to shift was achieved by replicating the network for each registration of the receptive field within the larger image. The resulting feedforward network consisted of the parallel implementation of 50,625 identical receptive field networks, which collectively process the entire input image and can detect the presence of a target regardless of its position within the input image. We trained the receptive field network with digitized video images of nine different miniature vehicles with 10% noise added. Invariance to perspective was achieved by training the network with representative perspective images taken from three different depression angles and from every 10° aspect as the vehicles were rotated through 360°. To achieve reliable false alarm rejection, we introduced the method of iterative training. For iterative training, we first trained the network with 800 to 1000 receptive field images representative of the target and non-target vehicles of interest. We then tested the trained network against large numbers of non-target images. We extracted those images that generated false alarms and we used them to augment the original training set. We found that with four iterations of training, the network was able to achieve 91% detection of the target vehicle and 3% false alarm rate on the non-target vehicles in the training set independent of perspective [5]. The network was shown to be capable of generalizing the within-class perspective view information.

The iteratively trained first-order network was not capable of one-class generalization. When it was tested with non-target objects that were not part of the training set, its performance was unacceptable. This experience is consistent with our intuition about the nature of the decision boundaries formed by a first-order network. The first-order network has the capability of forming only hyperplane decision boundaries in its first layer of hidden nodes. The succeeding layers then can form combinations of these linear decision boundaries. In order to form a one-class

classifier with the first-order network, it would be necessary to completely surround the target class with a closed decision boundary. For a network with n-dimensional inputs, forming a closed decision boundary would require $n+1$ nodes in the first hidden layer [2]. If fewer than n nodes are used in the first hidden layer, a target class cannot be completely surrounded, and invalid one-class generalization will result. As an example, consider a two-dimensional problem with only two nodes in the first hidden layer. The two hyperplanes can never completely surround a finite-size target class, and as a result there will always be an infinitely large part of the feature space where invalid generalization results.

Maxwell and Giles have shown that higher order networks can generalize more capably and more efficiently than first-order multi-layer networks [3]. This is consistent with our intuition, because the higher order network has the capability of forming a closed hyperellipsoidal decision boundary with only a single node. Also, Psaltis has shown that the storage capacity of an associative memory can be increased by implementing polynomial discriminant functions of a higher order [6]. We have implemented a second-order network trained with backpropagation to test its one-class generalization ability. We compare the generalization ability of the second-order network to that for a first-order network trained in the same way.

2. Training the Second-order Network for One-class Classification Because a full second-order implementation would require more than one million input connections, the cross terms in the second-order network were set to zero, and only the squared data values were input to the network. Therefore, the network is limited to forming high order decision boundaries that are aligned with the axes of the input data samples. The network has 2048 inputs, 16 nodes in one hidden layer, and one output node. Higher order terms are generated for the input layer only, not for the hidden layer. The network contains a hidden layer to account for the possibility that the target class is multi-modal. The performance of this network with second-order inputs was compared to a single-order input network with 1024 inputs, 32 nodes in each of two hidden layers and one output node. The sizes of these two networks were chosen for comparison because both contain approximately the same number of weights. The first-order network has 33,889 weights and the second-order network has 32,801 weights. Unlike the network in reference [5], both these networks were trained iteratively using only image data from a fixed training set containing the nine vehicles; no false alarms from additional objects were included in the training set.

The first-order network was trained for three iterations. For each iteration, the network trained for hundreds of epochs before it converged. Then, the network was tested and false alarm images were extracted for the next iteration of training. The training set for the first iteration contained 792 images, and the training sets for the second and third iterations contained 1411 images. The second-order network was first trained with a special noise addition method [5]. It was then trained for three iterations, each of which required hundreds of epochs to converge. The training set for the first and second iterations contained 954 images, and the training set for the third iteration contained 690 images. In each case, the network trained iteratively until it achieved reasonable performance on a number of the images in the fixed size training set. The detection thresholds for both networks were set to the same value.

3. Results The two networks achieved similar performance on a number of test images extracted from the training set. Both networks were then tested on new images, not included in the training set. Figure 1 shows Test Image #1 containing all the training objects in new orientations plus a telephone, an object not part of the training set. The target object is the tank in the lower right corner of the image. Figure 2 shows the thresholded response of the first-order network, and Figure 3 shows the thresholded response of the second-order network. The first-order network generalizes incorrectly because it responds to the telephone image with a false alarm. The second-order

network generates no false alarms. Figure 4 shows Test Image #2, another arrangement of the objects from Figure 1. Notice that in this figure, the training set vehicles are spaced more closely than they were in Test Image #1. Figure 5 shows that the first-order network generates a number of false alarms even for the vehicles from its training set. It responds with false alarms for the following reasons. The feedforward network processes one receptive field network for each registration of the 32 x 32 receptive field within the input image. When the vehicles are spaced closely together, there are a number of receptive field images that see a piece of more than one vehicle. Since the network was not trained with more than one vehicle at a time, the network treats these pieces as unknown objects. Some of these multi-piece objects fall in regions of the input space which, by default, have been declared to be target regions. The first-order network must contain lots of these default target regions because it does not have enough hyperplane decision boundaries to completely surround and close off the true target regions. Figure 6 shows that the second-order network does not experience false alarms for the Test Image #2. We also tried a third experiment to test the ability of the two networks for one-class generalization. Test Image #3 contained an image of a white sheet of paper. The first-order network responded to the white paper image with a large region of false alarms at the edge of the paper. The second-order network indicated no false alarms at all.

4. Conclusions We have trained two networks, one with first-order inputs and another with first and second-order inputs, to recognize a single target vehicle and reject eight non-target vehicles from a fixed training set. While both networks perform comparably on the training set, the second-order network rejects more examples of new non-target objects. These preliminary results show that the second-order network generalizes as a one-class classifier better than the first-order network. References [3] and [5] have shown that higher order networks provide better generalization for learning boolean functions and better capacity in associative memories. We have extended these results to show that higher order networks also provide better generalization in networks trained for image classification. We also plan to test the second-order network on a larger number of images for more statistically significant results. To do so, we are now implementing the second-order network on the high-speed feedforward pipeline which implements the feedforward network at more than 2.5 billion connections per second [1] [5].

5. References

[1] R. J. Fogler, R. L. Williams, and L. D. Hostetler, "A Billion Connection per Second Feedforward Pipeline for Computer Vision Applications," presented at INNS, Boston, MA, Sept. 6-10, 1988.

[2] D. R. Hush and J. R. Salas, "On Determining the Number of Nodes/Layers in a Multi-layer Perceptron Classifier," submitted to *Neural Computation*.

[3] T. Maxwell, C. L. Giles, and Y. C. Lee, "Generalization in Neural Networks: The Contiguity Problem," *Proceedings of the First IEEE ICNN*, Jul. 1987, Vol. II, pp. 41-46.

[4] M. M. Moya, R. J. Fogler, and L. D. Hostetler, "Back-propagation Networks for Perspective-invariant Pattern Recognition in SAR Imagery," presented at INNS, Boston, MA, Sept. 6-10, 1988.

[5] M. M. Moya, R. J. Fogler, and L. D. Hostetler, "Generalization in Backpropagation Networks: An Empirical Study Using Image Data," submitted to NIPS, Denver, CO, Nov. 27-30, 1989.

[6] D. Psaltis and C. H. Park, "Nonlinear Discriminant Functions and Associative Memories," Neural Networks for Computing, AIP Conference Proceedings #151, ed. J. S. Denker, pp. 370-375, 1986.

[7] See also references listed in [5]

Figure 1. Test Image #1

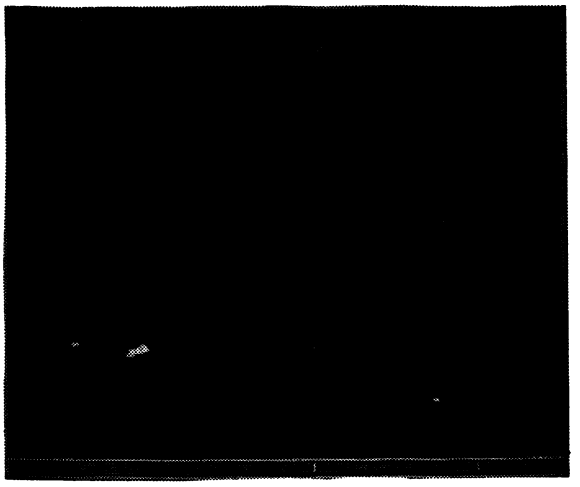


Figure 2. First-order Network Response to Test Image #1

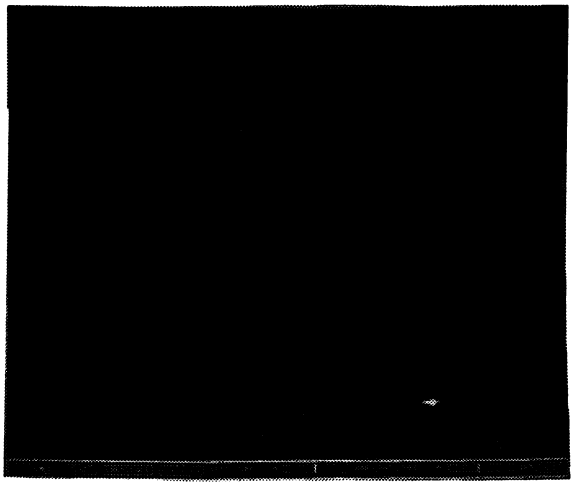


Figure 3. Second-order Network Response to Test Image #1

Figure 4. Test Image #2

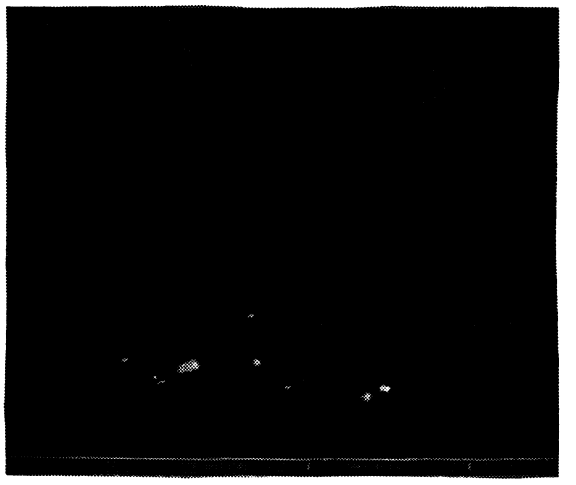


Figure 5. First-order Network Response to Test Image #2

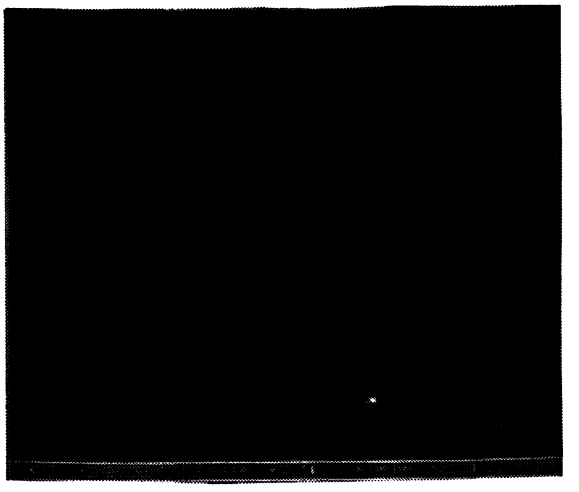


Figure 6. Second-order Network Response to Test Image #2

Model-Based Perceptual Grouping (MPG): A Cooperative-Competitive Approach to Shape Recognition in Neural Networks

J. Michael Oyster[1] and Nancy B. Lehrer

The Hughes Signal Processing Laboratory[2]
Building E53, Mail Station E250
PO Box 902
El Segundo, CA 90245
email: oyster%tcville@hac2arpa.hac.com telephone: 213-616-8776

Abstract

A neural network for recognizing complex shapes in natural scenes is described. Using an energy function formalism, a regularization network is derived to extract edges which are consistent with a model representation. Models of specific shapes are encoded in a parameter network. Top-down feedback drives the formation of edge boundaries given they are consistent with model information. It is demonstrated using real images that model-driven, top-down feedback provides superior performance to a purely data-driven, feedforward network. Because our model is general enough to admit a wide variety of useful features and shape representations, it is an attractive paradigm for segmentation and perceptual organization.

Introduction

The recognition of visual objects in natural scenes is a major scientific challenge. Real images may contain a million pixels of grey level information and a totally connected network processing a one million pixel image requires over one trillion connections. This number of connections is difficult to achieve in a practical system. The raw images span a feature space of very high dimensionality and the discriminant surfaces in this feature space are topologically complex. Because it is difficult to achieve learning convergence under these conditions, it is necessary to systematically reduce the complexity of the image to a point where the recognition problem becomes tractable.

A standard solution technique for reducing the complexity of an image is segmentation. Segmentation divides the scene into disjoint subregions. Associated with each subregion is the collection of original grey levels defined over the subregion, the perimeter of the subregion, and a list of features for each subregion. Segmention provides a simplified description of the scene which greatly reduces the complexity of the recognition process.

Unfortunately, the segmentation process is seldom error free in real imagery. Because real images are complicated by noise, occlusion, complex lighting and other confounding factors, it is extremely difficult to reliably partition the image into segments and accurately assign the correct attributes to each segment. Segmentor error results in poor classification performance, and whether a conventional or a neural network classifier is used, the segmentor is typically the bottleneck to the attainment of a high probability of recognition.

Segmentors fail largely because the data they receive is inherently ambiguous and they have no mechanism for recovering from the ambiguity. For example, it is difficult to discriminate between an object boundary and a shadow boundary. Shadows and other confounding factors can only be disambiguated by appeal to higher level knowledge (eg. knowledge of the shapes being sought after). This leads us to the *segmentation/recognition dilemma*: it is hard to classify shapes in a scene without segmentation, and it is hard to achieve good segmentation without having classified the shapes.

A solution to the segmentation/recognition dilemma is to provide controlled feedback from the shape recognition process to the segmentation process. A conventional approach to this problem is to implement feedback using a hypothesis generation and test procedure, but this approach is far too slow and brittle to be practical on real time systems working with real images. We suggest that cooperative-competitive neural network models provide the

[1]Person to whom correspondence should be addressed.

[2]This work was funded under DARPA contract #DAAA21-88-C-0183

right conceptual framework for solving the segmentation/recognition dilemma and that neural networks derived from Markov Random Field (MRF) models provide the mathematical and computational tools required to make the cooperative-competitive computations do their job effectively.

Neural network systems that systematically exploit cooperative-competitive behavior can be derived from MRFs[4]. In particular, a rich theory for early vision, regularization[7,9,10], can be derived from the MRF formalism. Koch and Yuille[8] have shown how to derive Hopfield-like neural network models which provide deterministic solutions to the MRF. Their neural network update equations provide a strictly negative time derivative for the energy function thus preventing limit cycle behavior. The energy function formalism assures superior performance over simple connectionist models based upon the ad-hoc linking of cooperating and competing units.

Model-Based Perceptual Grouping: A General Approach

We perform perceptual organization in two stages collectively called Model-Based Perceptual Grouping (MPG). In the first stage, regularization, features and segmentation boundaries are computed. In the second stage, perceptual grouping, the features and boundaries are organized to form collections of shape and segmentation hypotheses. These hypotheses, encoded by patterns of neural activity, are formed in parallel and compete for activation in the neural network. The regularization network receives feedback from the perceptual grouping network to enhance the formation of boundaries and features given they are consistent with the coexistent perceptual groupings. As the network converges, only the most successful perceptual grouping survives.

Prototype for MPG: The Weak Membrane Coupled with a Parameter Network

A prototypical MPG network (Figure 1) is constructed by combining a regularization network which extracts edge features and a perceptual grouping network which groups the edge features. Edges are extracted using the weak membrane model[3,8] and the edges are grouped together to form target outlines using a parameter network[2].

The regularization network extracts edge information from the scene by minimizing the following Hopfield-like[6] energy function:

$$\sum_{i,j}(f_{i,j}-d_{i,j})^2+\lambda^2\sum_{i,j}(f_{i,j}-f_{i+1,j})^2(1-h_{i,j})+(f_{i,j}-f_{i,j+1})^2(1-v_{i,j})+\alpha\sum_{i,j}h_{i,j}(1-p_{i,j})+\alpha\sum_{i,j}v_{i,j}(1-p_{i,j}) \quad (1)$$

In this formula, $d_{i,j}$ represents the input sensor data, $f_{i,j}$ represents the regularized output, and $h_{i,j}$ and $v_{i,j}$, are continuous variables representing the confidence that vertical and horizontal edges are present at a particular position. The variables $h_{i,j}$ and $v_{i,j}$ are called *line processes*. The difference terms, $\lambda^2\sum_{i,j}(f_{i,j}-f_{i+1,j})^2(1-h_{i,j})$ and $\lambda^2\sum_{i,j}(f_{i,j}-f_{i,j+1})^2(1-v_{i,j})$, measure smoothness, the term, $\alpha\sum_{i,j}l_{i,j}(1-p_{i,j})$, provides a penalty for introducing edge discontinuities, and the term, $\sum_{i,j}(f_{i,j}-d_{i,j})^2$, measures the least squares distance between the data and the output. The factors $1-h_{i,j}$ and $1-v_{i,j}$ couple the edge formation process to the smoothing process, and the factor $1-p_{i,j}$ couples the model output to the edge formation process.

Following Hopfield [6] we derive a set of neural network update equations:

$$\frac{df_{i,j}}{dt}=-\frac{\partial E}{\partial f_{i,j}} = -2\left(f_{i,j}-d_{i,j}\right)-2\lambda^2\left(f_{i,j}-f_{i-1,j}\right)\left(1-h_{i,j}\right)+2\lambda^2\left(f_{i,j+1}-f_{i,j}\right)\left(1-v_{i,j}\right) \quad (2)$$

$$\frac{dV_{i,j}}{dt}=-\frac{\partial E}{\partial v_{i,j}} = \lambda^2\left(f_{i,j}-f_{i-1,j}\right)^2-\alpha \quad (3)$$

$$\frac{dH_{i,j}}{dt}=-\frac{\partial E}{\partial h_{i,j}} = \lambda^2\left(f_{i,j}-f_{i,j-1}\right)^2-\alpha \quad (4)$$

where $v_{i,j}=g(V_{i,j})=\frac{1}{1+e^{-2\mu V_{i,j}}}$ and $h_{i,j}=g(H_{i,j})=\frac{1}{1+e^{-2\mu H_{i,j}}}$.

The values of the $p_{i,j}$ are updated using a parameter network. We use a parameter network which encodes shapes by a mechanism similar to the Generalized Hough Transform (GHT)[1]. The outputs of the regularization network, $f_{i,j}$, are passed through a collection of neurons with oriented receptive fields[5]. The link weights of the

parameter network perform a correlation on the outputs of the oriented receptive fields. The neurons of maximal activity indicate the presense of a particular shape in the scene. This shape is then backprojected as a control signal to the horizontal and vertical line process neurons in the regularization network.

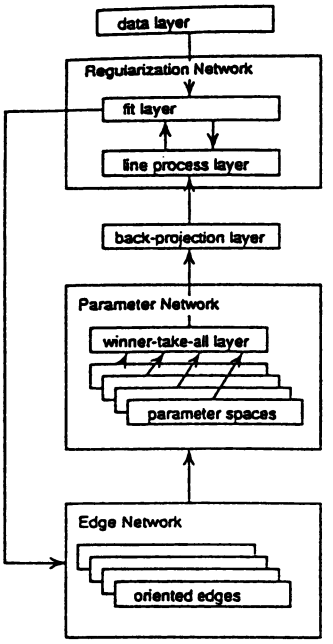


Figure 1: The MPG network. The smoothing and continuity constraints of the regularization network are coupled with the shape constraints of the parameter network.

Experimental Results

Figure 2 shows the results of applying the MPG network to a very noisy infrared image of a tank. This image is representative of the images that must be processed by military Automatic Target Recognizers (ATR). The image is small and the signal-to-noise ratio is low. Substantial blurring occurs around the tank perimeter. The target contains a hot spot which tends to confuse edge detectors using a global adaptive threshold.

The result of applying the Sobel edge operator to the image is shown in Figure 2(b). The Sobel edge operator is unable to find all of the edge features because the image is very noisy and the image gradients tend to take on a bimodal edge distribution. Figure 2(c) shows the output of the weak membrane regularization network (without feedback) on the tank image. The weak membrane recovers more of edges than with the Sobel operator but many of the edges are displaced due to extreme blurring around the target perimeter. Once the selection of edges is driven by a shape model (Figure 2(d)), the performance improves dramatically.

Conclusions

We have demonstrated the feasibility of using models to drive the emergence of features and segmentation boundaries. We have shown it is possible to couple model information with the extraction of oriented edge features using an energy function formalism and that this coupling results in improved recovery of edge information in a complex image. MPG networks can be used in conjunction with any early vision process based upon a Markov Random Field (MRF). Since the form of the coupling terms is independent of the model representation, MPG networks can utilize a wide variety of shape recovery techniques. MPG networks provide a useful paradigm for solving problems in segmentation and perceptual organization.

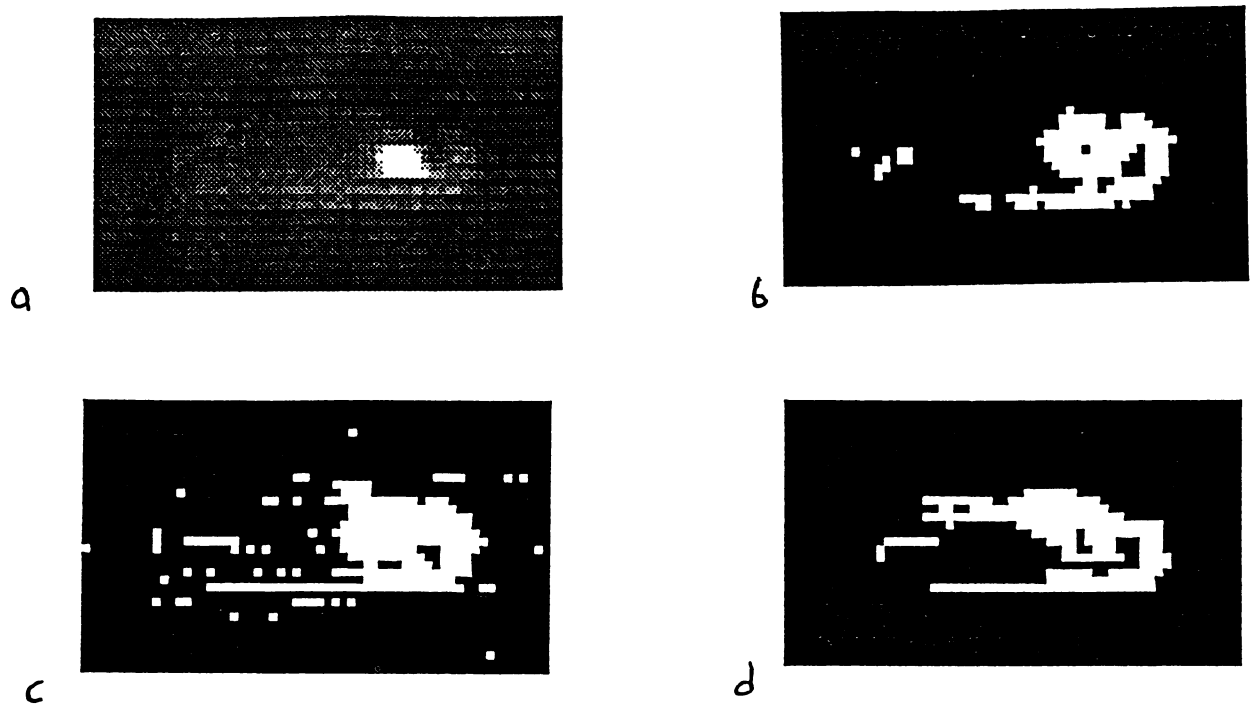


Figure 2: Edge extraction is improved through feedback from the perceptual grouping network. (a) The input image to the network - an infrared tank image. (b) The Sobel edge operator fails to find all of the edges of the tank. (c) Regularization using the weak membrane improves the performance slightly, but considerable noise is introduced. (d) Model-driven feedback greatly improves the recovery of the target shape and the rejection of noise.

References

[1] D.H. Ballard. Generalizing the Hough Transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–112, 1981.

[2] D.H. Ballard. Parameter nets. *Artificial Intelligence*, 22:235–267, 1984.

[3] Andrew Blake and Andrew Zisserman. *Visual Reconstruction. The MIT Press Series in Artificial Intelligence*, MIT Press, Cambridge, Massachusetts, 1987.

[4] Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

[5] S. Grossberg and E. Mingolla. Neural dynamics of surface perception: Boundary webs, illuminants, and shape from shading. *Computer Vision, Graphics, and Image Processing*, 37:116–165, 1987.

[6] J. J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proc. Natl. Acad. Sci. USA*, 81:3088–3092, May 1984.

[7] B. K. P. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[8] Christof Koch, Jose Marroquin, and Alan Yuille. *Analog "Neuronal" Networks in Early Vision.* AI Memo 751, MIT AI Laboratory, June 1985.

[9] T. Poggio, H. Voorhees, and A. Yuille. *A Regularized Solution to Edge Detection.* AI Memo 833, MIT AI Laboratory, May 1985.

[10] Tomaso Poggio, Vincent Torre, and Christof Koch. Computational vision and regularization theory. *Nature*, 317:314–319, Sep 1985.

Neural Computation for Collision-free Path Planning

Jun Park* and Sukhan Lee**
*,** Department of Electrical Engineering-Systems
University of Southern California, L.A. CA 90089-0782
* ETRI POB 8 Daedog Science Town, Korea

Abstract

An algorithm using a neural network is proposed for the collision-free path planning problem in which a polyhedral object goes through a set of polyhedral obstacles. For each obstacle, a collision penalty function is defined with a three-layer neural network which computes how much a path collides with that obstacle. Then, energy is defined based on the collision penalty function and the length of the path. By moving the path so that energy is minimized, a collision-free path can be obtained at the equilibrium state. By simulation, the behavior of the algorithm is examined.

1. Introduction

Collision-Free Path Planning is a well-known problem in robotics. Given an object with an initial position and a goal position, and a set of obstacles located in space, the problem is to find a continuous path for the object to move from the initial position to the goal position which avoids colliding with obstacles along the trajectory. Many algorithms such as the configuration space approach[1] and the generalized cone approach[2], have been developed for this problem. These schemes require some preprocessing and graph search technique, and thus can be quite expensive.

This paper presents a new approach to solve the collision path planning problem by utilizing **Neural Optimization Network** Concept [3, 4]. The idea of most neural networks for optimization is to move in a space of possible configurations representing solutions so that progress is achieved in a direction that tends to minimize the cost function, and the space and method of moving are smooth enough that a good solution will ultimately be reached. To solve the collision-free path planning problem, we first define the collision penalty function which is used to determine how much an object collides with obstacles along the path. Next, by defining the energy and then deriving the dynamical equations, the path planning algorithm is developed for an object represented by a point. This algorithm is simply extended to the case of a polyhedral object. Finally, simulation results are presented.

2. Neural Computation for path planning

A typical situation in the 2 dimensional case is shown in Figure 1. The initial position and the goal position are specified as a point, and a set of obstacles are represented by a set of polygons. To find a feasible path, we choose representations for initial and goal positions, obstacles, and a path as follows:

- Initial and Goal position: The initial point and the goal point can be represented by the 3(2) coordinates in the 3(2) dimensional space.
- Obstacles: The polyhedral obstacle can be represented by a set of inequality constraints, which are mapped one-to-one to planes(edges) in the 3(2) dimensional space. With these inequality constraints, it is possible to decide whether any point is inside the obstacle or not.
- Path: The path is determined by a set of via points and is approximated by the line segments between the adjacent via points. Each via point can be specified by the 3(2) coordinates in the 3(2) dimensional space like the initial/goal point.

With these representations, the basic procidure in our algorithm can be described as follows:

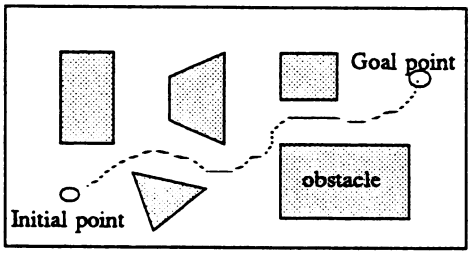


Figure 1. Typical situation in the 2-D space

Given coordinates of the initial and goal points and sets of constraints of obstacles, via points of the path are arbitrarily chosen. These via points are desired to move in space, and finally to converge to the positions which make the optimal collision-free path.

Collision Penalty Function

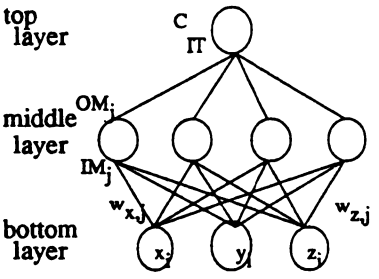
Because the path for an object is represented by a set of via points, its collision with obstacles can be considered as a sum of the collision between its via points and obstacles. To determine the degree of collision between a point and an obstacle, the collision penalty function is defined as a three layer neural network for each obstacle as shown in Figure 2. Each of the three units in the bottom layer represents respectively the x, y, z coordinate of a point. Each unit in the middle layer corresponds to one inequality constraint of the obstacle. The connections between the bottom layer and the middle layer are assigned to the coefficients of x, y, z in the inequality constraints, and the threshold of a middle layer unit is assigned to the constant term in its inequality constraint. The connections between the top layer and the middle layer are all assigned to 1 and the threshold of the top layer unit is assigned to 0.5 less than the number of constraints.

When a point is given to the units in the bottom layer, each of the middle layer units decides whether the given point satisfies its constraint. The top layer unit is "on" only if all of the middle layer units are "on", i.e. all the constraints are satisfied. This means that the network outputs "on" only when the point is inside the obstacle, thus a path collides with the obstacle if any of its via points force the network to output "on". Therefore, with a set of networks, we can compute the degree with which the path collides with the obstacles. By using the sigmoid function

$$f(x) = k / (1 + \exp(- x / T)) \qquad (1)$$

as its activation function, the analysis can be easily done for continuous movement.

An example for the collision penalty function is shown in Figure 3. The rectangular obstacle in (a) has four inequality constraints as shown in (b). The corresponding network is illustrated in (c), and the three dimensional view of the collision penalty function is drawn in (d). This shape can be changed by adjusting the gain, k, or the temperature, T, in the activation function. Since the network produces a type of potential field, it can be used for the artificial potential field approach for a polyhedral object.



$C = f(IT)$

$IT = \sum_{j=1,M} OM_j + \Theta T$

$OM_j = f(IM_j)$

$IM_j = w_{x,j} * x_i + w_{y,j} * y_i + w_{z,j} * z_i + \Theta M_j$

C: output of the top layer unit

IT: input of the top layer unit

OM_j: output of the j^{th} middle layer unit

IM_j: input of the j^{th} middle layer unit

$w_{x,j}$, $w_{y,j}$, $w_{z,j}$: coefficients in the j^{th} constraint

Figure 2. Network for a collision penalty function

<u>Energy and Dynamical equations</u>

In collision-free path planning problem, there are two subgoals to be accomplished. One is to avoid colliding with obstacles, and the other is to minimize the length of the path. These two factors are to be considered in defining the energy function. As for the path length, the energy is simply defined as the sum of squares of all the lengths of the line segments connecting the adjacent via points, $P_i(x_i, y_i, z_i)$, for i=1, 2,..., N, as:

$$E_L = \sum_{i=1,N} L_i^2 \qquad (2)$$
$$= \sum_{i=1,N} [(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 + (z_i - z_{i-1})^2]$$

where L_i is the i^{th} line segment length and N is the number of via points. As the path converges to the minimal one, E gets smaller, and the via points tend to be equally separated due to the usage of its square rather than the line segment length itself. With the collision penalty function, we define the energy for collision as:

$$E_C = \sum_{i=1,N} \sum_{k=1,M} C_i^k \qquad (3)$$

where M is the number of obstacles and C_i^k is the collision penalty at the i^{th} via point, P_i, due to the k^{th} obstacle. With those two measures, we define the total energy, E, as:

$$E = E_L + E_C. \qquad (4)$$

Since low energy implies less collision and a shorter path, the dynamical equation for a via point, $P_i(x_i, y_i, z_i)$, is chosen to make the time derivative of the energy, E, be negative along the trajectory. The derivative of E with respect to time is

$$dE/dt = \sum_i (\nabla P_i E)\, dP_i/dt$$
$$= \sum_i \{ [\partial L_i/\partial x_i + \sum_k \partial C_i^k/\partial x_i] * dx_i/dt$$
$$+ [\partial L_i/\partial y_i + \sum_k \partial C_i^k/\partial y_i] * dy_i/dt$$
$$+ [\partial L_i/\partial z_i + \sum_k \partial C_i^k/\partial z_i] * dz_i/dt \} \qquad (5)$$

So by choosing the time derivative of each coordinate as:

$$dx_i/dt = - [\partial L_i/\partial x_i + \sum_k \partial C_i^k/\partial x_i]$$
$$dy_i/dt = - [\partial L_i/\partial y_i + \sum_k \partial C_i^k/\partial y_i] \qquad (6)$$
$$dz_i/dt = - [\partial L_i/\partial z_i + \sum_k \partial C_i^k/\partial z_i],$$

the time derivative of energy becomes:

$$dE/dt = - \sum_i \{ [dx_i/dt]^2 + [dy_i/dt]^2 + [dz_i/dt]^2 \} < 0$$

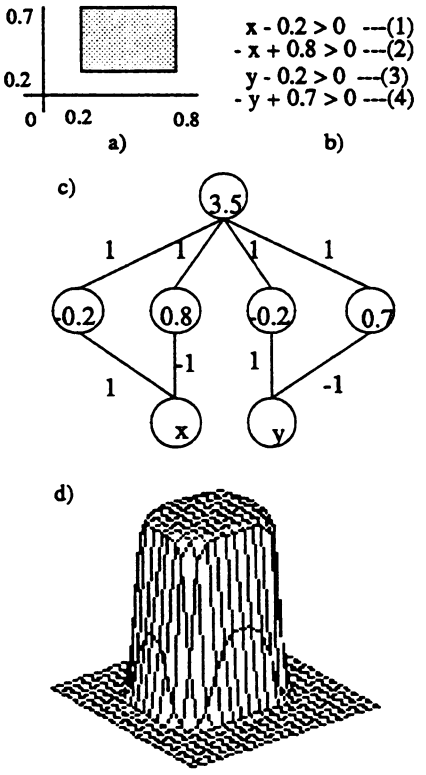


Figure 3. An example of the collision penalty function for a rectangular obstacle. a) Obstacle b)Its constraints c) Network d) 3-D shape with T=0.05, k=1

along the trajectory, and

$dE/dt = 0$ if and ony if $dx_i/dt = 0$, $dy_i/dt = 0$, and $dz_i/dt = 0$.

This means that all via points move along trajectories that decrease the energy, and finally reach equilibrium positions. From Eq. (6) and Figure 2,

$$\partial C/\partial x_i = \partial C/\partial IT * \partial IT/\partial x_i$$
$$= \partial C/\partial IT * \sum_{j=1,J} \partial OM_j/\partial x_i$$
$$= \partial C/\partial IT * \sum_{j=1,J} \partial OM_j/\partial IM_j * \partial IM_j/\partial x_i \quad (7)$$
$$= f'(IT) * \sum_{j=1,J} f'(IM_j) * w_{x,j}.$$

where J is the number of constraints and f '(.) is equal to (k/T) * f(.) * [1 - f(.)]. Thus, from equations (6) and (7), the dynamical equation for x_i is derived as:

$$dx_i/dt = -[(2x_i - x_{i-1} - x_{i+1}) + \sum_k (k/T) * f(IT^k) * [1-f(IT^k)] * \sum_{j=1,J} (k/T) * f(IM_j^k) * [1-f(IM_j^k)] * w_{x,j}^k]$$
$$dy_i/dt = -[(2y_i - y_{i-1} - y_{i+1}) + \sum_k (k/T) * f(IT^k) * [1-f(IT^k)] * \sum_{j=1,J} (k/T) * f(IM_j^k) * [1-f(IM_j^k)] * w_{y,j}^k] \quad (8)$$
$$dz_i/dt = -[(2z_i - z_{i-1} - z_{i+1}) + \sum_k (k/T) * f(IT^k) * [1-f(IT^k)] * \sum_{j=1,J} (k/T) * f(IM_j^k) * [1-f(IM_j^k)] * w_{z,j}^k]$$

This analysis for the collision penalty function is similar to that of the back-propagation algorithm. The difference is that the input is varying in this algorithm, while the connection weights are varying in the back-propagation algorithm[5].

Path Planning for a polyhedral Object

The above algorithm for an object represented by a point is easily extended to the case of a polyhedral object. The collision between an object and an obstacle implies that some points within the object collide with the obstacle. Thus, the collision penalty of a polyhedral object can be considered as the sum of the collision penalties at some of its points which can be placed at the vertices, edges, planes, etc.

For a polyhedral object, rotation as well as translation can occur along the path. Thus, due to rotation, one or three more independent variables are needed to represent via positions in two or three dimensional space. And dynamical equations for the rotational variables are also required. The i^{th} via position in the two dimensional space can be represented by a reference point, $P_{i,0}(x_{i,0}\ y_{i,0})$, and a rotational variable, θ_i. And each of the collision test points, $P_{i,s}(x_{i,s}, y_{i,s})$, j=1, 2,...,S, can be determined by a fixed distance, d_s, and an angle, ϕ_s from the reference point. Based on these observations, the energy is modified as:

$$E = E_L + E_C = \sum_{i=1,N} L_i^2 + \sum_{s=1,S} \sum_{i=1,N} \sum_{k=1,M} C_{i,s}^k \quad (9)$$

where L_i is a distance between the reference points of the i^{th} and $(i-1)^{th}$ via positions and $C_{i,s}^k$ is the collision penalty at the s^{th} test point of the i^{th} via position due to the k^{th} obstacle. In two dimensional space, the time derivative of the energy is

$$dE/dt = \sum_i \{ \partial E/\partial x_{i,0} * dx_{i,0}/dt + \partial E/\partial y_{i,0} * dy_{i,0}/dt + \partial E/\partial \theta_i * d\theta_i/dt \} \quad (10)$$

From Eq.(10), the dynamical equation for translation can be derived in a similar way as Eq. (8), and the dynamical equation for a rotational variable, θ_i, is chosen as

$$d\theta_i/dt = -\partial E/\partial \theta_i = -\sum_s \sum_k (k/T) * f(IT^{k,s}) * [1-f(IT^{k,s})] * \sum_j (k/T) * f(IM_j^{k,s}) * [1-f(IM_j^{k,s})] * d_{i,s} * [-w_{x,j}^k * \sin(\theta_i + \phi_s) + w_{y,j}^k * \cos(\theta_i + \phi_s)] \quad (11)$$

where the superscript, k and s,represents the k^{th} obstacle and the s^{th} test point.

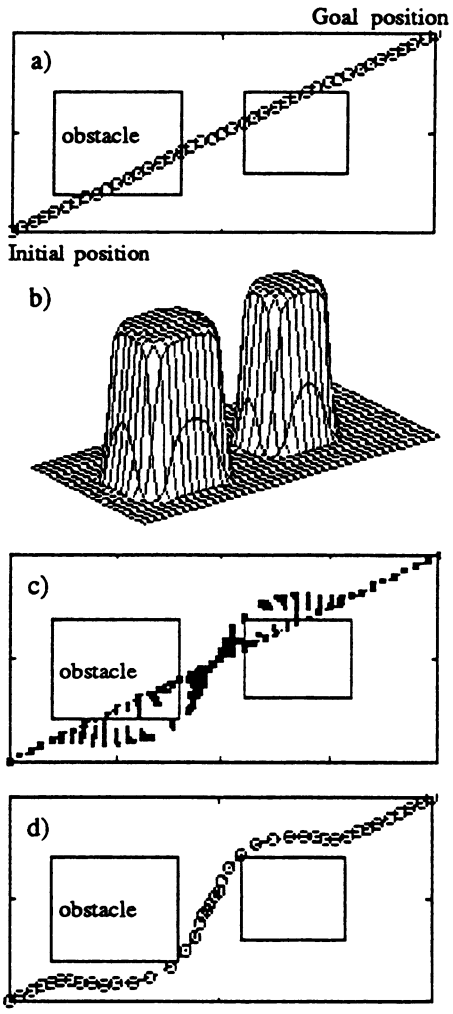


Figure 4. Path planning for a point object with two obstacles a) Initial path b) 3-D view of the C.P.F. c) Trajectories of the via points d) Final path

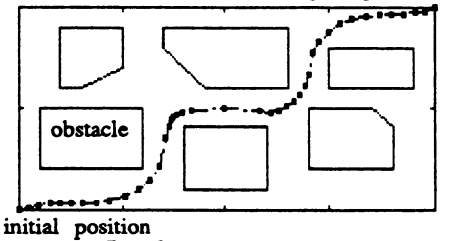


Figure 5. Path from the virtuaboundary

3. Simulation Results

Through computer simulation, the behavior of the proposed algorithm is examined. Figure 4 shows the results for an object represented by a point with two obstacles. The initial path is arbitrarily chosen as a straight line from the initial position to the goal position as shown in (a). The shape of the collision penalty function for the obstacles is shown in (b). At the beginning of the computation, the via points near the boundaries of the obstacles move much more than the others. This is because the slope of the collision penalty function of an obstacle is steep at its boundary. The trajectory illustrated in (c) is obtained by plotting all the via points which converge to form the collision-free path illustrated in (d).

Because the path is approximated by its via positions, the resultant path may collide with obstacles while passing from one via position to its adjacent via position. This situation can be improved by increasing the number of via positions. With more via positions, the distance between the adjacent via positions gets smaller, thus decreases the chance of collision. In such a case, the computation time remains constant, even though the size of the network is increasing. This is because the computation for each via position can be totally localized, and thus can be performed in parallel. It is also possible to get a safer path by introducing the virtual boundary for the obstacle. The enlarged virtual obstacle can be placed at the same location as the real obstacle. By using the virtual boundary, the resultant path can avoid colliding with the real obstacles while it may collide with the virtual boundary itself. Figure 5 shows that a safer path can be obtained with the virtual boundary.

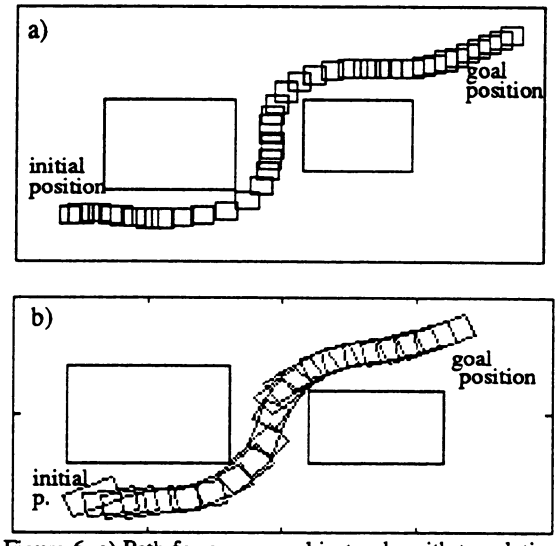


Figure 6. a) Path for a square object only with translation b) Path for a rectangular object with rotation as well as translation

The results for a polyhedral object are shown in Figure 6. The path in (a) is allowed only translation while the path in (b) is allowed rotation as well as translation. The above methods to obtain a safer path can be applied also to the polyhedral object case. Since the object itself is approximated by a few test points, a safer path can also be obtained by increasing the number of test points.

However, the algorithm does not guarantee a globally optimal path. This is due to the local minima of the energy. Thus, depending on the initial path assignment, the final path can converge to the locally minimal path. Moreover, the resultant path may not correspond to the feasible path in some cluttered environment, while this can be detected using the length of adjacent via positions which tend to be equally separated in the feasible path. For initial path assignment and path verification, a combination of a higher level planning scheme with the proposed algorithm is desirable.

4. Conclusion

In collision-free path planning, there are two subgoals to be accomplished. One is to avoid a collision between a moving object and obstacles, and the other is to minimize the length of the path. To determine how much collision occurs, a collision penalty function is presented as a three-layer network. This network produces a kind of potential field around an obstacle, so that it can be easily applied to the artificial potential field approaches for a polyhedral object. By simply adjusting the gain or the temperature of the activation function in the neuron, the shape of the field can be altered.

Based on this collision penalty function, the path planning algorithm is developed for an object represented by a point, and then extended to the case of a general polyhedral object. Simulation results show that the proposed algorithm finds a collision-free path for both an object represented by a point and a polyhedral object. Because the path is approximated as a set of via positions, the object may collide with obstacles while passing from one via position to its adjacent via position. Such a situation is improved by increasing the number of via positions. Also, with the virtual boundary concept, a safer path can be obtained. Since the proposed algorithm suffers from the local minima problem, a higher level planning scheme could be implemented in combimation with it.

References

[1] Tomas Lozano-Perez,"Spatial Planning: A Configuration Space Approach," IEEE Trans. on Computers. Vol C-32 No. 2 Feb. 1983

[2] Rodney A. Brooks, "Solving the Find-Path Problem by Good Representation of Free Space," IEEE Trans. on Sys.Man and Sybern. Vol SMC-13 No.3 Mar-Apr. 1983

[3] Hopfield, J.J. and Tank, David W. "Computing with Neural Circuits: A Model" SCIENCE, Vol.233 pp.625-633 August 1986

[4] Hopfield, J.J. and Tank, David W. "Neural Computation of Decision in Optimization Problems" Biol. Cybernetics 52, 141-152(1985)

[5] D.E.Rumelhart, G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation" in D.E. Rumelhart & McClelland(Eds.), Parallel Distributed Processing, Vol 1. MIT Press (1986)

Learning Aspect Graph Representations of 3D Objects in a Neural Network[1]

Michael Seibert and Allen M. Waxman
Machine Intelligence Group
MIT Lincoln Laboratory
Lexington, MA 02173-9108

Abstract. View-based representations, such as "aspect graphs," are well-suited for recognizing 3D objects when both the views and the relationships (i.e., transitions) between views are stored and exploited. Using concepts in neural learning originally developed by Grossberg (1982), we show how an "aspect network" (based on the "aspect graph" concept of Koenderink and van Doorn (1979)) evolves from an initially amorphous (fully interconnected) adaptive network, fusing sequences of 2D-invariant views of a 3D object into a coherent graph-like representation. Following learning of the network, we suggest how this representation can be used to recognize 3D objects from previously experienced or novel view-sequences. This work extends our neuromorphic vision system for learning and recognition of 2D shapes (Seibert & Waxman, 1989a) to include the learning of representations for 3D objects.

Our perception of events is sequential, yet we have little difficulty building multidimensional representations from collections of experiences. In the vision application motivating this contribution, we wish to combine the many views arising from the exploration of a 3D object into a single representation. Moreover, this representation should be self-organizing and useful for recognizing novel view sequences of this object. We have recently designed a network, called an *aspect network*, based on the concept of an *aspect graph* proposed by Koenderink and van Doorn (1979). An aspect network extends our 2D object recognition system (Seibert & Waxman, 1989a) by organizing 2D-invariant[2] view sequences of 3D objects into coherent graph representations suitable for recognizing the objects. An aspect network extends the aspect graph concept by incorporating self-organizing competitive learning and temporal sequence recognition.

The network which emerges through learning is analogous to an undirected, cyclic graph. Each node of an aspect graph represents a single 2D-invariant view from a sequence. Let us refer to a single view as an "*aspect*" (or a "*symbol*" in other applications). An arc represents a feasible transition between adjacent aspects in a sequence. The collection of arcs store the relationships among the aspects and sequences. Figure 1 schematically diagrams the aspect network. Several (possibly partially overlapping) representations of multiple objects are stored in the network. As a consequence of the graph representation, there is no a priori aspect which initiates or terminates a sequence. Any observed sequence provides evidence for its parent graph. When attempting to recognize an object from a sequence, multiple running hypotheses are maintained. Any sequence is recognizable in either the forward or reverse directions. We seek to exploit the transitions between the aspects by detecting "*trajectories*" of experience through the aspect network. Thus, we are able to recognize objects using views which may be statically ambiguous and noisy.

A set of simultaneous first-order differential equations (motivated by cell membrane dynamics) governs the activity of the nodes in the network, as well as the slowly varying synaptic weights (i.e., arcs of "variable strength") between the nodes. Our dynamical equations below are based on principles of neurodynamics and learning developed over the past twenty years by Grossberg.

Input is provided by the I-nodes, which are the output nodes of the F_2 (category) layer of an ART 2 network used to categorize 2D-invariant views in (Seibert & Waxman, 1989a). As a result of competition within the F_2 layer, only a single view is active whenever the ART network is in resonance. Any view category could potentially be an aspect for any object during the initial learning phase, so each I-node is connected widely across the aspect network to potential aspect graphs, although sparsely to allow other I-nodes connections also. The input from node I_l (either 0 or 1) to node x_i is gated by a nonadaptive connection J_{li}:

$$J_{li} = \begin{cases} 1.0 & \text{if } I_l \text{ is connected to } x_i \\ 0.0 & \text{otherwise.} \end{cases}$$

[1] This work was supported by the Department of the Air Force. The views expressed are those of the authors and do not reflect the official policy or position of the U.S. Government.

[2] The 2D neural system learns representations of 2D shapes that are invariant to position, orientation, scale, foreshortening, and small deformations (due to perspective).

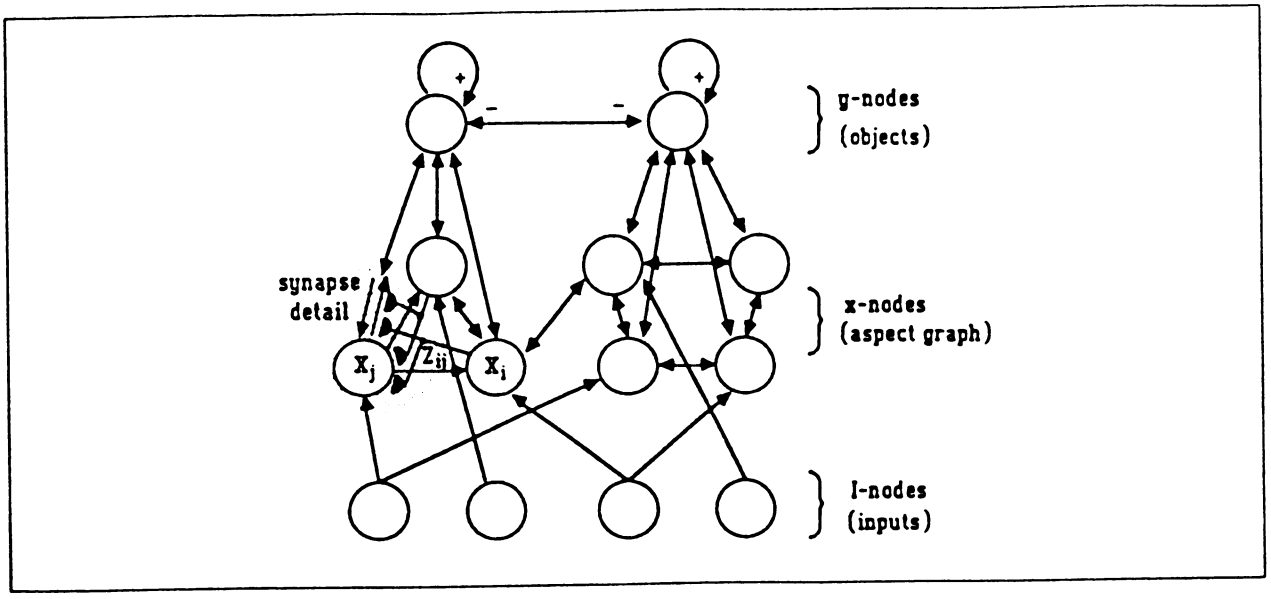


Figure 1: *Aspect Network.* The learned graph representations are realized in the x-node layer. "*Trajectories*" through the graphs are detected by the y-nodes. (To simplify labeling, only a few interconnects are drawn.)

The x-nodes actually form the aspect graphs: each node represents a particular 2D invariant view while the (variable strength) arcs code the learned relationships (i.e., transitions) among the views. The rates-of-change of the x-nodes are given by a set of coupled, first-order differential equations:

$$\frac{dx_i}{dt} = (1 - x_i)\left[I_l J_{li} + \sum_{j \neq i} Z_{ji}^{+} h(x_j)\right] - x_i\left[\mu + \sum_{j \neq i} Z_{ji}^{-} h(x_j)\right]. \tag{1}$$

In (1), Z_{ji} denotes the synaptic strength (i.e., possibility of a transition) from node x_j to node x_i and the superscript indicates either excitatory (+) or inhibitory (−) connections. A passive decay of activity is provided with rate μ. The signal function $h(\cdot)$ can be used for noise suppression (i.e., sigmoid or threshold-linear), but we have not experimented with anything other than linear $h(\cdot)$ functions. Equation (1) elaborates Grossberg's (1973) shunted short-term memory model. The first term is shunted by $(1-x_i)$, which approaches 0.0 as x_i approaches 1.0. This serves as an automatic gain control on the excitatory terms, keeping the value of x_i less than 1.0. Similarly, the inhibitory and decay terms are multiplied by x_i so that the value of x_i cannot decrease below 0.0.

The excitatory inter-x-node weights, Z_{ji}^{+}, serve to subliminally and laterally prime possible adjacent views (since they are correlated by experience), as well as help in the detection of views which legally follow predecessor views. The inhibitory inter-x-node weights, Z_{ji}^{-}, discourage these events which are decorrelated by experience. Both excitatory and inhibitory weights evolve from identical initial conditions to reflect the experience of the network as it observes 2D views of 3D objects in 3D motion. Each Z_{ij} becomes either excitatory (i.e., positive) or inhibitory (i.e., negative) and therefore contributes to either the first or second summation in (1). (For inhibitory weights, $Z_{ij}^{-} \equiv -Z_{ij} > 0$.)

The equation governing the inter-x-node weights uses a modified Hebbian learning rule. Inspiration for weight adaptation via (2) comes from competitive *in-star* learning which was proposed by Carpenter and Grossberg (1987a, b) as part of Adaptive Resonance Theory. The rates of change of the weights are:

$$\frac{dZ_{ij}}{dt} = \kappa f(x_j)\left[(1 - Z_{ij})\,\alpha\, h(x_i) - (1 + Z_{ij})\,\beta \sum_{k \neq i,j} h(x_k)\right]. \tag{2}$$

In (2), $\kappa \ll 1$ controls the speed of evolution of the weights relative to the x-node dynamics, $\alpha > 1$ and $\beta < 1$ balance the importance of Hebbian learning and competition, respectively. Weight changes are gated

by $f(x_j)$, a threshold-linear function of the post-synaptic activity x_j. The weights undergo competition according to $-\sum h(x_k)$, which serves to normalize the weights impinging on each x_j node. This synaptic competition reflects mass-action dynamics where axons compete for a limited number of synaptic sites on dendrites. The shunting terms $(1 - Z_{ij})$ and $(1 + Z_{ij})$ provide automatic gain control on the strengths of the weights (Grossberg, 1973), just as in (1), to keep the value of Z_{ij} in the interval from -1.0 to $+1.0$. Weights between x-nodes which are activated in sequence become excitatory, while other inter-x-node weights become inhibitory through competitive learning. *From an initially amorphous network, an aspect network crystallizes as the trajectories through it are repeatedly experienced.* A significant side effect is that the emerging representation is more general than the experiences which combined to create it, in the following manner. When two trajectories cross at a single node, many other trajectories are implicitly represented by taking the first half of one trajectory and branching into the second trajectory at the shared node. That is, once a view transition is learned, it contributes to any trajectory that incorporates it. Thus, view sequences which have never previously been experienced can still facilitate recognition of the appropriate object.

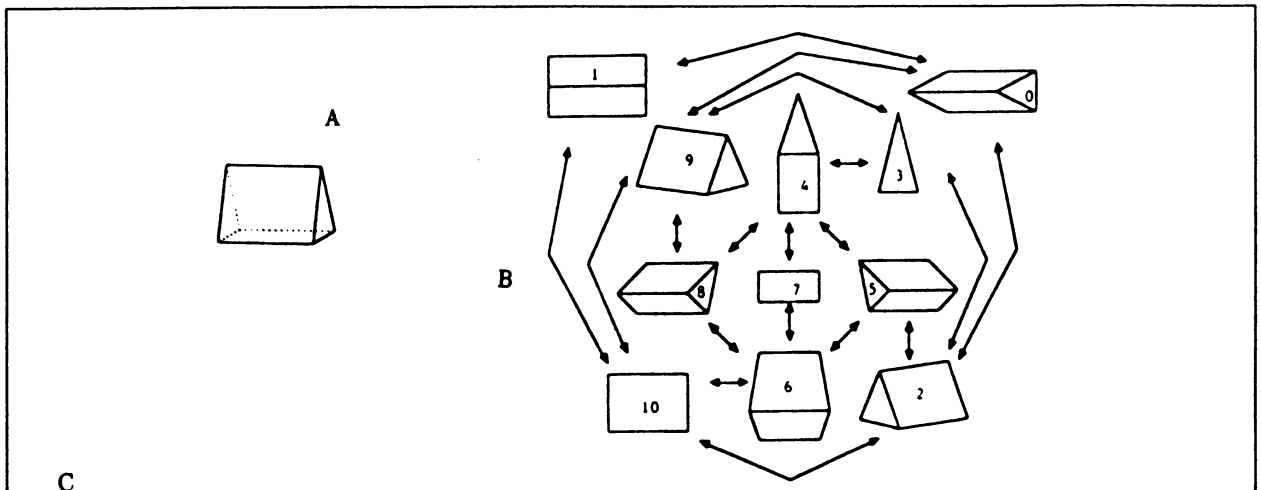
Competitive learning plays a key role in the evolution of the synaptic weights. Suppose the element x_1 is active, but begins to decay as x_2 becomes active due to the next view. The activity of x_1 does not decay to zero since it receives lateral priming from x_2, and indeed takes quite a while (depending on the setting of μ) to decay. While both nodes are semi-active, the weights $Z_{1,2}$ and $Z_{2,1}$ are both strengthened according to (2). The weight $Z_{1,2}$ is strengthened at the expense of all the other weights impinging on x_2, namely $Z_{k\neq 1,2}$, while the weight $Z_{2,1}$ is strengthened at the expense of $Z_{k\neq 2,1}$. All of these weight changes proceed simultaneously. Since $\beta < \alpha$, the reinforced weights do not impose a large penalty on the other weights (permitting many-way branches within the graph), but the cumulative effect of seldomly experienced view transitions is to drive the associated weights negative (and inhibitory), effectively halting excitatory lateral priming (anticipation) along the arcs between uncorrelated nodes.

Referring again to Figure 1, let us examine the general architecture. Each bottom–up input aspect activates widely dispersed nodes within the aspect network via randomly distributed, hardwired connections. This is necessary since any particular aspect may be a member of many different object representations. The time-sequences of aspects, through competitive learning, then build up local graph structures within the network. The activities of nodes in the network laterally prime one another (sequence predictions), as well as gate the flow of activity to the object classifying y-nodes. The y-nodes compete on a slow time-scale (relative to the x-node evolution) so that the maximally active y_k indicates the currently best object-hypothesis. The y-nodes also provide top-down, subliminal priming (expectations) to the x-nodes.

Example of Aspect Network Learning

We have recently described (seibert & Waxman, 1989b) the integration aspect network learning with the recognition system for 2D-invariant views. For example, a simple prismatic object (Figure 2A) might generate a graph such as Figure 2B when some symmetries are taken into consideration. This will be used to illustrate the learning of the aspect graph. The sequence of views presented to the system was: *reset* $\rightarrow 0 \rightarrow 2 \rightarrow 10 \rightarrow 1 \rightarrow 0 \rightarrow 9 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 4 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 6 \rightarrow 8 \rightarrow$ *reset* $\rightarrow 3 \rightarrow 2 \rightarrow 5$. The *reset* item indicates that all x-node activities were set to zero before the next view was presented, which is necessary when the next view is not a successor of the previous view, such as when attention is shifted due to camera motions. This sequence was presented as input to the aspect network 5 or 6 times. As a result of this presentation, each possible transition among views was experienced in at least one direction for the graph of Figure 2. The equations for the aspect network were implemented using 4-stage Runge–Kutta and predictor–corrector coupled differential equation solvers. We systematically investigated the effects of perturbation on the equation parameters in terms of robust organization, recognition, network size, and stability.

Figure 2C shows the matrix of evolved connection weights for the aspect graph of Figure 2, after five complete cycles through a sequence of views. The parameter settings were as follows: $Z_{ij}(0) = 0.01$, $\alpha = 3.0$, $\kappa = 0.01$, $\beta = 0.5$, $\mu = 0.5$, and the threshold for the threshold-linear function $f(\cdot)$ was 0.3. No top–down priming was used in this example. There are no weights Z_{ii} from x_i to x_i: the bracketed diagonal elements are used to display the current activity values of the x-nodes. In this case, x_5 has the highest value since it corresponds to the most recent view of the prism.



	0	1	2	3	4	5	6	7	8	9	10
0	‹0.0337›	0.0517	0.0707	-.0178	-.0660	-.0453	-.0660	-.0283	-.0552	0.0593	-.0317
1	0.0893	‹0.0134›	-.0495	-.0390	-.0707	-.0552	-.0495	-.0281	-.0521	-.0414	0.0286
2	0.0521	-.0027	‹0.1202›	0.0424	-.0564	0.0571	-.0583	-.0264	-.0538	-.0464	0.0548
3	-.0393	-.0261	0.0659	‹0.0424›	0.0611	-.0078	-.0592	-.0251	-.0418	0.0290	-.0421
4	-.0453	-.0293	-.0466	0.0425	‹0.0454›	0.0725	-.0426	0.0483	0.0677	-.0461	-.0670
5	-.0416	-.0249	0.0186	-.0397	0.0292	‹0.6710›	0.0556	-.0002	-.0275	-.0627	-.0586
6	-.0414	-.0179	-.0619	-.0459	-.0417	0.0249	‹0.0644›	0.0957	0.0607	-.0638	0.0048
7	-.0439	-.0279	-.0636	-.0444	0.0624	-.0418	0.0325	‹0.0126›	-.0082	-.0644	-.0695
8	-.0441	-.0286	-.0633	-.0381	0.0218	-.0399	0.0190	-.0102	‹0.0128›	0.0531	-.0487
9	0.0406	-.0234	-.0434	0.0816	-.0454	-.0490	-.0490	-.0267	0.0238	‹0.0145›	0.0493
10	-.0167	0.0919	0.0226	-.0331	-.0673	-.0429	0.0436	-.0268	-.0392	0.0103	‹0.0352›

Figure 2: A prism (A) could be represented by the graph in (B) when certain symmetries are considered. The learned weights are tabulated as the diagonal elements in (C).

Note that the connections $Z_{0,1}$ and $Z_{1,0}$ were both learned, even though the transition $0 \rightarrow 1$ was never experienced. This is the case in general: the weight matrix is qualitatively symmetric. The quantitaive asymmetry is due to the effects of different competitions for post-synaptic receptive sites. That is, the competition tends to normalize the weights in each column of the connection matrix according to (2). This has the convenient effect of lessening the priming of anticipated views when there is a large number of possible successor views (and thus less certainty).

In addition to vision, such learned graph representations are applicable to speech recognition, simulating finite state automata, representing semantic networks, music recognition and improvisation, and planning.

Carpenter, G. A., & Grossberg, S. (1987a). ART 2: Self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, **26**(23), 4919-4930.

Carpenter, G. A., & Grossberg, S. (1987b). A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*, **38**, 54-115.

Grossberg, S. (1973). Contour enhancement, short term memory, and constancies in reverberating neural networks. *Studies in Applied Mathematics*, **52**(3), 217-257.

Grossberg, S. (1982). **Studies of mind and brain: Neural principles of learning, perception, development, cognition, and motor control.** Boston: Reidel Press.

Koenderink, J. J., & van Doorn, A. J. (1979). The internal representation of solid shape with respect to vision. *Biological Cybernetics*, **32**, 211-216.

Seibert, M., & Waxman, A. M. (1989a). Spreading activation layers, visual saccades, and invariant representations for neural pattern recognition systems. *Neural Networks*, **2**, 9-27.

Seibert, M., & Waxman, A. M. (1989b). Learning Aspect Graphs from View Sequences in a Neuromorphic System. Submitted to the *1989 IEEE Workshop on Interpretation of 3D Scenes.*

Training Continuous Speech Linguistic Decoding Parameters as a Single-Layer Perceptron

Mark T. Anikst and David J. Trawick
Speech Systems Incorporated
18356 Oxnard Street
Tarzana, California 91356

Abstract

In this paper we present a provably convergent variant of the corrective training method (see [Bahl 88]) for estimating the scoring parameters of a linguistic decoder for speech recognition. This method does not rely on the treatment of the scoring model as a hidden Markov model (HMM), which is viewed here as unnecessarily limiting the values of the scores to those of transition or output probabilities. Our variant of the corrective training can be characterized as a direct application of the LMS algorithm to single-layer perceptron training (see [Lippmann 87]). As such, when the target labelling is kept fixed, it possesses the known stochastic convergence property of the LMS algorithm. This variant of corrective training can also be applied to the HMM-based scoring model if we abandon the interpretation of that model's components as probabilities. We also include a more natural method for finding near-misses for the algorithm, by using the near-misses generated during the speech decoding process.

1. Introduction

Many current speech recognition systems use hidden Markov models, which are typically trained using the Baum-Welch (or forward-backward) algorithm [Baum 72]. An alternative HMM training method has been introduced by Bahl, *et al.* [Bahl 88], called corrective training, which yielded a 16% reduction in the number of word errors on their 2000- or 5000-word speaker-dependent, isolated-word office correspondence task. Lee *et al.* [Lee 89] extended this method to speaker-independent continuous-speech, which yielded a 20% reduction in word error rate on a (no grammar) 997-word DARPA resource management task. Both of these methods rely on the Baum-Welch algorithm for reestimating probabilities as an underlying step of the training process.

Here we describe a similar algorithm that we have used for training our speech decoding scoring parameters. However, we have not restricted ourselves to continually recasting the parameters of the model as probabilities, nor are we dependent on an underlying Baum-Welch algorithm. This allows for a direct application of training methods for a single-layer perceptron, and allows us to optimize directly for improved recognition performance.

Our decoding algorithm is a version of a dynamic programming beam search [Lowerre 80]. As such, it keeps alternatives when decoding. After decoding, several alternatives remain that were not chosen as the output. These alternatives are used in our training algorithm as a natural choice for near-misses, making reinforcement training easy. In the case of an error in decoding, we can furthermore identify where the decoder went wrong and concentrate our training at that point.

In section 2, we describe our speech encoding and decoding model, particularly as it relates to the training algorithm. Section 3 describes the training algorithm, and section 4 gives the results along with some discussion of an application of the algorithm on a speaker-independent continuous-speech digits task.

2. Speech Encoding and Decoding Model

2.1 Speech Segmentation and Coding

A continuous-speech utterance is first quantized by the acoustic front-end into a time-sequence of acoustic frames. Next, the acoustic frames are coded by a Frame Encoder (neural-network based) trained to maximize the average mutual information between its code alphabet and the alphabet of the broad phonetic classes (which are used for frame segmentation). The time-sequence of coded acoustic frames is then processed by a Segmenter which forms acoustic segments by merging some time-contigous blocks of frames using their codes and the code/broad phonetic class statistics to make segmentation decisions. Finally, the resulting acoustic segments are coded by a Segment Encoder (also neural-network based) trained to maximize the mutual information between its code alphabet and the alphabet of the dictionary phonetic classes (which are used for transcribing words in the dictionary). We denote the output time-sequence of segment-codes by $S = \{code(i), i = 0,\ldots,I\}$ where each $code(i)$ belongs to the code alphabet (or codebook) of the Segment Encoder.

2.2 Language Model

A language model is defined by a dictionary and a grammar. Each word in the dictionary is represented by a directed graph with nodes associated with certain phonetic classes. Multiple paths in the word graph account for the phonetic variations of the word.

The grammar is a directed graph with nodes associated with certain word categories. Paths in the grammar graph correspond to sentences allowed by the grammar.

Expanding each node of the grammar graph with the word graphs of all the words found in the node's word category yields a (directed) graph which we call a sentence transcription graph. Each path of the latter graph represents an allowable transcription (in the phonetic classes) of some sentence. We denote such a transcription by $\{class(j), j = 0,\ldots,J\}$, where each $class(j)$ belongs to the set of phonetic classes.

2.3 Speech Decoder Model

To find the spoken sentence which produced the sequence of segment codes S = {*code*(i), i = 0,...,I}, we search the sentence transcription graph for a transcription T = {*class*(j), j = 0,...,J} which maximizes the score of the (best) alignment between S and T. The sentence text corresponding to this transcription is then chosen as the output.

The alignment between the segment code sequence S and the sentence transcription T includes time alignment between elements of S and T. Possible time alignments are restricted to those in which a subsequence of codes {*code*(i), i = i(0),...,i(1)}, i(0) ≤ i(1), can be time-aligned with a single class *class*(j); conversely, a subsequence of classes {*class*(j), j = j(0),...,j(1)}, j(0) ≤ j(1), can be time aligned with a single code *code*(i).

A typical alignment is illustrated in Fig. 1., where *code*(0) is aligned with *class*(0); *class*(1) is aligned with {*code*(1), *code*(2)}; *code*(3) is aligned with {*class*(2), *class*(3), *class*(4)}; *class*(4) (the last class of the previous subsequence) is also aligned with *code*(4); *code*(4) (the last code of the previous subsequence) is also aligned with *class*(5); and finally, *class*(5) is also aligned with *code*(5).

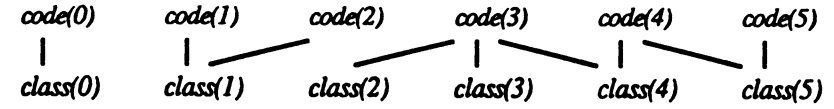


Figure 1.

2.4 Alignment Score Computation

To score an alignment between the segment code sequence S and the sentence transcription T, we first assign score values to the time-aligned subsequences of S and T, and then take the sum of these score values. To score the time-aligned subsequences, we account for (a) each alignment between a code and a class, (b) the number of successive classes aligned with the same code, and (c) the number of successive codes aligned with the same class. To express the score of an alignment, we introduce the alignment scoring model comprised of three components (one for each of the above cases):

(1a) p(*code*, *class*) -- the score for aligning *code* with *class*;
(1b) q(*code*, *numclass*) -- the score for aligning *code* with *numclass* successive classes;
(1c) r(*class*, *numcode*) -- the score for aligning *class* with *numcode* successive codes.

The score of an entire utterance is simply the sum of the scores of the time-aligned subsequences of the entire utterance. To illustrate the score computation, we give the score for the alignment of Fig. 1:

score(alignment Fig. 1) = p(*code*(0),*class*(0)) +q(*code*(0),1) +r(*class*(0),1)
+p(*code*(1),*class*(1)) +p(*code*(2),*class*(1)) +q(*code*(1),1) +q(*code*(2),1) +r(*class*(1),2)
+p(*code*(3),*class*(2)) +p(*code*(3),*class*(3)) +p(*code*(3),*class*(4)) +q(*code*(3),3) +r(*class*(2),1) +r(*class*(3),1) +r(*class*(4),2)
+p(*code*(4),*class*(4)) +q(*code*(4),2) +p(*code*(4),*class*(5)) +r(*class*(5),2) +p(*code*(5),*class*(5)) +q(*code*(5),1)

We will now rewrite the scoring formula into a form suitable for single-layer perceptron training. Since the score components of the aligned subsequences of the codes and classes are summed to obtain the alignment score, we can group the terms of that sum in the following way:

$$(2)\ \text{score(alignment)} = \sum_{\text{all } code,\ class} \sum p(code,class) \cdot x(code,class)$$
$$+ \sum_{\text{all } code,\ numclass} \sum q(code,numclass) \cdot y(code,numclass) \quad + \sum_{\text{all } class,numcode} \sum r(class,numcode) \cdot z(class,numcode)$$

where x(*code*,*class*), y(*code*,*numclass*), and z(*class*,*numcode*) are the number of times the given pairs (*code*,*class*), (*code*,*numclass*), and (*class*,*numcode*) occured in the alignment, respectively. If we introduce the vectors P, Q, and R corresponding to the p, q, and r scoring components respectively, and the vectors X, Y, and Z corresponding to the x, y, and z alignment components respectively, we may rewrite (2) as a sum of the dot product of these vectors:

(3) score(alignment) = P * X + Q * Y + R * Z

Finally, concatenating P, Q, and R into a (block-) vector L, and concatenating X, Y, and Z into a (block-) vector U, we obtain the formula for the score of an alignment as the dot product of L and U:

(4) score(alignment) = L * U.

Note that L has all the scoring parameters of the decoding model, which are the parameters that will be subject to training, while U = U(alignment) has all the essentials of the alignment as far as the scoring formula is concerned.

2.5 Decoding Method

To convert the segment-code sequence into the sentence text, we use a version of the dynamic programming beam search method. An utterance is decoded left-to-right by progressively aligning its segment-code sequence with the transcriptions found in the sentence transcription graph. A set of prefix alignments is maintained by the DP beam-search algorithm, consisting of all the prefix or complete sentence transcriptions aligned with the prefix segment-code sequence. The prefix alignments are successively expanded each time the next code is added to the segment-code sequence. The set of prefix alignments is pruned by retaining only the top scoring alignments. Only those alignments within a relative scoring threshold of the best scoring alignment are kept.

After the decoding process reaches the end of the segment-code sequence, the complete sentence transcription yielding the highest scoring alignment with the complete segment-code sequence is found. The text corresponding to this transcription is the chosen output. If no complete sentence transcription exists among the aligned transcriptions remaining on the beam, the utterance is rejected (a decoding failure). For more details see Meisel, *et al.* [Meisel 88].

3. Training Method

For the purpose of training, we designate the target alignment and complete sentence transcription for each utterance of the training set. Our task is to train the alignment scoring model (2) so that the score of the target alignment is greater (by some threshold) than the score of any incorrect alignment (aligned with the same prefix segment-code sequence) remaining in the beam structure after each beam-pruning step. An incorrect alignment is defined as any alignment of any prefix or complete transcription of any incorrect sentence. Thus if RA is the target alignment, and WA is any incorrect alignment remaining in the beams after the beam-pruning step, we want:

(5) score(RA) - score(WA) $\geq$ h, (h > 0, a fixed margin) for all WA.

Replacing these scores and alignments as in (4), yields

(6) L * U(RA) - L * U(WA) = L * (U(RA) - U(WA)) $\geq$ h.

If we define the set of vectors $V = \{V : V = U(RA) - U(WA)\}$, we can formulate the scoring model training task as the training of the single-layer perceptron with weight vector L to perform the separation of the set V:

(7) L * V $\geq$ h, for all V in V.

This task can be approached with a number of the single-layer perceptron training algorithms (see [Lippmann 87]). We selected the LMS algorithm with the linear threshold non-linearity:

$t(d) = 0$ for $d < 0$, $t(d) = d/h$ for $0 \leq d < h$, and $t(d) = 1$ for $d \geq h$

An algorithm to train the scoring parameters L is then:

(8) $L' = L + (1 - t(L * V)) \cdot \beta \cdot V$

where L' is the updated L, V is some vector of V, and β is the training rate ($0 < \beta \leq 1$). The effect of this update is to adjust the scores so that the target alignment's score is increased while the incorrect alignment's score is decreased.

The update step (8) is performed for all vectors V of V. For the next utterance of the training set, a new set V is constructed. One iteration through the training data consists of processing each utterance in this manner, updating L each time.

3.1 Generating Target Alignments

The target alignments necessary for training are generated automatically by a special mode of the decoding process. In this mode the sentence transcription graph is restricted to transcriptions of the known sentence. Then a nearly exhaustive search is performed on the alignments of that graph with the segment-code sequence for that utterance. The highest scoring alignment/transcription is chosen as the target alignment. (Specifics of the alignment/transcription including the score, class sequence, and class-code alignment are retrieved from the remaining beam structures after decoding is complete.)

This automatic target alignment generation step is performed just prior to performing the parameter training with that utterance, using the scoring parameters being trained to do the decoding. This constitutes a modification of the standard LMS algorithm which precludes a straightfoward extension of the proof of convergence. Nevertheless, we found that it resulted in the stochastic convergence in all the cases tested so far. Note that if we could select the target alignment *a priori* and use it in all training iterations, then convergence is guaranteed for a large enough number of iterations. In our modification, it is possible that the target alignment will never settle into a steady state, and thus the scoring parameters would not converge.

The values of the vector L, the scoring parameters of the decoder, are initialized with a heuristic estimate before any training is performed. The estimate is made from a large pool of phonetically labelled speech data. (The data are labelled with the phonetic classes of the dictionary.)

3.2 Selecting Incorrect Alignments: Preventive Maintenance Training

Several possible incorrect alignments remain in the beam structures after the decoding process, as well as many prefix alignments. By using these remaining incorrect alignments we focus on the errors actually made by the scoring algorithm. This addresses the problem more directly than generating near-misses, as has been done in other approaches [Bahl 88, Lee 89].

Due to the left-to-right nature of the decoder, we can determine the point in time (the segment-code sequence index) where the target alignment was lost by being pruned from the beams. We can thus focus the training on that area where the chosen alignment lost the possibility of attaining the target alignment, by training with the prefix target alignment and the corresponding prefix incorrect alignment(s). These prefix alignments are treated essentially like complete alignments by the training algorithm. Training is performed on each whole word prefix alignment until the target alignment is pruned off the beams (if it is), at which point we stop the training for that utterance.

We call this training method "preventive maintenance training" since it attempts to prevent the score of each prefix target alignment from dropping too low. It thus attempts to prevent each prefix target alignment from being

pruned from the beams, as well as attempting to make (eventually) the complete target alignment the highest scoring complete alignment.

4. Results and Discussion

An experiment was run to investigate the performance of the new algorithm. For this experiment, a digits application was used. The grammar for this application accepts all sentences containing the words "zero" through "nine" and "oh" in any combination and of any word length. The word "point" may also occur anywhere in an accepted sentence, but only once.

A set of 3017 digits utterances collected from eleven male speakers were used for the training process. Another 800 digits utterances collected from eight of the original eleven were reserved for testing purposes. The average number of words in a training (test) utterance was 4 (4). Baseline results were obtained using the original scoring parameter estimations, generated from large-vocabulary (1526 word) training data. These will be indicated as iteration 0 in the following tables. For this test, a training rate of $\beta = 1.0$ and a training threshold of $h = 5.0$ were used. Some brief experimentation showed that performance was satisfactory at these values.

Table 1 shows the change in performance on the test data for the six iterations that were made. Table 2 shows the change in performance on the training data for the same six iterations. The utterance accuracy refers to utterances that are exactly correct; no deletions, substitutions, or insertions. The "+ Off 1" statistics refer to the percentage of utterances with at most one error (one insertion, one deletion, or one substitution only). The "+ Off 2" statistics refer to the percentage of utterances with at most two errors.

Iteration	Utterance Accuracy	+ Off 1	+ Off 2
0	45.25	72.00	86.25
1	61.75	85.25	95.38
2	59.38	86.00	94.88
3	64.63	87.75	95.62
4	68.75	89.12	96.62
5	67.50	88.50	96.00
6	69.13	89.38	96.50

Table 1.Test Data

Iteration	Utterance Accuracy	+ Off 1	+ Off 2
0	56.25	84.22	94.66
1	72.69	92.38	98.08
2	72.92	93.80	98.54
3	79.22	94.93	98.77
4	85.58	97.18	99.57
5	86.84	97.32	99.44
6	87.04	97.12	99.47

Table 2.Training Data

At the end of six iterations, the utterance recognition rate on test data has increased from about 45% to 70%, for a reduction in utterance error rate of about 55%. The recognition rate on training data has increased from about 56% to 87%, for a reduction in utterance error rate of about 30%.

5. Conclusions

A new training algorithm for continuous speech linguistic decoding parameters has been presented and tested successfully. This new algorithm has several novel aspects, which we believe are advantages over other tested corrective training approaches [Bahl 88, Lee 89]. In particular, it:

a. Deals with an alignment-scoring model more general than HMM (does not rely on the interpretation of the model components as probabilities);

b. Is based on the LMS algorithm directly, for which the property of stochastic convergence holds;

c. Selects confusible alignments (near-misses) from the incorrect sentences remaining on the beams, which were actual confusions the decoder encountered;

d. Trains using prefix target alignments, so that training emphasizes survivability of target alignments throughout the left-to-right processing of the decoder.

Acknowledgements

The authors would like to thank the rest of the team at Speech Systems for their support and contributions.

References

L.R. Bahl, P.F. Brown, P.V. deSouza, and R.L. Mercer. "A New Algorithm for the Estimation of Hidden Markov Model Parameters," in *Proceedings of the 1988 IEEE ICASSP*, pp. 493-496, April 1988.

L.E. Baum. "An Inequality and Associated Maximization Technique in Statistical Estimation of Probabilistic Functions of Markov Processes," in *Inequalities*, 3:1-8, 1972.

K.F. Lee, and S. Mahajan. "Corrective and Reinforcement Learning for Speaker-Independent Continuous Speech Recognition," Technical Report CMU-CS-89-100, Carnegie Mellon University, January 1989.

R.P. Lippmann. "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, pp. 4-22, April 1987.

B. Lowerre, and R. Reddy. "The Harpy Speech Understanding System," in *Trends in Speech Recognition*, W.A. Lea, ed., pp. 15.1-15.9, Academic Press, 1980.

W.S. Meisel, P.C. Shinn, D.J. Trawick, and W.A. Wittenstein. "Speech Representation and Speech Understanding", Final report under DARPA Contract DAAH01-87-C-0953, April 1988.

Neural Network Based Data Compression Using Scene Quantization

Mohammed Arozullah and Aran Namphol
Department of Electrical Engineering
The Catholic University of America
Washington, DC 20064

Abstract

A neural network based data compression system using a technique called *scene quantization* is presented. In scene quantization a whole scene of information is quantized at a time and is compressed to a single representation for low rate transmission and subsequent recovery of the original scene. The proposed network model is a 4-layer feed forward architecture with back propagation learning paradigms. Simulation results for the compression transmission and recovery of a number of scenes which are members of the training set or out side the training set are presented.

1. Introduction

This paper presents the results of an investigation in the use of neural networks to realize data compression using scene quantization. Scene quantization differs from vector or scalar quantization in the sense that it allows simultaneous parallel-processing of a whole scene of information whereas vector or scalar quantization only quantize one vector or one point respectively at a time. Therefore, with scene quatization technique one scene of information can be compressed to one representation. Thus, the transmission rate, time and number of parallel links for transmission, and storage requirements are much less than those required by scalar or vector quantization for the same scene.

2. Data Compression Using Scene Quantization

2.1 The Proposed Neural Network Architecture

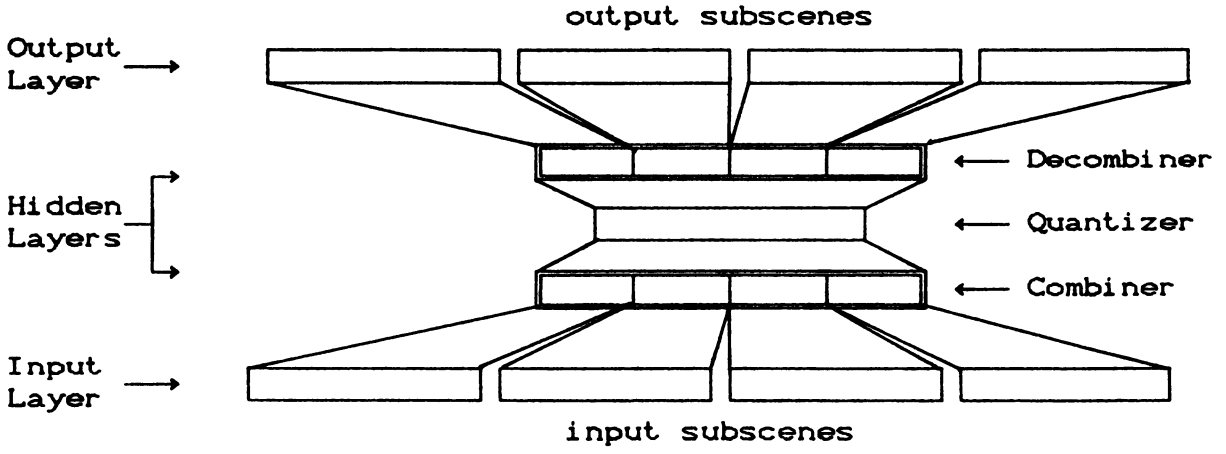


Fig. 1 The Proposed Neural Network Based Data Compression System

As shown in Fig. 1, the system consists of three sections: input-layer section, hidden-layer section, and output-layer section. Input-layer section consists of a number of subnets which serve as input layer to the network. The hidden-layer section consists of three parts: combiner, quantizer, and decombiner. The combiner acts as the multiplexer to the input subnets. The quantizer produces the internal representation of the scene. Finally, the decombiner acts as the demultiplexer. A preprocessor may also be used, but is not shown here.

If the network consists of m input-subnets, and m output-subnets and each subnet has k nodes and a sigmoid nonlinear tranformation is applied to the activation level of each node, then,

At the combiner: $C_{ni} = f\left[\sum_{j=1}^{k} w_{nij} x_{nj} \right]$, where

$C_{ni} \cong$ ouptut of i^{th} combiner node corresponds to n^{th} subnet

$x_{nj} \cong$ output of j^{th} node of n^{th} subnet

$w_{nij} \cong$ weight between the j^{th} node of the n^{th} subnet to the i^{th} node of the combiner.

At the quantizer: $q_i = f\left[\sum_{n=1}^{m} \left(\sum_{j=1}^{l} w_{nij} C_{nj} \right) \right]$, where

$q_i \cong$ output of i^{th} node of the quantizer

At the decombiner: $d_{ni} = f\left[\sum_{j=1}^{p} w_{nij} q_j \right]$, where

$d_{ni} \cong$ output of i^{th} node corresponding to n^{th} subnet

$p \cong$ number of nodes of the quantizer

At the output: $y_{ni} = f\left[\sum_{j=1}^{r} w_{nij} q_j \right]$

The distortion measurement is the mean square error between the output scene and the input scene. Therefore,

$$MSE = \sum_{n=1}^{m} \sum_{i=1}^{k} \left(x_{ni} - y_{ni} \right)^2$$

The overall desired outputs are the same as the corresponding inputs.

2.2 A Simulation Model of the Network

The proposed neural networtk model for scene quantization was simulated on an IBM AT using the Professional II software package of NeuralWare, Inc.[3]. Due to the limitation of the software tool, a scene of 12x12 pixels was used as the input scene. The

number of nodes in the quantizer was 5 or 6.

As shown in Fig. 1, the network model is divided into three sections: input-layer, hidden-layer, and output-layer sections. Futhermore, the hidden-layer section consists of three layers, namely combiner, quantizer, and decombiner. The overall network model is symmetric. One can take advantage of this symmetry in the training process as follows: the network was trained inside out, i.e. the hidden layers were trained first, thenthe outer layers were trained using those previous computed weights for the hidden layers.

3. Simulation Results

Shown in Figure 2 are the examples of recovered output scenes which have been quantized and reconstructed from input scenes. Table 1 shows the distortion, MSE, over trained and untrained input scenes with different number of hidden nodes.

Scene	Hidden Nodes	Compression Ratio	MSE	Type
a	5	28.80	0.0380	Trained
b	6	24.00	0.0270	Trained
c	6	24.00	0.1285	Untrained

Table 1 : Compression Ratio and MSE

4. Conclusion

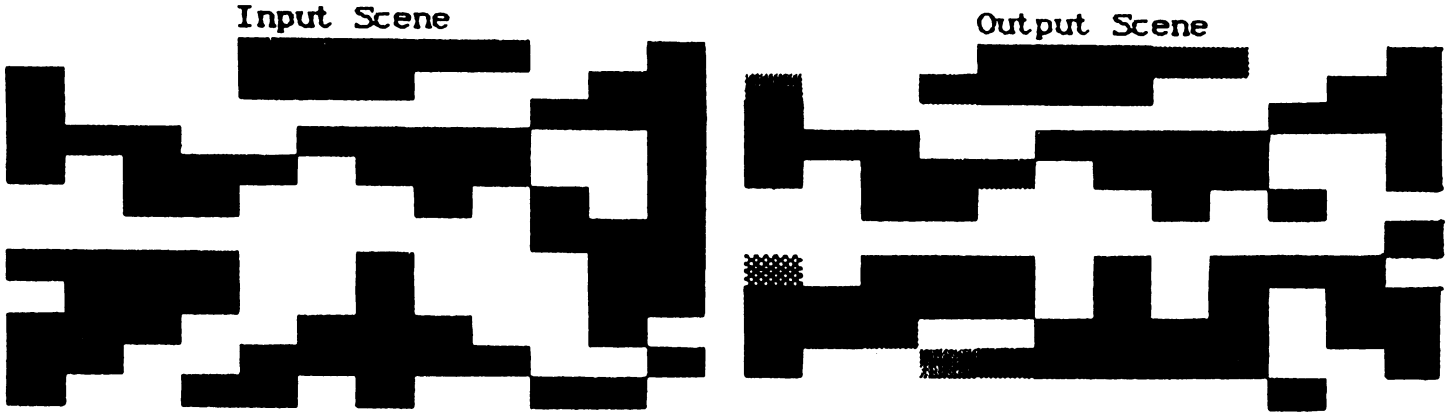
It is observed from the simulation results that the network worked with reasonable error and hence the scene quantization provides a viable method for compression of data scenes.

5. References

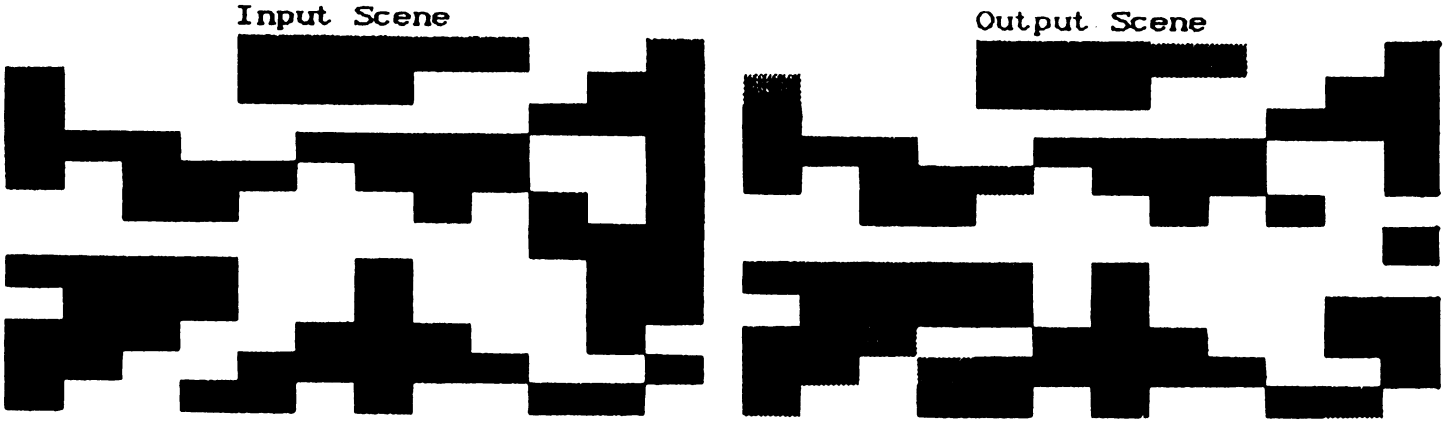
[1] Garrison Cottrell, Paul Munro, and David Ziper, "Learning Internal Representation from Gray-Scale Images: An Example of External Programming," The Ninth Conference of Cognitive Science Society, 1988.

[2] N. Sonehara, M. Kawato, S. Mitake, and K. Nakane, "Image Data Compression Using a Neural Network Model", Proceeding of IJCNN, Washington DC, June 1989.

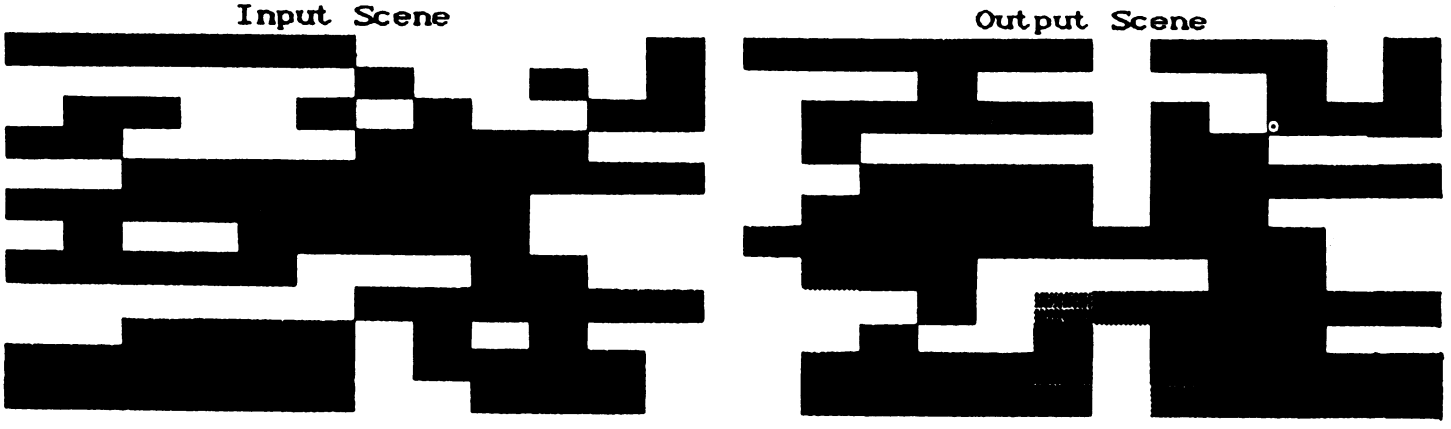
[3] Neural Ware Inc., "Users Handbook for Neuralworks Professional II", Version 2.0, 1988.



(a) shows reconstructed scene from trained data with 5 quantizer nodes.



(b) shows reconstructed scene from trained data with 6 quantizer nodes.



(c) shows reconstructed scene from untrained data with 6 quantizer nodes.

Figure 2 - Examples of Simulation Results.

A Preliminary note on training static and recurrent neural networks for word-level speech recognition

Kamil A. Grajski, Dan P. Witmer and Carson Chen.

Ford Aerospace
Advanced Development Department/Mail Stop X-22
San Jose, CA 95161-9041
kamil@wdl1.fac.ford.com

The temporal nature of speech signals provides strong constraints on the neural network (NN) solution design process for automatic speech recognition (ASR) at the word-level. In this report we present initial experiments comparing the performance of several static and recurrent back-propagation (BP) trained networks on a spoken English word-recognition task. We conclude that judiciously applied, a recurrent neural network solution aimed directly at the word-level may be appropriate for certain ASR problems.

Methodology

Speech data and preprocessing

The ASR problem we study here is a simple, yet indicative, spoken English word-recognition task. Neural nets are trained to classify isolated utterances of the digits "0" through "9" and control words "up", "down", "left", "right", "pitch", "roll" and "yaw" recorded from three male-speakers. Five sets (I-V) are recorded from each speaker over a several week period. Recording is done in ambient noise conditions at 10K A/D samples per second, low-pass filtered at 3.3kHz cutoff. Preprocessing of recorded speech yields input data for NN training. First, selection by hand yields 0.5 second segments containing a single word or "noise." FFT analysis (with Hamming window; linear pre-emphasis in the frequency domain) in a 25 msec moving time-window (2/3 overlap between successive windows) transforms each utterance into a 3D image, e.g. a frequency spectrum at each of 60 times in a 0.5 second segment. Second, NN input is obtained by transforming each individual spectrum to a 15-D vector by integrating the spectral amplitudes centered at the following frequencies in Hz (bandwidth indicated in parentheses): 130 (30), 164 (38), 206 (48), 260 (60), 327 (76), 412 (96), 520 (121), 655 (152), 828 (192), 1040 (242), 1310 (305), 1650 (384), 2078 (485), 2619 (611), 3300 (770). Each 15-vector is optionally peak-, vector- or not normalized. Four of the five word sets are used for NN training; the fifth serves as an independent test set.

Network architectures

We systematically study several static and recurrent NN trained with the BP algorithm [1,2]. Static nets with one or two hidden layers are trained by presenting the entire 0.5 second spectrograph as a 3D "image" on the input layer, e.g. 60 x 15 input units. The single-hidden layer nets use 20 hidden units; double hidden-layer nets use 15 units per layer. For all nets, the output layer contains 17 output nodes (one ON per word); noise vectors to turn all output units OFF.

Our recurrent net is an extension of that reported by Elman [2] and analyzed by Servan-Schreiber [3]. The extension is simply an implementation of the nested equation describing recurrence of a hidden unit's output value. For the case of a two-step time-delay, the equation dictates the number and relationship of local memory cells associated with each hidden unit. The equation generalizes to arbitrary time-delays by further nesting.

$$y_j^t = f\left(\sum_{i=0}^{i=n} y_i^t w_{ji} + w_j^* f\left(\sum_{i=0}^{i=n} y_i^{t-1} w_{ji} + w_j^* y_j^{t-2}\right)\right)$$

y_j^t - hidden unit output, f(x) - unit transfer function, y_i^t - input unit output, w_{ji} - input to hidden unit connection, w_j^* - the recurrent connection, n - number of input units (includes bias).

The recurrent net allows time-sampling of the input spectrogram without the necessity for multiple copies of time-shifted inputs; this is well-suited for real-time applications. Our approach is to present, for each utterance, sequential, overlapping groups of spectra contained in that word's spectrogram. Exhaustive search suggests that for the present data, an optimal combination of grouping, overlap and time-delay stores hidden unit activation for the equivalent of up to 200 milliseconds preceding current input. The number of hidden units used in the single hidden layer case is twenty.

Training and testing

Training of all nets proceeds in an iterative fashion. For speaker-trained nets, training initially proceeds with set I alone using random weight strengths as initial conditions. Following convergence, set II is trained using as initial conditions the weight matrix generated at the previous step. This procedure is repeated until all four training sets have been pooled. The multiple-speaker case proceeds similarly, but with pooling of numbered sets at each iteration.

Our usage of the back-propagation algorithm is as follows: a.) errors are accumulated across the entire training set; b.) learning rates range between 0.05 - 0.001, depending on training set size and net type; c.) "viscosity" term is 0.9; d.) learning rate and viscosity terms are divided by 10 for the first 20 training cycles; e.) errors greater than 0.1 are propagated; and f.) calculation of error is as described by Solla [4]. Empirically, gradient descent has proven most useful. (Extensive experimentation with the conjugate gradient descent procedure coupled with simulated annealing is disappointing so far for anything but the smallest nets.) In the case of recurrent nets, a time-varying teacher signal has proven most successful. (See also [5].) In particular, for a given utterance, the selected output node is trained to peak at 1.0 in the range 250-350 milliseconds into the 500 msec utterance. Over the interval [0,250-350], the value rises linearly.

Trained nets are tested by classification performance on word set V. In the case of static nets, we define two types of classification score: a.) "raw" recognition; and b.) "fine-tuned" recognition. In raw recognition, the highest valued output unit greater than 0.5 denotes network response. Fine-tuned recognition is one approach to a problem inherent in processing each utterance as a single 3D image: time-alignment of repeated instances of an utterance. Noise spectra are appended to the beginning and end of the test spectrogram and the composite image is shifted sequentially through the window defined by the number of net input units. Each shift position yields a peak output value; the highest value (> 0.5) achieved across an empirically derived shift set (< 100 msec in either direction) denotes net recognition. In cases where the correct output node is signalling < 0.5, fine-tuning can often raise the value to > 0.9.

Testing performance of recurrent nets is more problematic. Since the input is time-varying so is the output. To detect classification, output layer activity is fed one-for-one into a continuously running mutually inhibitory net. This additional net (MAXOUT) signals recognition when one of its units achieves a value of unity, e.g. signals the corresponding most active recurrent net output unit across time.

Results

Table I summarizes network performance on our 17-word (plus noise tokens) database. "Raw" and "fine-tuned" recognition scores (max= 18/18) are listed separately for the one and two hidden layer (HL) static networks. For the one HL static case, fine-tuning the position of the spectrogram image on the input layer adds significantly to performance. For the two HL static case, raw recognition for two of three speakers is 100%; for the third, fine-tuning increases performance. The response values in the two HL case are robust, so that for speakers 2 and 3, a minimum output threshold value of 0.8 yields 100% and 88% raw recognition, respectively. On average, performance of the two static net types is quite close. Training time, measured by cycles through the learning set at each iteration, averages between 50-150 for all

nets; note that the one HL net contains > 19000 connections and the two HL case < 15000.

Table I

Network Performance on a 17-Word Test Set					
Speaker	Static (1 HL)		Static (2 HL)		Recurrent
	Raw	Fine-tuned	Raw	Fine-tuned	
1	61% (11)	78% (14)	44% (8)	67% (12)	83% (15)
2	78 (14)	100 (18)	100 (18)	100 (18)	88 (16)
3	94 (17)	94 (17)	94 (17)	100 (18)	94 (17)
Mean	78%	91%	79%	89%	88%
MS*	87% (27/31)	90%(28/31)	81% (25/31)	84% (26/31)	94% (29/31)

Results for 1 and 2 hidden layer (HL) static nets and a 1 HL recurrent net using a "home-grown" 17-word (plus noise) database (see text). (# correct in paren.) We are currently installing the TI/NBS Database. The recurrent net achieves competitive and more consistent results than the static nets. (*MS - multiple-speaker digit recognition "0" - "9" plus noise token only)

Table I (last column) lists performance of the single HL (20 units) 2 time-delay recurrent nets. (Initial studies suggest that a time-delay beyond 2 adds significantly to training time, with marginal effect on performance for the single HL net.) Though training time is significantly greater, e.g., 100's of cycles, these far smaller recurrent nets perform competitively and more consistently than the static nets. The recurrent nets contain fewer than 1600 connections: a.) 17 output units; b.) 20 hidden units with a recurrent connection; and c.) 75 input units, e.g. 5 sequential 15-D spectrogram-derived input vectors per sub-sample.

Classification proceeds by sequentially sub-sampling a test spectrogram and tracking activity of the MAXOUT net. Typically, the initial samples, at the transition of background noise and speech, force all outputs towards zero, e.g., the noise response. As sampling proceeds, the correct output node (as well as one or two others) increase activity; the final convergence of the correct output node to a peak value (on average, in excess of 0.8) follows a non-linear path, see also [5]. Significantly, the misclassification is most often the type "NOT CLASSIFIED", e.g., no output node activates enough to drive a MAXOUT convergence over the entire word. Hence, the recurrent nets trained with noise vectors (or a "garbage" output node) resist false positive classification; we have not observed this property in the static nets.

The final row of Table I indicates performance on a multiple-speaker digit recognition problem. The data used are from the same dataset above, but new nets (with 10 output nodes only) are iteratively trained as described in Training and testing, above. Again, fine-tuning the static net input increases performance, though in this case, raw performance is high. The far smaller recurrent net outperforms both the one and two HL static nets.

Conclusions and Discussion

The present effort is aimed ultimately at word-spotting in continuous speech. Thus we must deal directly with the temporal characteristics of the formant-phoneme and word levels. At the word-level the key problem is time-alignment owing to differential speaker rates, amplitudes, fundamental frequencies, etc. We have explored two approaches to the problem. One using static nets in conjunction with an image shifter, and another using (temporally) recurrent nets. In this restricted study, recurrent nets yield competitive classification rates with a level of consistency not seen with static nets.

The present results are similar to those reported by Anderson, et al., [5] and Franzini, et al., [6]. Similarly, the study by Rossen and Anderson [7] suggests that there is a variety of

input representations which may enhance performance of both static and recurrent nets. The usage of different databases makes direct performance comparisons difficult. (We are presently installing the TI/NBS database in order to facilitate such comparisons.) However, these several studies, among others, support the conclusion that recurrent nets offer a viable neural network approach to ASR.

The approach adopted in this study is different in an instructive way from that adopted by Waibel and colleagues [8]. First, we seek to effect discrimination at the word level, thus aiming to implicitly incorporate co-articulation effects and the like, into the network. In their application (Japanese ASR) Waibel, et al., have found it efficacious to focus on the formant-phoneme level; word level recognition is achieved by forming heirarchical nets in a bottom-up fashion. Second, a recurrent neural network structured properly allows each hidden unit to optimize its own time-delay characteristics. The result is smaller networks where each hidden unit requires only small local memory for (summed) activation values. Moreover, the local memory affords the opportunity for more advanced temporal processing by the hidden units, e.g. see [9]. On the other hand, in their application, Waibel, et al., have been able to select and fix temporal dependencies by pre-specifying connection patterns between layers. The differing nature of the two applications may fully account for the differing NN designs. However, it is important to recognize that temporally, ASR is a multi-level problem. An application-driven NN design process ought to take advantage of and be informed by the unique characteristics which exist at each time-scale. The inherent temporal processing capability of recurrent nets makes them strong candidates as components in NN solutions for automatic speech recognition.

References

1. Rumelhart, D. E., G. E. Hinton and R. J. Williams. (1986). Learning internal representations by error propagation. In: **Parallel Distributed Processing: Explorations in the microstructure of cognition. Vol. I** Rumelhart, D. E. and J. L. McClelland, Eds. pgs. 318-362. Cambridge, MA: MIT Press.
2. Elman, J. L. (1988). Finding structure in time. Center for Research in Language Technical Report 8801. UC San Diego. La Jolla, CA.
3. Servan-Schreiber, D., A. Cleeremans and J. L. McClelland. (1988). Encoding sequential structure in simple recurrent networks. Technical Report CMU-CS-88-183. Carnegie Mellon University, Pittsburgh, PA.
4. Solla, S. A., (1988). Accelerated learning in layered neural networks. AT & T Bell Labs Technical Report. Holmdel, NJ.
5. Anderson, S., J. Merrill and R. Port. (1988). Dynamic speech categorization with recurrent networks. In: **Proceedings of the 1988 Connectionist Models Summer School.** Touretzky, D., G. Hinton and T. Sejnowski, Eds. pgs. 398-406. San Mateo, CA: Morgan Kaufman.
6 Franzini, M. A., M. J. Witbrock and Kai-Fu Lee. (1989). Speaker- independent recognition of connected utterances using recurrent and non-recurrent neural networks. In: **Proceedings of the IJCNN.** Washington, D.C. June 18-22, 1989. pgs. II-1-II-6. San Diego, CA: IEEE TAB Neural Network Committee.
7 Rossen, M. L. and J. A. Anderson. (1989). Representational issues in a neural network model of syllable recognition. In: **Proceedings of the IJCNN.** Washington, D.C. June 18-22, 1989. pgs. I-19-I-25. San Diego, CA: IEEE TAB Neural Network Committee.
8 Waibel, A., T. Hanazawa, G. Hinton, K. Shikano and K. J. Lang. (1989). Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing.* Vol. 37. No. 3. pgs. 328-339.
9 Williams, R.J. and D. Zipser. (1989). A learning algorithm for continually running fully recurrent neural networks. Institute for Cognitive Science. UC San Diego. La Jolla, CA.

AN ADAPTIVE DISCRETE-SIGNAL DETECTOR BASED ON SELF-ORGANIZING MAPS

Teuvo Kohonen, Kimmo Raivio*, Olli Simula*, Olli Ventä*, Jukka Henriksson+*

*) Helsinki University of Technology
Laboratory of Computer and Information Science
TKK-F, Rakentajanaukio 2 C, SF-02150 Espoo, Finland

+) Nokia Research Center, Transmission Systems
P.O. Box 156, SF-02101 Espoo, Finland

INTRODUCTION

Self-Organizing Maps are competitive neural networks that both produce localized responses to input signals (i.e., accomplish a kind of vector quantization of an input vector space) and represent the topology of the input signal space over the network [*1,2,3*]. Most of the applications of Self-Organizing Maps are based on the proposition given in, e.g., [*3*] which says that if the input vectors x are samples of a stationary probability density function $p(x)$ then, due to the self-organizing learning algorithm, an ordered image $[(p(x)]^{n/(n+2)}$ will be formed onto the reference (weight, codebook) vectors of the cells of the network. I.e., the ability of the network to optimally align the reference vectors according to the input space is usually utilized.

However, in this report we outline a discrete-signal identification application that, above all, employs the *active topology preserving property* of the learning scheme. In modern signal transmission technology, ideally, values of transmitted signals are quantized, i.e., in one dimension, only certain values, e.g., $a+n(t)h$, $n(t)=0,1,...$ can occur (figure 1) and, in two dimensions, only (coordinate) values of, e.g., gridpoints of a rectangular area can occur. Consequently, the $p(x)$ function is very simple - a sum of properly shifted delta functions. However, due to various reasons, the transmitted signals may be distorted by the transmission path of the communication system (figure 2) and thus, adaptive signal identification mechanisms are needed for recovering from possible errors. Although $p(x)$ is now time-varying it can be assumed that the form of the signal topology is preserved and the self-organizing learning can easily adjust its reference values according to the signal levels.

ADAPTATION BY SELF-ORGANIZING LEARNING

As described above, ideal transmission of quantized signals occupy discrete values, e.g., $x(t) = a+n(t)h$, $n(t) = 0,1,...$ (see figure *1*). Due to disturbances, etc., the receiver sees modified values of the transmitted signals, i.e., $x'(t) = a+n(t)h+e_n(t)$, resulting in signal trajectories such as in figure 2. An obvious choice for the self-organizing framework is now a linear array of learning cells, each characterized by an adaptive parameter m_i. As the ideal and distorted signal values are one-dimensional, the m_i values are also scalars. The m_i values may be initialized to the ideal values, or, according to the signal levels received in the beginning of the transmission, or, they may be given even random values because the m_i values will effectively converge to possible asymptotic values of the received quantized values in the coarse of the self-organizing learning process. The adaptive and time-varying signal identification proceeds according to the following rules that are based on the original self-organizing algorithm.

Figure 1.

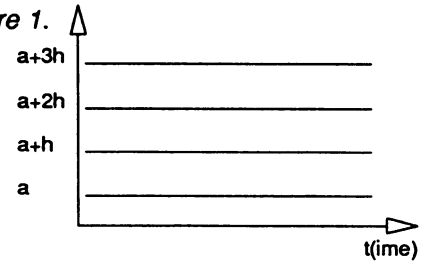
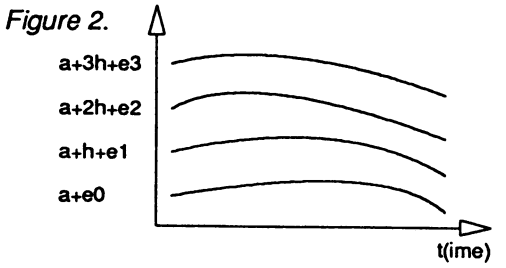


Figure 2.



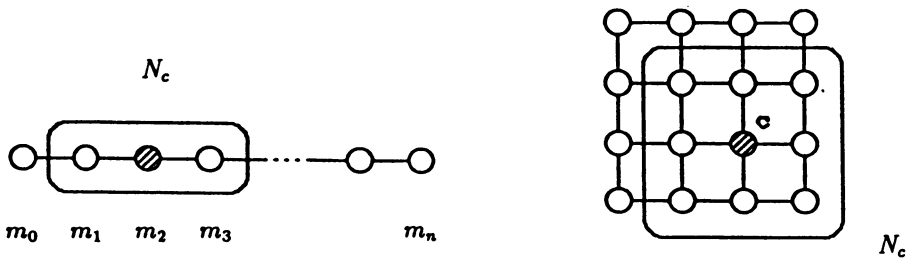


Figure 3. One and two dimensional adaptive networks.

(i) At each discrete time instant t, determine the cell c with the best matching parameter $m_i(t)$ in respect to the current received signal sample $x(t)$, i.e.,

$$\|x(t) - m_c(t)\| = min_i \{\|x(t) - m_i(t)\|\} \quad (1)$$

(ii) Adapt the parameters in the neighborhood N_c of the computed cell c

$$m_i(t+1) = m_i(t) + \alpha[x(t) - m_i(t)], \quad i = c \quad (2a)$$
$$m_i(t+1) = m_i(t) + \beta[x(t) - m_i(t)], \quad i \in N_c, i \neq c \quad (2b)$$
$$m_i(t+1) = m_i(t) \quad i \notin N_c \quad (2c)$$

The topological neighborhood N_c consisted of the recalled cell itself and its direct neighbors up to depth *1,2,...* (see figure *3a*). Since the problem now is not primarily to find an image for a complicated stationary input distribution but rather to follow the drifts, etc., in the quantized signal values we used values for the iteration coefficients, α and β (and not monotonically decreasing functions of time, etc.). As is obvious from the experience of the capabilities of the self-organizing maps in general and also as shown by the demonstrations described below, if the α and β values and the neighborhood radius are selected properly, the m_i values will trace reasonably well the time-varying $x(t)$ values, i.e., adaptively identify the received quantized signal. It must be further emphasized that as the broadcast signal space has a well-defined topology and as the mapping is topology-preserving, due to neighborhood learning, the adaptation is very effective unless the distortions are so large that the linear topology of the quantized signals is destroyed.

The neighborhood learning is always applied symmetrically in each direction in the array of adaptive cells. Because cells near the edges of the array may not have neighbors in both directions the learning causes some bias in the m_i values towards the group center of the parameter values. To compensate for this, the input signal $x(t)$ can be modified to $b_i+d_ix(t)$, b_i and d_i being node-specific parameters, i.e., the input signal space is effectively enlarged yielding the following modified adaptation formulas:

$$m_i(t+1) = m_i(t) + \alpha[b_i+d_ix(t) - m_i(t)], \quad i = c \quad (3a)$$
$$m_i(t+1) = m_i(t) + \beta[b_i+d_ix(t) - m_i(t)], \quad i \in N_c, i \neq c \quad (3b)$$
$$m_i(t+1) = m_i(t) \quad i \notin N_c \quad (3c)$$

The ability of the method to preserve a signal space topology becomes even more evident when the quantization of the signal space is *two* (or higher) *dimensional*, i.e., the ideal signal values occupy, e.g., the coordinate values of the gridpoints of a rectangular area. Two-dimensional signal quantization is utilized in the so-called *QAM coding* in modern communication systems [*4*]. Obviously, it is advantageous to use now a respective rectangular array of learning cells familiar from the multitude of demonstrations given of the Self-Organizing Maps (see figure *3b*). Accordingly, the learning neighborhoods are also two-dimensional. Otherwise, the above adaptation equations are directly applicable to the two-dimensional case, too.

The gridpoints in a two-dimensional space may be shifted, zoomed, rotated, etc., in various ways but still the order of the signal levels tends to be preserved, i.e., the rectangular grid-like structure is preserved. Even when absolute signal levels change drastically, they are effectively and steadily followed by the topology-preserving learning rule.

SIMULATIONS

The purpose of the simulations was two-fold. Firstly, we wanted to demonstrate the *ability* of the self-organizing learning network to identify adaptively discrete and distorted signal levels. Secondly, we wanted to preliminarily survey acceptable values for the coefficients, etc., of the method, primarily for α and β. Especially, we wanted to test the *sensitivity* of the parameters against different sort of deformations in the input space. The emphasis of the simulations was on the two-dimensional quantization.

The adaptation capability was measured in terms of *identification accuracy*, i.e., counting the number of correctly identified signal levels in proportion to the amount of all transmitted signals. A received *x(t)* was always identified according to the nearest-neighbor rule, i.e., Eq. *(1)*. For comparison, the identification accuracy was also measured using a straightforward adaptive method that, in one-dimensional case, traces the smallest and largest values of the received signal in time and then divides the space in between evenly assuming an equally spaced modified quantization. Such an identification technique is commonplace in many practical *AGC* (Adaptive Gain Control) based receivers. In two-dimensional case, both dimensions were treated similarly and the simplified method assumed a signal quantization occupying at the intersections of equally spaced parallel division lines in both directions.

The first simulation was carried out with one-dimensional signal quantization. The ideal quantization consisted of four equidistant levels and the amount of signal samples broadcast were *2000*. An artificial distortion was made in the middle of the transmission by shifting the lowest and highest levels towards the other two levels, however, without intersecting the signal trajectories and thus, maintaining the topology (see figure *4*). The distances of the shifted signal levels to the intact levels were *1/4* of the original. For the form of the shift, a smooth step (like a sigmoid function) was used. In addition, simulations were always carried out with random, normally distributed *noise* included in the received signals. In this particular example, the standard deviation of the noise was $\sigma = 5$ per cent of the difference between the ideal quantization levels. Equations *(1)* and *(2a-c)* were used for the adaptation. The neighborhood radius was *1* in all simulations. Each simulation actually consisted of a series of systematic test runs to find out proper or safe values for α and β to obtain good adaptation. Figure *5* depicts the region of (α,β)-values with identification accuracy of *95* per cent or above for the received signals with time indices *201* through *800*, i.e., during the nondistorted region except noise. Figure *6* depicts the corresponding (α,β)-values for time indices *1251* through *2000*, i.e., after the collapse. For practical operation, (α,β)-values giving high accuracies in both time regions should be chosen. Figure *7* depicts the decrease of identification accuracy obtained with properly chosen (α,β)-values within time indices *1000* through *1150*, i.e., during and right after the transition from normal to collapsed mode. The accuracies are calculated of *30* sample windows, each *10* samples apart. With the *AGC* based method, the identification accuracy was *100* per cent for signals with time indices *201* through *800* and only *85.6* per cent for signals with time indices *1251* through *2000*.

Figure 4.

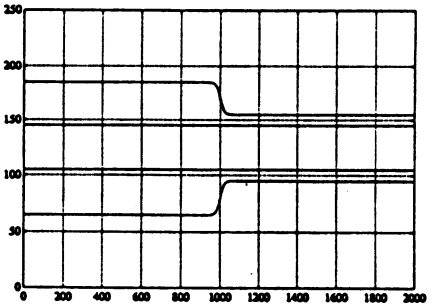
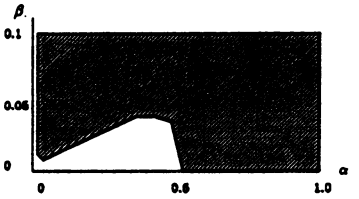


Figure 5.



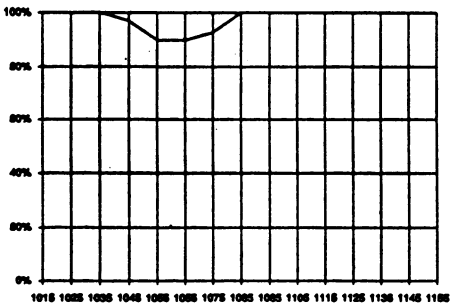


Figure 7.

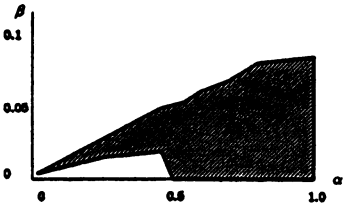


Figure 6.

With two-dimensional signal quantization, the forms of deformations experimented were the following (see also figure *8*). *a)* Collapse of all signal levels or "zooming in", *b)* collapse of corner levels inwards of the grid *c)* rotating the grid by *10*, *20*, or *30* degrees, and *d)* escape of one quantization level from the grid. In each type of deformation, the transition from normal to the abnormal mode occurred in the middle of the transmission, around the time index *1000*, analogously to the one-dimensional simulation. The total amount of samples was extended to *3000*, however. The rate of each transition was the slope of the sigmoid function ($f(t) = 1/(1+\exp(0.1*t))$). The standard deviation of the noise (two-dimensional) varied between *2* and *5* per cent. Both *4 by 4* and *6 by 6* grids were tested. For adaptations, equations *(1)* and *(2a-c)* were used in each case.

In general, the adaptation is most reliable in deformations that cause topologically similar changes to all gridpoints, e.g., zooming in or out, rotation, or shift. The effect of noise was neglible, at least in the range of *0* to *5* per cent (standard deviation). If there are gridpoints whose behavior diverges from the others, e.g., escape, it is advantageous to decrease the β-parameter to make the effect of the neighborhood learning weaker.

As a special case, we will describe the simulations *c)*, i.e., rotating the grid, in more detail. At first, it was not difficult to assign good values for α and β before the transition, i.e., a *100* per cent identification was easily obtained. A *30* degree rotation of a *4 by 4* grid was initiated at time index *751* and was completed at time index *1250*. The standard deviation of noise was *5* per cent. The region of *(α,β)*-values giving a high identification accuracy (*95* per cent or more) for received signals with time indices *1501* through *2200* is shown in figure *9*. As can be seen, the adaptation is robust within a wide range of *(α,β)*-values. For comparison, the identification accuracy of the *AGC* based method was below *25* percent assuming the rotation of the grid can not be taken into account.

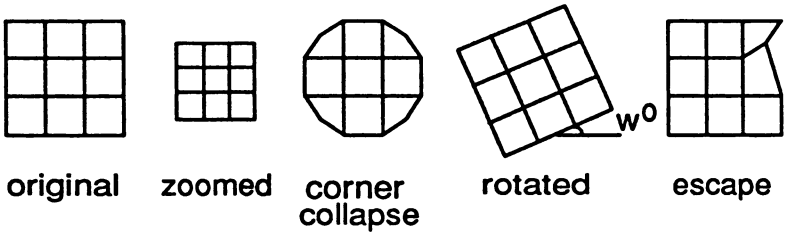


Figure 8.

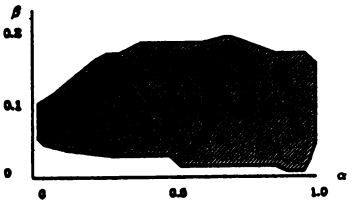


Figure 9.

CONCLUSIONS

These preliminary simulations clearly indicate that self-organizing learning may offer a very effective and robust means for adaptive signal identification in modern communication systems. The virtues of the method can be seen especially with two-dimensionally quantized signals with noise and distortion. The choice of the parameters of the method does not seem to be critical. The adaptation is fairly fast and stable although no provisions of the characteristics of the signals have been made (admitted the test signals were artificial, so far).

Encouraged from these early simulations, we are now concentrating on carrying out more extensive designs and test runs with real communication signal samples.

REFERENCES

[*1*] T. Kohonen, Clustering, Taxonomy, and Topological Maps of Patterns, Proc. 6th ICPR, Munich, Germany, Oct. 19-22, 1982 (Invited Paper), pp. 114-128.

[*2*] T. Kohonen, Self-Organized Formation of Topologically Correct Feature Maps, Biological Cybernetics, Vol. 43, 1982, pp. 59-69.

[*3*] T. Kohonen, *Self-Organization and Associative Memory*, Series in Information Sciences, Vol. 8, Springer-Verlag, 3rd Ed. 1989.

[*4*] B. Carlson, *Communication Systems, An Introduction to Signals and Noise in Electrical Communication*, McGraw-Hill, 1986.

SOME PRACTICAL ASPECTS OF THE SELF-ORGANIZING MAPS

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

Abstract. The main purpose of this presentation is to advocate the Self-Organizing Map algorithm by emphasizing the simplicity of its implementation by standard hardware (electronic, optic). In particular, if the original dot-product matching is used, the Map is easily implementable, e.g., by optic devices; the learning law must then be modified accordingly. Nonetheless it remains possible to define almost arbitrary nonlinearly separable and even disjoint class regions with this simple algorithm. - A survey of a few earlier and contemporary applications is also given.

1. Motivation of the Map for a Neural Network Paradigm

If we try to make a comparison of the nowadays perhaps already familiar Self-Organizing Map [1], [2] with the most widespread NN structure, namely, the Backpropagation network (BP), the most salient difference is that BP is designed to implement a wanted input-output mapping of signal patterns, whereas the Map carries out a Vector Quantization, directly identifying the input signal pattern with the closest "codebook pattern" (in some suitable metric). If one then applies these networks to pattern recognition or pattern classification tasks, the difference becomes even more apparent. In BP, a "regularizing" input-output mapping between sample data is first designed in terms of nested functional expansions of the form

$$y_k = \sigma(\sum_h w_h \sigma(\sum_i v_i \sigma(...\sigma(\sum_j u_j x_j)...))) \tag{1}$$

where the x_j are input signals and the y_k output responses, respectively, and σ denotes the "sigmoid" nonlinearity; classification of the input vector then results on the basis of the relative output values of y_k. The Map, on the other hand, does not need nested mappings, since *in classification, it is only the decision border between classes in the signal space that is important, and any (optimal, nonlinear) form for it can directly be set by using a sufficient number of suitably located codebook vectors for each class*; to define the codebook vectors on the basis of wanted decision borders is a vastly lighter task than determination of the functional expansions in BP networks. Still the classification accuracy of the Map is at least as high as that of the BP networks.

2. The Euclidean Map Algorithm

Let us denote the input vector by $x \in R^n$ and the codebook vectors of the Map by $w_i \in R^n$, respectively. The simplest matching defines the nearest codebook vector, with subscript c, in the Euclidean metric:

$$\|x-w_c\| = \min_i \{\|x-w_i\|\} \ . \tag{2}$$

Updating of the codebook vectors is usually based on the following law. If $x = x(t)$ and $w_i = w_i(t)$ are discrete-time sequences of vectors, then

$$w_i(t+1) = w_i(t) + \alpha(t)[x(t)-w_i(t)] \ \text{ for } i \in N_c \ ; \qquad w_i(t+1) = w_i(t) \ \text{ for } i \notin N_c \ , \tag{3}$$

where $\alpha = \alpha(t)$ is a monotonically decreasing "gain" sequence ($0 < \alpha < 1$), and N_c is the "topological neighborhood" of the codebook vector w_c ("winner") (see [1], [2]).

If the Map is then used for classification, each class is usually made to contain a small number of codebook vectors. Post-training (fine-tuning) of the Map is now carried out in a *supervised* process whereby the class-affiliation of the input vectors x must be known. If the class-affiliation of the closest codebook vector in the Map on the basis of Eq. (2) is the same as that of x, then a *positive* α like in Eq. (3) is used; on the other hand, *if the classification was wrong, then* $-\alpha$ *is used instead of* α. This principle (which is applied after the Map was already formed and especially with $N_c = \{c\}$) is one of the algorithms called Learning Vector Quantization [2], [3].

3. The Dot-Product Map Algorithm for Simpler Hardware Implementation

The hardware implementation of the Map, in which the "optimal" codebook vectors can be computed off-line, may be extremely simple. For instance, with high-dimensional input vectors it is often possible to normalize them, or at least to assume that all the important information is contained in the relative values of the components. All the available information is then preserved although the codebook vectors are normalized. Such a Map may be implemented by a linear mapping followed by a maximum selector ("winner-take-all"). The matching law is first modified to read:

$$x^T w_c = \max_i \{x^T w_i\} \quad . \tag{4}$$

If this transformation is used, the *learning law* for the codebook vectors w_i must read:

$$w_i(t+1) = \frac{w_i(t) + \alpha' x(t)}{\|w_i(t) + \alpha' x(t)\|} \quad \text{for } i \in N_c \; ; \qquad w_i(t+1) = w_i(t) \quad \text{for } i \notin N_c \; , \tag{5}$$

where $\alpha' = \alpha'(t)$ is a monotonically decreasing scalar function of time (this time $0 < \alpha' < \infty$), and N_c is a similar neighborhood around the "winner" w_c as in the previous Map algorithm. One might try the form $\alpha' = \alpha_0/t$ where α_0 is rather large (10 to 100).

If this Map is again applied to classification tasks, a supervised learning scheme analogous to that mentioned in Sec. 2 can be used; now, again, the sign of α' is a function of classification vs. misclassification of the known input training vectors x.

A simple hardware implementation, where the dot products are first computed by a "synaptic matrix" (optically: by transmission filters) and summation is made by the columns (like in many neural models) is illustrated in Fig. 1. The different dot products are compared by a "winner-take-all" circuit that outputs a 1 at the line corresponding to the maximum dot product $x^T m_c$ and 0 at all the other output lines. The outputs corresponding to all those codebook vectors that belong to the same class are combined with a logic OR gate; each gate thus corresponds to one class and the corresponding output indicates to what class x belongs.

4. Survey of Practical Applications of the Maps

In addition to numerous more abstract simulations, theoretical developments, and "toy examples", the following practical problem areas have been approached by the Self-Organizing Maps. In some of them rather concrete work is already in progress. Although the number of published papers is already much greater, I have tried to collect references to the most central ones.

- Statistical pattern recognition, especially recognition of speech [4], [5];
- control of robot arms, and other problems of robotics [6], [7], [8];

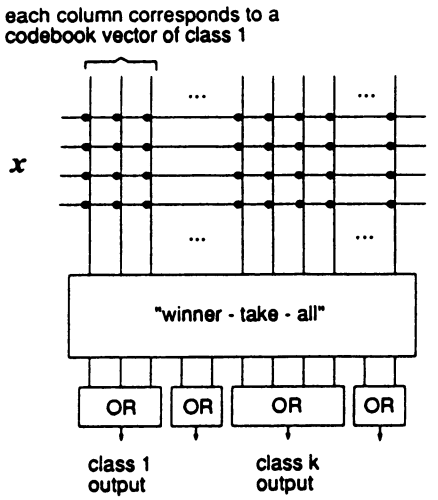


Fig. 1. Schematic illustration of the Dot-Product Self-Organizing Map. Each column of the "synaptic matrix" computes a dot product of the type $x^T w_i$.

- control of industrial processes, especially diffusion processes in the production of semiconductor substrates [9], [10];
- adaptive devices for various telecommunication tasks [11], [12];
- optimization problems [13]; and
- analysis of semantic information [14].

Among these, the application to speech recognition has longest traditions in demonstrating the power of the Map method, when dealing with difficult stochastic signals. My personal expectations are, however, that the greatest industrial potential of this method may lie in process control and telecommunication tasks.

It is a little surprising that so few applications of the Maps to Computer Vision are being studied. This does not mean that the problems of vision were not important; it is rather that automatic analysis and extraction of visual features, without any heuristic or analytical approach, has turned out to constitute an extremely hard problem. In the biological vision, and in its artificial implementations, one probably needs a very complicated hierarchical system that employs many stages (i.e. several different maps). One of the unclear problems is how the maps should be interconnected, e.g., whether one should use special interfaces; and in hierarchical systems, adaptive normalization of inputs also seems necessary. There are only some isolated problems, such as texture analysis, that might be amenable to the basic method as such.

The problem of hierachical maps has indeed turned out to be very tough. One of the particular difficulties arises if the inputs to a cell come from very different sources; it then seems inevitable that an unsymmetrical distance function, in which the signal components are provided with adaptive tensorial weights, must be applied [15]. Another aspect concerns the interfaces of modules in a hierarchical map system: the signals merging from different modules may have to be combined nonlinearly [16].

It has already been demonstrated that the Map can be used, e.g., as a preprocessing stage for other models [17], [18]. In the Counterpropagation Network of Hecht-Nielsen [19], the map is nicely integrated into a hierarchical system as a special layer.

More mathematical aspects of the Maps can be found in Refs. [20] through [25].

References

[1] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biol. Cybern.*, vol. 43, pp. 59-69, 1982.

[2] T. Kohonen, *Self-Organization and Associative Memory*, 3rd ed. Berlin, Heidelberg, Germany: Springer-Verlag, 1989.

[3] T. Kohonen, "An introduction to neural networks," *Neural Networks*, vol. 1, pp. 3-16, 1988.

[4] T. Kohonen, "The 'neural' phonetic typewriter," *Computer*, vol. 21, pp. 11-22, March 1988.

[5] T. Kohonen, K. Torkkola, M. Shozakai, J. Kangas, and O. Ventä, "Microprocessor implementation of a large vocabulary speech recognizer and phonetic typewriter for Finnish and Japanese," *Proc. European Conference on Speech Technology*, CEP Consultants, Edinburgh, pp. 377-380, 1987.

[6] D.H. Graf and W.R. LaLonde, "Neuroplanners for hand/eye coordination," *Proc. IJCNN 89 Int. Joint Conf. on Neural Networks*, Washington, D.C., pp. II-543 - II-548, 1989.

[7] H.J. Ritter, T.M. Martinetz, and K.J. Schulten, "Topology conserving maps for learning visuo-motor-coordination," *Neural Networks*, vol. 2, pp. 159-168, 1989.

[8] H.J. Ritter and K. Schulten, "Extending Kohonen's self-organizing mapping algorithm to learn ballistic movements," *NATO ASI Series*, vol. F41, pp. 393-406, 1988.

[9] K.M. Marks and K.F. Goser, "Analysis of VLSI process data based on self-organizing feature maps," *Proc. Neuro-Nîmes'88*, Nîmes, France, pp. 337-347, 1988.

[10] V. Tryba, K.M. Marks, U. Rückert, and K. Goser, "Selbstorganisierende Karten als lernende klassifizierende Speicher," *ITG Fachbericht*, vol. 102, pp. 407-419, 1988.

[11] D.S. Bradburn, "Reducing transmission error effects using a self-organizing network," *Proc. IJCNN 89 Int. Joint Conf. on Neural Networks*, Washington, D.C., pp. II-531 - 537, 1989.

[12] T. Kohonen, K. Raivio, O. Simula, O. Ventä, and J. Henriksson, "An adaptive discrete-signal detector based on self-organizing maps," submitted to IJCNN-90-WASH DC.

[13] B. Angéniol, G. de la Croix Vaubois, and J.-Y. Le Texier, "Self-organizing feature maps and the travelling salesman problem," *Neural Networks*, vol. 1, pp. 289-293, 1988.

[14] H. Ritter and T. Kohonen, "Self-organizing semantic maps," *Biol. Cybern.*, vol. 61, pp. 241-254, 1989.

[15] J. Kangas, T. Kohonen, J. Laaksonen, O. Simula, and O. Ventä, "Variants of self-organizing maps," *Proc. IJCNN 89 Int. Joint Conf. on Neural Networks*, Washington, D.C., pp. II-517 - II-522, 1989.

[16] H. Ritter, "Combining self-organizing maps," *Proc. IJCNN 89 Int. Joint Conf. on Neural Networks*, Washington, D.C., pp. II-499 - II-502, 1989.

[17] R.M. Holdaway, "Enhancing supervised learning algorithms via self-organization," *Proc. IJCNN 89 Int. Joint Conf. on Neural Networks*, Washington, D.C., pp. II-523 - II-529, 1989.

[18] P. Morasso, "Neural models of cursive script handwriting," *Proc. IJCNN 89 Int. Joint Conf. on Neural Networks*, Washington, D.C., pp. II-539 - II542, 1989.

[19] R. Hecht-Nielsen, "Applications of counterpropagation network," *Neural Networks*, vol. 1, pp. 131-139, 1988.

[20] M. Cottrell and J.-C. Fort, "A stochastic model of retinotopy: A self-organizing process," *Biol. Cybern.*, vol. 53, pp. 405-411, 1986.

[21] M. Cottrell and J.-C. Fort, "Étude d'un processus d'auto-organisation," *Ann. Inst. Henri Poincaré*, vol. 23, pp. 1-20, 1987.

[22] S.P. Luttrell, "Self-organization: A derivation from first principles of a class of learning algorithms," *Proc. IJCNN 89 Int. Joint Conf. on Neural Networks*, Washington, D.C., pp. II-495 - II-498, 1989.

[23] H. Ritter, "Asymptotic level density for a class of vector quantization processes," Helsinki University of Technology, Lab. of Computer and Information Science, Report A9, 1989.

[24] H. Ritter and K. Schulten, "On the stationary state of Kohonen's self-organizing sensory mapping," *Biol. Cybern.*, vol. 54, pp. 99-106, 1986.

[25] H. Ritter and K. Schulten, "Convergency properties of Kohonen's topology conserving maps: Fluctuations, stability and dimension selection," *Biol. Cybern.*, vol. 60, pp. 59-71, 1989.

A Technique for the Classification and Analysis of Insect Courtship Song[1]

Eric K. Neumann, David A. Wheeler, Jamie W. Burnside*, Adam S. Bernstein, and Jeffrey C. Hall, Dept. of Biology, Brandeis University, Waltham, MA. 02254, *M.I.T. Lincoln Laboratory, Lexington, MA 02173[2].

Abstract

A technique is described and preliminary results presented for the classification and analysis of acoustic signals from insect courtship song using Neural Networks. The system consists of two parts: 1) a Kohonen self-organizing network that directs its output into 2) a 2-layer feedforward network which uses back-propagation learning. The signal is first preprocessed by a zero-crossing transform before training. The network eventually learns an optimal representation of the pulse events based on learned vector quantization (LVQ). The vector-quantized output is used to calculate a histogram of time-summed quantized events and is then used as input for the subsequent feedforward network. The second network learns with a modification of back-propagation the correspondence between the processed-song input and an external parameter such as the genotype of the song producing animal. The adapted system can then be used to analyze new mutant songs and to classify them into appropriate categories based on genetic allelism and song characteristics. Finally we describe a novel technique for the identification, classification and analysis of other complex behaviors and its application to the general analysis of animal behavior and communication phenomena.

Introduction

The complex behaviors of animals are of great interest to both behaviorists and neurobiologists alike. Models of how the nervous system of an animal is capable of producing complex behavior are being developed for many species (Drosophila, Hall [1]; Aplysia, Kandel [2]; canary, Nottebohm et al. [3]). With the advent of new molecular techniques and tools for analyzing neural systems, the need to accurately correlate behavioral patterns to the data obtained by these techniques is critical. A complex system such as an animal's nervous systems can yield very complicated results to even the most focused investigation. This translates into a need for more accurate and comprehensive techniques which reproducibly measure and classify these behavioral ensembles.

We have been investigating the mechanism of male Drosophila melanogaster (fruit fly) courtship song from both wild type and song mutants. The song is produced by wing vibration and is made up of intermittent hum and pulse component. The latter consists of a train from a few to many individual pulses of 2-3 cycles/pulse(polycyclicity) at ~225 Hz separated from each other by a mean interpulse interval (IPI) of 35-45 milliseconds (Fig. 1). It is a necessary part of the courtship repertoire and is believed to enhance the female's receptivity for mating. Variation is observed between pulses from different trains, but is much less between pulses within a train. A complete song lasting many minutes can contain many types of pulses with different amplitudes, frequencies, and polycyclicity.

Figure 1. Pulse song from wild type Drosophila melanogaster. Time intervals are 50 milliseconds.

Two separate mutations, cacophony and dissonance, which affect pulse polycyclicity are presently being studied at the molecular level in our lab. However, both mutations exhibit high pulse variation and therefore simple classification based on pulse polycyclicity alone is not very accurate or reliable. In addition, many experiments involving mutant transformation rescue and other genetic modifications yield very complex results which are not definable by any single pulse feature, such as polycyclicity. We eventually wish to develop a fast, automated approach to screening and analyzing these and other putative mutations. Before this can be done, a technique must be developed on the computer which provides for a more complete song phenotype assessment.

1. This project was funded by Grants GM-21473 from the NIH to J.C.Hall.

2. The views expressed here are the preliminary views of the authors, and do not reflect the views or policies of M.I.T. Lincoln Laboratory.

Conventional signal processing techniques (signal rectification, low-pass filtering, peak detection) have been employed to identify putative pulses in a courtship song recording; however ~10% of the pulses are either not identified or other non-pulse events (wing cleaning, female wing vibrations, bumping) are incorrectly labeled as pulses. In addition, though their polycyclicity and IPIs are measured to yield an average set of values for a given song, the identified pulses are not individually classified because of their complex variability. In order to design a more complete analytical tool, the use of adaptive techniques which would specifically tune a system to features in the acoustic signals of interest was investigated. The goal was to develop a system which could best describe a given pulse, classify it, and then use the entire pulse classification record of a song to correctly assess the type of fly that produced it.

Kohonen [4] describes a system which is able to vector quantize data. This system adapts itself using self-organizing rules to effectively construct a set of prototype input vectors. These vectors when compared to real inputs yield a least mean squared error based on a nearest neighbor measure. The result is a network which best describes the distribution of the input data and can be used to label signals into classes if the prototypes are ordered according to some clustering (classification) method. This technique can also be used as an effective codebook for data compression.

The vector quantized song data are then used to adapt a feedforward network that attempts to map the song to the correct type of animal producing it. Preliminary data on the effectiveness of this approach are presented and additions to the technique are suggested. A brief outline to be applied to more general behavioral analysis is also presented.

Implementation

Courtship songs of *Drosophila* males were monitored using a special microphone system called the Insectavox [5]. The song data were digitized at a sampling rate of 1 kHz. Pulses were identified manually by an operator who logged in the individual pulses using previously described software [6].The location of the pulse was defined as the time at which the maximum absolute amplitude of the pulse occurred. The identified pulses were used to create zero-crossing (ZX) interval files for each of the pulses. This consisted of starting at the pulse location and calculating the ZX interval for that half cycle and also for 4 preceding and 4 subsequent ZX intervals. A theorem by Logan [7] states that if a signal is contained within an octave bandwidth, then the signal can be described completely by its zero-crossing information and a multiplicative constant. The spectral energy of a pulse is concentrated in such a frequency interval, and therefore can be effectively transformed by ZX without much loss of information. In order to compare the efficiency of the adaptive system to more conventional signal processing and identification methods, pulses were also identified by an automatic procedure. This procedure employed threshold peak detection of rectified and low-pass filtered signal. For each pulse event a total of nine ZX intervals were stored along with its time, its event number and whether the pulse was identified (true) or not (missed). Non-pulse events which were incorrectly identified by the conventional automatic approach were also entered into the record, along with their ZX information (non-pulse). The auto-identification information was later used for analysis of the classification capabilities of the network.

I. Kohonen Self-Organization

A Kohonen LVQ network [8] with 72 nodes and nine inputs to each node was used to learn the distribution of the pulses' ZX information. Each node i consisted of nine coefficients for each input element of vector **x**. This network was then allowed to adapt to the ZX information of each pulse. This was performed by adjusting the coefficients of each node according to the rules:

choose node **c** such that:

$$\| x - m_c \| = \min_i \{ \| x - m_i \| \} \quad (1)$$

i is within a neighborhood of **c** (i.e. $NE_c(t)$) if:

$$\| m_i - m_c \| \leq Ao*\exp(-t/T1) \quad (2)$$

then adapt the coefficient vector according to:

$$m_i(t+1) = \begin{cases} m_i(t) + a(t)[x(t) - m_i(t)] \text{ for } i \in NE_c(t) & (3) \\ m_i(t) \text{ otherwise} & (4) \end{cases}$$

$$\text{where } a(t) = A1* \exp(-t/T2) + A2 \quad (5)$$

x is the input vector (ZX) and m_i is the node vector (each node corresponds to a pulse prototype). The m_i are adjusted first coarsely to the the input distribution with $NE_c(t)$ and a(t) being maximum for $t \approx 0$. As t approaches T1 and T2, $NE_c(t)$ and a(t) both decrease exponentially, respectively, and the adjustments are fine. The final m_i for all the nodes eventually attain a

distribution similar to the training input distribution. The nodes can be represented as a set of points, each corresponding to a node, whose position in input space (i.e. nine dimensions) is specified by the coefficients of that node. An input (e.g. pulse) is also represented in the same space, and that node which is closest to the input (Euclidean distance) is "activated".

After training the network can be used to vector quantize novel inputs whose distribution is similar to the training set's. If the distributions are stationary, then the mean squared distortion error (MSE) is minimized:

$$< \text{error}^2 > = E[\min_i \{ \| x(t) - m_i \|^2 \}] \quad (6)$$

where i = 1...M.

This quantity is also useful in assessing the efficacy of vector quantization for a given network presented with a given song sample.

Therefore for M nodes, the distortion error of a signal is minimized for the mapping of a signal to the "closest" node. Subsequently, all pulses can be mapped to a node to create a compressed file of only the node labels corresponding to the original pulse sequence. Such a file still retains all the information as to the type of pulse event and their distribution. However, the pulse data has been significantly compressed from a vector of nine elements to a scalar quantity ranging from 1 to M. The "node" event file can be utilized in several different ways in the analysis of song structure. In this set of experiments, the entire accumulated event record (i.e. node activation histogram) is used as a "signature" to predict some characteristic of the animal which produced the song.

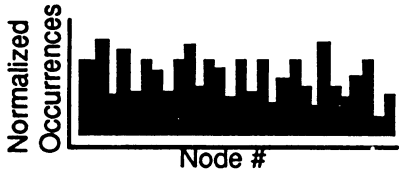


Figure 2. Node activation histogram of a song after being analyzed by a self-organized network.

A node activation histogram can be calculated based on the distribution of nodes activated by a given song sample (Fig.2). The histogram can be represented as a vector which is usually scaled to unity. This vector can be thought of as a time-independent (stationary) signature for the song. A measure to quantitate the similarity between song activation histogram vectors was defined using the cosine of the angle between different vectors: the less in common the distributions, the greater the orthogonality and the angle. This was performed by calculating a dot product between normalized histogram vectors (1-D correlation) or between normalized matrices containing the distribution of node activation in one index and their distortion error in the other (2-D correlation). Both give the cosine of the angle between the distribution vectors or matrices (orthogonality) ranging from -1 to 1 which is effectively a measurement of similarity between event distributions .

The node activation histogram is useful for the screening of potential behavioral and song mutants in which this lab is active. To do this, another system is required that will be able to best estimate whether a given song is wildtype or mutant. We have utilized a two-layer feedforward network whose input is the normalized node activation histogram vector from each song file.

This vector is fed into a feedforward network and the output trained to match one of the genotypes. The network is trained on the vector pulse records for many examples using a modified back-propagation method. After training, the feedforward network is tested with novel examples.

II. Backpropagation Learning Rule

The familiar error backpropagation learning rule [9] was used to train a two-layer (one hidden layer) neural network. In a conventional backpropagation network (BPN), the connection weights are modified for each input/output pair (p) according the the rule:

$$\Delta_p W_{ji} = \mu \delta_{pj} O_{pi} \quad (7)$$

where the error, δ_{pj}, is defined as

$$\delta_{pj} = (T_{pj} - O_{pj}) f'(net_{pj}) \quad (8)$$

for the output layer and as

$$\delta_{pj} = f'(net_{pj}) \Sigma_k \delta_{pk} W_{kj} \quad (9)$$

for hidden layers. We used the conventional sigmoidal function:

$$f(net_{pj}) = 1 / (1 - e^{-net_{pj}}) \quad (10)$$

The variable μ in Eq. (7) is the scalar learning rate coefficient and net_{pj} is the linear sum of weighted inputs. In order to increase the learning rate, our implementation used adaptive learning rate coefficients. The scalar learning rate coefficient in Eq. (7) was replaced by a learning coefficient matrix. The weights of the network were modified only after an entire pass through the training set:

$$\Delta W_{ji} = \mu_{ji} \Sigma_p \delta_{pj} O_{pi} \quad (11)$$

This was both computationally efficient and prevented the order in which the training set was presented to the network from affecting the adaptive learning coefficients. The learning rate coefficients, μ_{ji} , were adapted according to the delta-bar-delta rule of Jacobs [9]:

$$\overline{\Delta W}_{ji}(n) = \theta \overline{\Delta W}_{ji}(n-1) + (1 - \theta) \Delta W_{ji}(n) \quad (12a)$$
$$\text{where } 0 \le \theta \le 1$$

$$\text{if } \overline{\Delta W}_{ji}(n) * \Delta W_{ji}(n) > 0$$
$$\text{then } \mu_{ji} = \mu_{ji} + \kappa \quad (12b)$$

$$\text{if } \overline{\Delta W}_{ji}(n) * \Delta W_{ji}(n) < 0$$
$$\text{then } \mu_{ji} = (1 - \phi)\mu_{ji} \quad (12c)$$

where $\overline{\Delta W}_{ji}(n)$ is the exponential moving average of ΔW_{ji} and n represents the nth pass through the training set. θ determines the "fading memory" characteristics of $\overline{\Delta W}_{ji}(n)$.

The rationale behind the delta-bar-delta rule is as follows: An exponential moving average of the weight updates, $\overline{\Delta W}_{ji}(n)$ in Eq. (12a), is multiplied by the current weight update, $\Delta W_{ji}(n)$. If the product is positive, stable learning is occurring and the learning rate coefficient, μ_{ji}, can be increased. If the product is negative, learning is unstable or oscillatory and the learning rate coefficient is decreased to improve convergence stability. Since we are searching for the minimum of an error surface, this can also be thought of as increasing the learning rate when passing over "smoothly sloping" regions of the error surface and decreasing the learning rate when we traverse a "valley" of the error surface. The learning rate is increased linearly by an amount κ, but decreased geometrically by the factor ϕ. This has been found to produce more stable convergence by quickly decreasing the learning coefficient when crossing a "valley", but not letting it increase too quickly when on a "smoothly sloping" section of the error surface.

The delta-bar-delta rule has similar advantages to using a momentum term [10] but unlike momentum, the learning rate can be allowed to increase without bound when appropriate. It is limited only by the term κ and the number of iterations through the training set. This has the advantage of speeding convergence on the "smoothly sloping" section of the error surface and is superior to momentum for most cases.

The values of θ and ϕ which produced consistent convergence of the network were 0.7 and 0.2, respectively. This produced more than an order of magnitude improvement over a constant learning coefficient in terms of convergence speed. A judicious choice of κ is dependent on the magnitude of the input vector and the size of the training set.

Results

In our experiments, we first trained the Kohonen self-organizing network on all the pulses of all the training files. The network consisted of 72 nodes, each receiving the nine inputs. These files consisted of 6 wildtype songs, 6 *cacophony* mutant songs and 4 *dissonance* mutant songs, and 3 *period* songs. Both wild type and *period* exhibit normal songs, while the other two mutants specifically affect pulse song, albeit not consistently. The total number of pulses used to train the network were 29,000 and the whole set was repeated 3 times (i.e. 87,000 pulse examples). The Kohonen training parameters were chosen to be: for the learning rate Ao = .001, T1 = 30,000; for the neighborhood contraction rate: A1 = 1.5, A2 = .001, T2 = 3000 .

Our experience has shown us that T1 should be at least 3-fold lower than the total training samples, and T2 at least 10 fold lower than T1. A1 was based on the radius of the initial random distribution and A2 on the desired smallest distortion error, usually set to 0. Ao was initially chosen based on an equation derived using a few assumptions (not shown). The initial values of each set of node coefficients were chosen randomly using a Gaussian distribution around the mean pulse value for all the song files.

The effective data compression is determined as the number of bits required to describe the original pulses divided by the bits required to describe the activated node maximized for entropy. The ZX transform has the added advantage of staying constant with increasing sampling rate. The compression ratios for tested songs

is ~50:1 for 1kHz sampling, and increases linearly with sampling rate. The bit rate (bit/sample) of the vector quantized ZX-preprocessed signal is defined as $\log_2 72$ nodes / 9 ZX-intervals = .69 bit/sample.

After completion of learning, the same set of examples were entered into the network and the normalized node activation histogram calculated. Both 1-D and 2-D correlation measurements were used to give an indication of how similar pulses classifications were compared to non-pulse classifications. Effective separation is accomplished using the 2-D correlation, in which 98% of pulses are correctly included as pulses, while ~75% of the previous falsely classified non-pulses are now properly excluded. This illustrates that pulses can be partially separated from non-pulses by VQ alone.

The same correlation measurements were used to compare different songs (pulse events only) with each other. Songs from the same and differing genotypes were compared to each other this way to determine if the Kohonen network was sufficient in distinguishing the genotypes by the node activation overlap. The correlation results indicate that most songs cannot be properly distinguished by this analysis alone.

The fact that a substantial portion of the song histograms from different genotypes overlap suggests that the genotypes share some pulse types. Furthermore, as the number of nodes defined increases, the overlap of general pulse classes with nodes will decrease due to the finer nodal separation of pulses within these classes. These two obstacles make simple comparisons based on scalar correlations difficult.

Therefore, an additional method is required which will identify those prototypes or groups of prototypes which belong to a pulse class and are most significant in correctly distinguishing the song according to genotype. Also, the analysis of an M-dimensional signature may be more complicated than simple correlation analysis (i.e. involving higher than second-order combinations of features). The output was targeted to the different genotype classes and the weights adapted according to the modified back-propagation rules described above. It was hoped that the song signatures would contain enough information to be able to distinguish them based on the genotypes that produced them.

Table 1 shows the averaged results of four runs in which the BP network was trained on a set of songs of all genotypes and then tested on four novel songs. The output response signifies the ability of the networks to

	Test Song Type			
	WT	*cac*	*diss*	*per^0*
Output				
WT	**0.73**	0.32	0.09	**0.53**
cac	0.08	**0.58**	**0.74**	0.33
diss	0.08	0.26	0.31	0.19
per^0	0.49	0.06	0.07	0.09

Table 1. Average output response of BP network trained on 4 genotypes (15 songs total) and tested on 4 novel songs of each genotype.

generalize to songs it has not seen before. The highest values in each column represent the network's "prediction". In summary, WT and *per^0* are classified both as WT, while *cac* and *diss* are usually classified as *cac*. This result is reasonable based on the observed pulse characteristics of these songs. These results indicate the use of the BP network after VQ is essential for proper song classification.

Future Applications

The system described can be adapted to more general behavioral analysis beyond song. Inputs could include animal movement, orientation, courtship components, and other measurable, and putatively significant aspects of behavior. These behavioral responses could be compressed using the Kohonen self-organizing network described here. Alternatively, or additionally, they could be orthogonally projected into a subspace to remove noise and insignificant features from the data. The compressed/projected data would then be inputted into a network similar to the feedforward network. This input need not be limited to time-independent analysis. Analysis using sequence dependent data would allow time and sequence specific events to be analyzed.

However, the "target" of the data is not limited to genotype predictions. It can be also targeted to experimental stimuli presented to the animal before analysis. Such an adapted system would contain information about the correlation between the observed behavior and the presented stimuli. In other words, the network could be used to identify and determine the underlying principles governing complex behaviors and the relation between different behavioral modes. It would not necessarily explain the behavioral mechanisms in detail, but it would aid in the construction of appropriate behavioral models. Such "black box" systems could be made to yield important information of their behavioral mechanisms by the careful mapping of their highly non-linear, yet

relationally significant, external expression.

Analysis could be expanded to all forms of animal communication and complex behavioral repertoires. Factors controlling and affecting these expressions could be studied and would contribute in the synthesis of models. The ability for these networks to achieve these aims is based on the fact that adaptive networks are able to focus and tune in the important features of phenomena (e.g. communication and behavior) and correlate them with any causal effectors.

Conclusion

A two-part Neural Network was used to develop a representation of specific preprocessed acoustic events, and to attempt to categorize them according to genotype through supervised learning. The results of this preliminary set of experiments, which are not conclusive, do indicate several key points:

1. Certain acoustic signals can be effectively described by their ZX interval information so as to retain enough information about their type for classification.

2. The condensed ZX information can be used to train a Kohonen self-organizing network. The adapted network is then an effective form vector quantization of the original acoustic data with minimized mean squared error.

3. The vector quantized events can be used to partially distinguish pulse events from non-pulse events without supervision or *a priori* knowledge of the events. ~75% of previous incorrectly classified events are correctly classified.

4. The stationary description of acoustic data processed by the Kohonen network (i.e. VQ) is sufficient for song-genotype matching using a subsequent two-layer feedforward network. The arbitrariness of the unsupervised adaption in the Kohonen network is unimportant for the BPN to function properly.

5. Early results indicate that the final dual-network system is capable of generalizing to songs on which it has not been trained.

Obviously, the effectiveness of the combined system classification is dependent on the performance of the Kohonen VQ. Improvements in the distribution and number of nodes may radically enhance classification. We have not yet tried sufficient variations of all such parameters. To this effect experiments on variations of these parameters will be performed. Future experiments also could utilize the sequence information in the VQ event files in order to find time-dependent features which differ significantly between animal songs of different genotypes. Such time-dependence can be analyzed in a variety of ways: 1) VQ song data can be analyzed for time-dependence of pulse types over different time-frames, 2) pulse probabilities can be analyzed for dependence on previous pulse events (first-order transition probabilities), or 3) self-organizing network could be designed for the elucidation and identification of complex time-dependence between pulse-events.

References

[1] Hall, J. C. (1982). Genetics of the nervous system of *Drosophila*. Q. Rev. Biophysics 15: 223-479.

[2] Kandel, E. R. (1976). *Cellular Basis of Behavior*. Freeman and Company.

[3] Nottebohm, F., Kelley, D.B. and Paton, J.A. (1982). Connections of vocal control nuclei in the canary telencephalon. J. Comp. Neurol. 207: 344-357.

[4] T. Kohonen, *Self-Organization and Associative Memory, 2nd Ed.* Berlin: Springer-Verlag,1988.

[5] Gorczyca, M. and Hall J.C. (1987). The Insectavox, an integrated device for recording and amplifying courtship song of *Drosophila*. Drosophila Info. Service, 66: 157-160.

[6] Wheeler, D.A., Fields, W.L., and Hall, J.C. (1988). Spectral analysis of *Drosophila* courtship song: *D. melanogaster*, *D. simulans*, and their Interspecific Hybrid. Behav. Genet. 18: 675-703.

[7] Logan, B.F. Jr. (1977). Information in the zero-crossing of bandpass signals. *Sys. Tech. J.* 56: 487-510.

[8] Nasrabadi, N.M. and Feng, Y. (1988). Vector quantization of images based upon the Kohonen self-organizing feature maps. IEEE International Conference on Neural Networks.

[9] Rumelhart, D.E., McCelland, J.L, and Williams, R.J. (1986). *Parallel Distributed Processing*. vol. 1, p. 327

[10] Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaption. *Neural Networks*,vol. 1, p. 295.

Radar Classification of Sea-Ice Using Traditional and Neural Classifiers

Jim Orlando, Richard Mann and Simon Haykin
Communications Research Laboratory
McMaster University
Hamilton, Ontario, Canada
L8S 4K1

Abstract

This paper presents the classification results of applying traditional classifiers and neural classifiers to the same real-world problem. The returns of a surface-based dual-polarized radar are used to classify sea-ice fields into one of four possible classes: first-year ice, multiyear ice, icebergs and shadows cast by icebergs. It is shown that a multilayer perceptron classifier trained with the back-propagation algorithm and a *Kohonen*/learning vector quantization (LVQ) classifier provide performance comparable to the traditional *Bayesian* (*Gaussian*) classifier. The decision regions of the classifiers are plotted, and are found to be similar.

I. Introduction

Ice fields in the Arctic are generally composed of three different types of ice, first-year ice (salt-water frozen during the present winter), multiyear ice (salt-water frozen during previous winters), and icebergs (large pieces of fresh-water ice). Safe navigation through sea-ice fields is hindered by both multiyear ice and icebergs. Hence, it is necessary to distinguish between the different ice types.

Although sea-ice surveillance can be performed using air-borne and satellite borne radars, the work presented herein is concerned with the classification of the returns of a surface-based radar system; specifically a noncoherent dual-polarized, *Ku*-band (16.5 GHz) radar. Data collection was performed on the northern tip of Baffin Island, Canada in a joint project between the *Communications Research Laboratory* (*CRL*), *McMaster University*, and the *Department of Fisheries and Oceans* (*DFO*), Canada. Since the radar is surface-based, in addition to the three ice types described above, there will be a fourth category, that of shadows cast by objects (usually icebergs) which extend above the height of the radar antenna. The only pre-processing performed was range-compensation.

Since the radar system employed is dual-polarized, two-images of the sea-ice field are obtained, one from the like-polarized channel and a second from the cross-polarized channel. Hence, it is necessary to employ a two-input classifier. In this paper we summarize the results of classifying the radar returns using a traditional *Bayesian* (*Gaussian*) classifier [1] and two neural classifiers, a multilayer perceptron classifier trained using the back-propagation algorithm [2], and a *Kohonen* feature map [3]/learning vector quantization (LVQ) [4] classifier. The classifiers were trained using 450 data pairs from each of the four classes and then tested using a different data set of the same size. A scatterplot of a subset of the sea-ice data is shown in figure 1.

II. Bayesian (Gaussian) Classifier

The first classifier, a parametric *Gaussian* classifier [1], is a well-known classifier. The results of this classifier will provide a benchmark for the neural classifiers. In the classification problem, we are given a vector, $\mathbf{x}$ (in this case a two-element vector), and must decide to which class it belongs (in this case 1 out of 4 possible classes). Using classical *Bayesian* theory, and assuming a constant loss function, the problem reduces to determining the class, i, with the maximum discriminant function,

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\mathbf{m}_i)\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\mathbf{m}_i) - \frac{1}{2}\log|\boldsymbol{\Sigma}_i|. \tag{1}$$

The mean, $\mathbf{m}_i$, and covariance, $\boldsymbol{\Sigma}_i$, are estimated from the training samples for each class, i, using,

$$\mathbf{m}_i = \frac{1}{N}\sum_{k=1}^{N}\mathbf{x}_k, \quad \text{and} \tag{2}$$

$$\boldsymbol{\Sigma}_i = \frac{1}{(N-1)}\sum_{k=1}^{N}(\mathbf{x}-\mathbf{m}_i)(\mathbf{x}-\mathbf{m}_i)^T, \tag{3}$$

where N is the number of training points available for each class (in this case 450).

The decision regions formed by the two-dimensional *Gaussian* classifier using the data of figure 1 are shown in figure 2. The vertical decision boundaries between shadows and first-year ice and between first-year ice and multiyear ice indicate that only the like-polarized channel is needed to distinguish between these two classes. However, the curved decision boundary between multiyear ice and icebergs indicates that the cross-polarized channel is needed to successfully distinguish between these classes. The results of applying the *Gaussian* classifier to the test data are shown in table 1. The classifier has an average classification accuracy of about 82.0%.

III. Multilayer Perceptron Classifier

Multilayer perceptron neural networks, trained using the back-propagation algorithm [2], are becoming increasingly more common for non-parametric classification. In this particular experiment, one-hidden layer networks were trained using the data of figure 1. Each network had 2 inputs nodes (one each for the like-polarized and cross-polarized channels), and 4 output nodes (one each for the 4 possible classes). Several different networks were trained, consisting of from 1 to 50 nodes in the hidden layer. Sigmoidal nonlinearities were used in each node, the learning rate constant was set to 0.5 and the momentum constant was set to 0.7. Convergence, in the sense that the mean-squared-error per output node has reached a steady-state value, is generally achieved after presenting the network with a total of 100,000 input points. The networks were trained such that the desired output of a given output node is 1 if that particular node corresponds to the current input class, and 0 otherwise. In testing the trained network, the output node showing the greatest activation was deemed to be the decision of the network. The results of applying the different sized classifiers to the test data are shown in table 1. The average classification accuracy reached a maximum of 82.6%.

The decision regions formed by a multilayer perceptron classifier with 6 hidden nodes in the hidden layer are shown in figure 3. The decision curves are very similar to the corresponding curves formed by the *Gaussian* classifier of figure 2. Again, it can be seen that only the decision boundary between multiyear ice and icebergs require the cross-polarized channel; the like-polarized channel alone can distinguish between the other classes.

IV. Kohonen Feature Map/Learning Vector Quantization Classifier

The final classifier consists of applying a *Kohonen* self-organizing feature map [3] to determine a good initial state for a LVQ classifier [4]. The *Kohonen* algorithm assumes a network in which all nodes are connected to the input vector, $\mathbf{x}$. Each node, i, stores a weight vector, $\mathbf{w}_i$, of the same dimension as the input vector. Following the rules of the *Kohonen* algorithm, each node computes,

$$\eta_i = ||\mathbf{x} - \mathbf{w}_i||_2, \tag{4}$$

the *Euclidean* distance between the input vector, $\mathbf{x}$, and the node weight, $\mathbf{w}_i$. Following this computation, a neighbourhood of nodes around (and including) the node which computes the smallest *Euclidean* distance is modified according to the rule:

$$\mathbf{w}_i = \mathbf{w}_i + \alpha(\mathbf{x} - \mathbf{w}_i), \tag{5}$$

where α is a small learning constant (set to 0.05 in this simulation). The neighbourhood typically begins large (to include half of the nodes in the network), and decays with time. The procedure continues with the next input vector.

Once the *Kohonen* algorithm has converged, all training vectors are once again input to the network, and class labels are assigned according to a majority vote to which class most often excites each node. This labelled map is then used as a starting point for LVQ learning. For the LVQ algorithm, the input vectors, $\mathbf{x}$, are once again presented to the network and the closest node, c, is determined using (4). The node weights of this node are modified according to,

$$\mathbf{w}_c = \begin{cases} \mathbf{w}_c + \alpha(\mathbf{x} - \mathbf{w}_c), & \text{if } \mathbf{x} \text{ and node c belong to the same class;} \\ \mathbf{w}_c - \alpha(\mathbf{x} - \mathbf{w}_c), & \text{if } \mathbf{x} \text{ and node c belong to different classes.} \end{cases} \tag{6}$$

All other nodes are left unmodified. Upon completion of LVQ learning, the unknown input vectors are applied to the network, and the class associated with the node computing the minimum distance according to (4) is the decision of the network.

Several different experiments were performed with the *Kohonen*/LVQ classifier, using different numbers of nodes, with one-dimensional networks ranging from a *10-by-1* map through to a *100-by-1* map, and two two-dimensional networks consisting of a *10-by-10* configuration and a *5-by-20* configuration. Convergence of the network using the *Kohonen* algorithm was achieved after inputting 10000 sample points (alternating between

the four possible classes) and training of the LVQ required a further 10000 iterations. It was found that networks composed of at least 20 nodes, in either the one-dimensional or two-dimensional configuration, performed nearly as well as the other two classifiers. The classification accuracies of the different networks are found in table 1. The average accuracy of the classifier peaked at 81.9%.

The decision regions formed by the *50-by-1 Kohonen*/LVQ map for the data of figure 1 is shown in figure 4. The points indicated on the map represent the node locations and labelled class type. The lines are drawn equidistant between nodes of different classes. The decision regions of this classifier closely match those of the previous two classifiers. Again, the cross-polarized channel is needed only to distinguish between multiyear ice and icebergs.

Table 1. Summary of Classifier Performance

	Percent Correct				
Classifier	Shadows	First-Year Ice	Multiyear Ice	Icebergs	Average
Gaussian Classifier	93.3%	84.2%	72.7%	77.8%	82.0%
Multilayer Perceptron Classifiers					
1 Hidden Unit	94.4%	80.7%	51.8%	85.1%	78.0%
3 Hidden Units	92.0	86.2	72.7	74.2	81.3
6 Hidden Units	91.8	86.2	73.6	78.7	82.6
12 Hidden Units	92.2	85.3	74.7	77.8	82.6
20 Hidden Units	90.7	88.0	73.8	77.8	82.6
Kohonen/LVQ Classifiers					
10-by-1 Map	82.7%	96.7%	66.7%	74.9%	80.2%
20-by-1 Map	91.8	89.6	71.6	74.9	81.9
50-by-1 Map	92.2	86.1	74.7	73.3	81.7
100-by-1 Map	91.6	86.0	72.7	74.2	81.1
10-by-10 Map	90.2	86.7	74.7	74.0	81.4
5-by-20 Map	91.8	85.8	75.8	73.3	81.7

V. Summary and Conclusions

Three classifiers have been applied to the sea-ice problem. Each classifier has produced similar decision regions and showed similar classification performance. Since the *Gaussian* classifier achieved a classification accuracy similar to that of the other two neural (and non-parametric) classifiers, we can conclude that the ice data are closely approximated by *Gaussian* distributions. Based upon this data, the choice of a specific classifier to process large sea-ice images in real-time will depend upon the efficiency of implementation of the classifier. Current work is aimed towards the real-time implementation of neural classifiers [5]. Another important conclusion drawn from this research is that the like-polarized channel contains information about the different forms of ice that is adequate for its classification into shadows, first-year ice and a combined class consisting of multiyear ice and icebergs; the addition of a cross-polarized channel is particularly useful in distinguishing between multiyear ice and icebergs.

VI. References

1. R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.
2. D.E. Rumelhart, G.E. Hinton and R.H. Williams, "Learning internal representations by error propagation." In D.E. Rumelhart and J.L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructures of Cognition.* Vol. 1: it Foundations, pp. 318-362, MIT Press, Cambridge, MA., 1986.
3. Teuvo Kohonen, *Self-Organization and Associative Memory*, 2nd ed., Springer, Berlin, 1988.
4. Teuvo Kohonen, György Barna and R. Chrisley, "Statistical Pattern Recognition with Neural Networks: Benchmarking Studies", in IEEE International Conference on Neural Networks, July, 1988.
5. Richard Mann and Simon Haykin, "A Parallel Implementation of Kohonen Feature Maps on the Warp Systolic Computer", submitted for publication to the International Joint Conference on Neural Networks (IJCNN-90-WASH), Washington, D.C., January 1990.

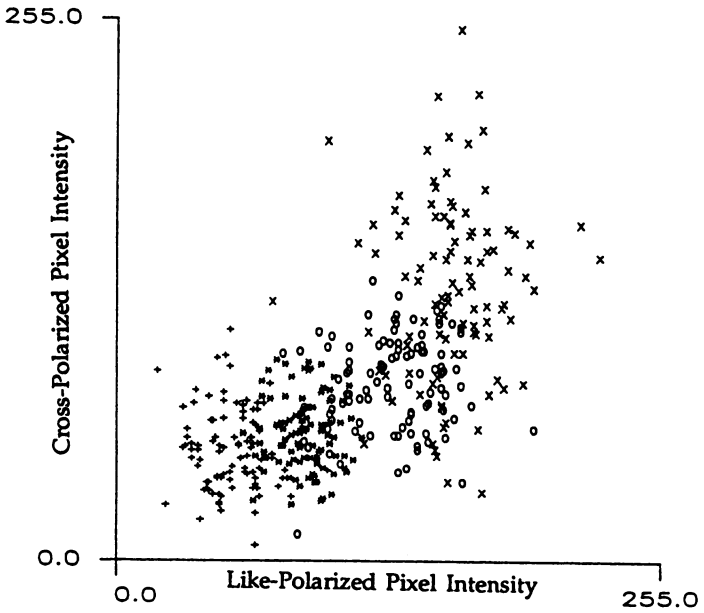


Figure 1. Scatterplot of a subset of the sea-ice data. (+ denotes shadows, * denotes first-year ice, o denotes multiyear ice and x denotes icebergs)

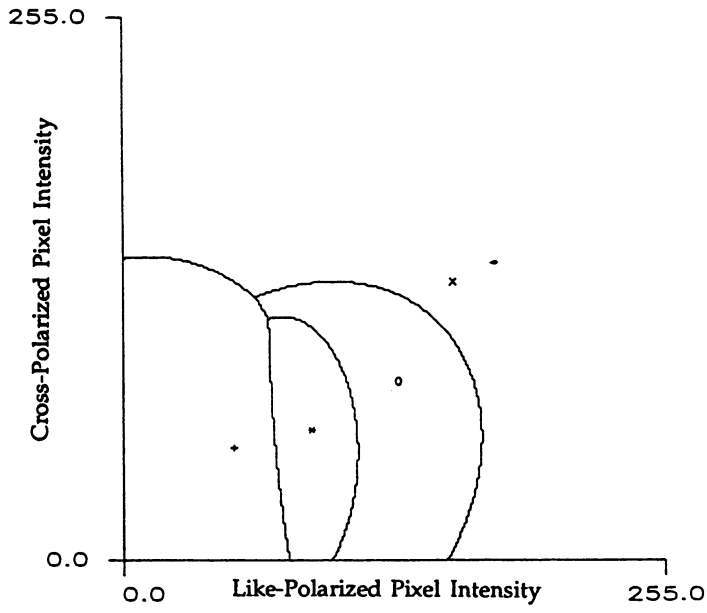


Figure 2. Decision regions formed by the Gaussian classifier. (+ denotes the centroid of shadows, * denotes the centroid of first-year ice, o denotes the centroid of multiyear ice, and x denotes the centroid of the icebergs).

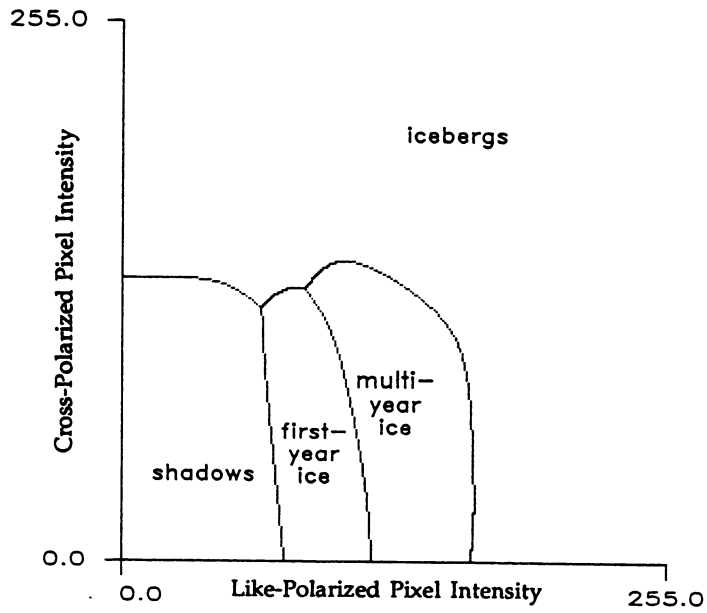


Figure 3. Decision regions formed by a multilayer perceptron classifier with 6 hidden nodes.

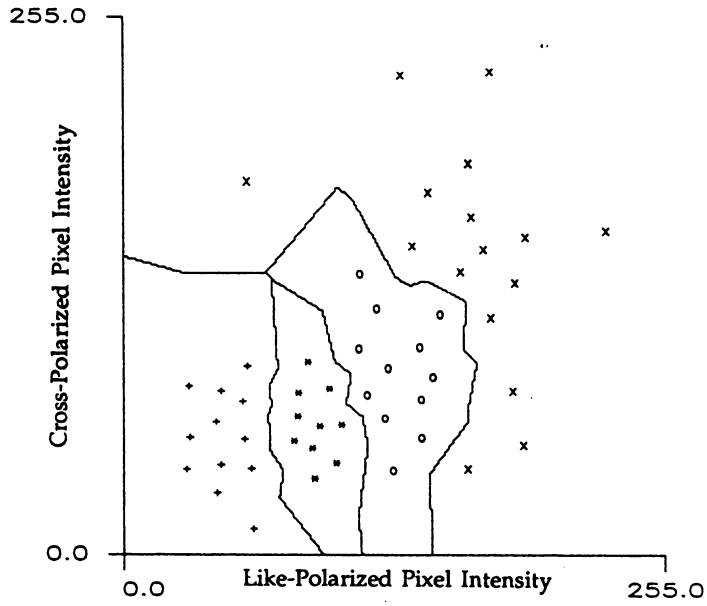


Figure 4. Decision regions formed by the 50-by-1 Kohonen/LVQ classifier. (+ denotes a node location which corresponds to the shadow class, * denotes a node location which corresponds to the first-year ice class, o denotes a node location which corresponds to the multiyear ice class, and x denotes a node location which corresponds to the iceberg class)

Neural Tree Structured Vector Quantization

Eric Wan Paul Ning Bernard Widrow
Stanford University Department of Electrical Engineering, Stanford, CA 94305-4055

Abstract

In this paper we present a new method for vector quantization design and implementation using a tree structured artificial neural network. Each node in the tree consists of a neural network which successively learns to partition the input space. While the structure is evaluated for use in image compression, the NTSVQ can be generalized as a means of data compression wherever vector quantization is appropriate. In addition, the adaptive tree structure should have many applications in related pattern recognition problems.

1 Introduction

In recent years, vector quantization (VQ) has become an increasingly popular form of data compression (see [1] for a review). A traditional vector quantizer consists of an encoder, decoder, and codebook as shown in Figure 1a. Given a vector to be quantized, the encoder finds in a codebook the codeword which minimizes the distortion between it and the vector to be quantized. The binary index of the codeword, called the channel symbol, represents this vector and can be processed, transmitted, or stored. A decoder having an identical codebook uses the channel symbol to locate the corresponding codeword for the reproduction vector. For a codebook of k codewords and a vector dimension of N, the bit rate is defined as $log_2(k)/N$ bits per vector component. The codewords themselves can be stored to an arbitrary accuracy and do not figure into the bit rate. The process of encoding and decoding is straightforward. The more difficult task, however, is the design of the codebook which minimizes the overall distortion of the data to be quantized. The standard VQ design technique, the LBG algorithm [2], utilizes training vectors to establish the codebook.

In a tree structured VQ, encoding is achieved through a series of binary decisions in which the input space is successively partitioned. Referring to Figure 1b, the input is first evaluated by the root node, which is at the top. Depending on the result of this initial test, the input vector is then passed to either the left or right child of the root. Each child, in turn, is another node which makes binary decisions to continue the classification process. Each node in a VQ tree actually consists of a tiny VQ with a codebook of size two. The binary decision at the node consists of determining which of the two reconstruction vectors incurs the smaller error. The effective codebook of the tree VQ is the union of the codebooks of the lowest level nodes. For a binary tree with L levels, the total number of classes or codewords is 2^L. Reconstruction vectors used by any node above the bottom do not appear in the final codebook but only determine the search path down the tree. This binary path through the tree corresponds directly to the actual channel symbol. Since only one comparison is needed at each level, the computations required to search a tree grow only with the logarithm of the codebook size (a great savings over the fullsearch VQ which grows linearly with codebook size).

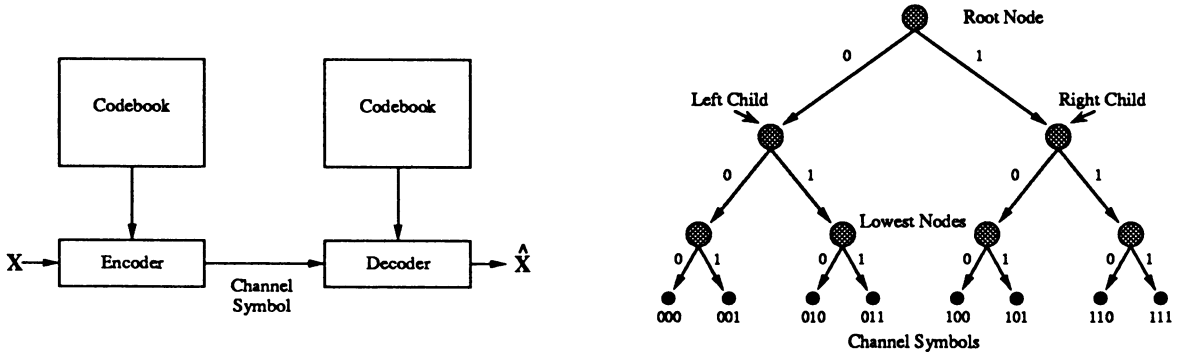


Figure 1: (a) Traditional VQ System, (b) Binary Decision Tree Encoder

The process of quantizing an N-dimensional vector amounts to partitioning N-dimensional Euclidean space into k partitions. The centroid of each partition represents each codeword. A vector falling into one of

these partitions is represented by the centroid of the partition. The error as a result of quantization can be viewed as the distance between the actual vector and the codeword. An example of a 2-dimensional vector space partitioned by a binary decision tree is shown in Figure 2a.

2 Learning in the Decision Tree

We now present a method of implementing a tree structured VQ based on neural network techniques. Each node in the decision tree of Figure 1b contains a small adaptive neural network capable of binary decision making. The architectures of the nets are the same for all the nodes. The weights are different from node to node, depending on learning experience.

The typical neural network for each node is shown in Figure 2b, The simple 2-layer neural net can be considered as a two-codeword vector quantizer. The first layer is an encoder and the second layer is a decoder. The values of the encoder weights determine the partitioning of the input space. For a single layer of encoder weights, this partition is a simple hyperplane at each node. While additional encoder layers would allow for non-linear partitions, the optimal partition for a two codeword codebook is always a hyperplane. The sign of the sigmoid output at the hidden channel layer determines the binary decision for the node. The collective binary decisions down the tree formulate the full channel symbol.

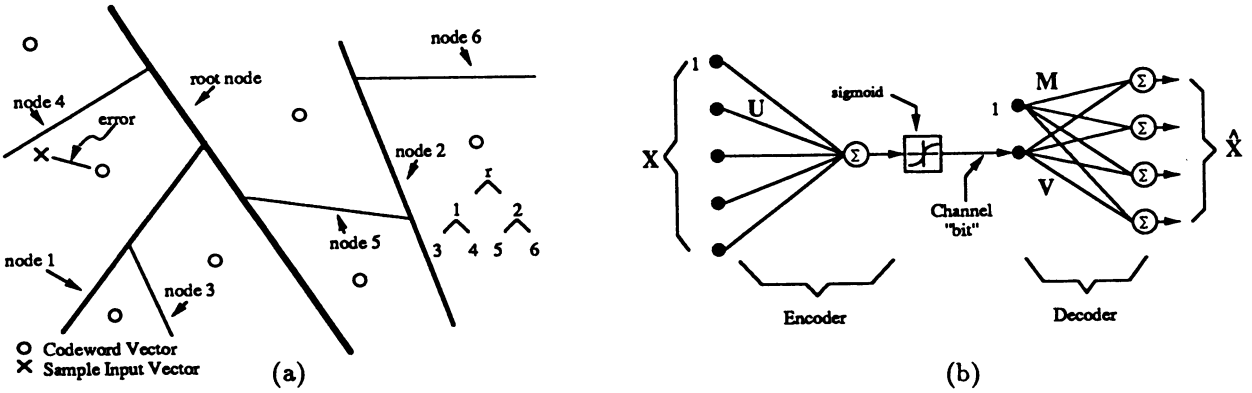


Figure 2: (a) Tree Partitions, (b) A Node's Internal Neural Network

Node Adaptation Referring to Figure 2b, training is effected using the backpropagation algorithm [3] to cause the decoder output $\hat{\mathbf{X}}$ to be a least squares reconstruction of the input $\mathbf{X}$. The sigmoid is necessary for backpropagation. However, after convergence of the two layers, the sigmoid must be replaced by a hard binary quantizer. This produces an actual bit at the encoder output and drives the second layer with a binary signal. The decoder weights are then readapted using simple LMS [4] to once again minimize the mean square error. With the encoder weights fixed, and the partitions set, the decoder outputs will converge to two possible reconstruction vectors $\hat{\mathbf{X}}$ which will become the centroids of each partition. Thus, the system of Figure 2b is a self-contained self-organizing classifier which is used here as a binary vector quantizer. The tree itself forms a more sophisticated vector quantizer that also learns without supervision.

Tree Adaptation Since nodes will only need to make decisions on inputs which are passed to them from higher nodes, it makes sense to adapt their weights on only those vectors. Thus to adapt the entire tree we use the following training cycle :

1. Adapt the root node to the current vector in the training set.
2. Depending on the side to which the root node classifies the input vector (based on the sign of the channel bit), select either the left or right child to receive the input.
3. Perform steps 1 and 2 for the selected node and continue until the bottom of the tree is reached.
4. Repeat the cycle for a new input chosen randomly from the training set.

In this way, for any given input training vector, only L nodes in the tree are adapted corresponding to that input's classification path. Multiple passes through the training set are performed until the weights converge or a suitable VQ distortion level is achieved.

3 Analysis

A study of an individual node reveals how partitions are formed and provides insight into the nature of the reconstruction vectors. Referring to Figure 2b, let the input $\mathbf{X} = [1, x_1, ..., x_N]^T$ be a sample vector plus bias. $\mathbf{U} = [u_0, u_1, ..., u_N]^T$ is the encoder weight vector, $\mathbf{V} = [v_0, v_1, ..., v_N]^T$ is the decoder weight vector, and $\mathbf{M} = [m_0, m_1, ..., m_N]^T$ is the decoder bias weight vector. The output reconstruction vector is then

$$\hat{\mathbf{X}} = \mathbf{V} f(\mathbf{X}^\mathbf{T}\mathbf{U}) + \mathbf{M} \tag{1}$$

and the error due to reconstruction is

$$\mathbf{E} = \mathbf{X} - \hat{\mathbf{X}} = \mathbf{X} - \mathbf{V} f(\mathbf{X}^\mathbf{T}\mathbf{U}) - \mathbf{M} \tag{2}$$

For a performance measure we consider the total mean squared error $\xi = E[\mathbf{E}^\mathbf{T}\mathbf{E}]$. The optimal solution for the weight vectors which minimizes the overall mean squared error can be found by setting the gradients of ξ with respect to the various weight parameters to zero and solving a set of nonlinear matrix equations. Unfortunately, no closed form solution exists. However, by assuming operation in the linear region of the sigmoid (i.e. $f(x) \approx x$), several interesting results may be obtained. With this assumption

$$\xi = TR[\mathbf{R}] + 2\mathbf{U}^\mathbf{T}\bar{\mathbf{X}}\mathbf{V}^\mathbf{T}\mathbf{M} + \mathbf{U}^\mathbf{T}\mathbf{R}\mathbf{U}\mathbf{V}^\mathbf{T}\mathbf{V} - 2\mathbf{M}^\mathbf{T}\bar{\mathbf{X}} + \mathbf{M}^\mathbf{T}\mathbf{M} - 2\mathbf{U}^\mathbf{T}\mathbf{R}\mathbf{V} \tag{3}$$

where $\mathbf{R} = E[\mathbf{X}\mathbf{X}^\mathbf{T}]$. By taking the gradient with respect to each of the weight vector, $\mathbf{U}, \mathbf{V}$, and $\mathbf{M}$, it can be shown that the following conditions must hold at a minimum

$$\mathbf{M} = \bar{\mathbf{X}} \qquad \mathbf{U} = \frac{\mathbf{V}}{\mathbf{V}^\mathbf{T}\mathbf{V}} \tag{4}$$

$$\mathbf{R}\mathbf{V} = \lambda\mathbf{V} \qquad \xi_{min} = TR[\mathbf{R}] - \mathbf{U}^\mathbf{T}\mathbf{R}\mathbf{V} \tag{5}$$

This implies that whenever $\mathbf{V}$ is an eigenvector of $\mathbf{R}$, and the conditions in (5) are met, we are at a local minimum. Thus there are up to N local minima, where N is the dimension of $\mathbf{X}$. The optimal global solution is achieved when $\mathbf{V}$ is the maximal eigenvector of $\mathbf{R}$. The globally minimum expected mean squared error is then

$$\xi_{min} = TR[\mathbf{R}] - \lambda_{max} \tag{6}$$

and the optimal partition defined by $\mathbf{X}^\mathbf{T}\mathbf{U} = 0$ is the hyper-plane normal to the maximal eigenvector of $\mathbf{R}$. While this result is only valid for the linear case, it should reflect how partitions are initially formed during the early stages of adaptation with the sigmoid. The nature of the final partitions with the full non-linearities involved is still under investigation.

4 Applications To Image Compression

For image compression, the input vectors to the VQ system are obtained by dividing up images into sets of non-overlapping square blocks of pixels. Thus, each image provides a large number of input vectors. Training vectors can be obtained from several representative images.

A sample image compressed to 2 bits per pixel with a 2x2 block size is shown in Figure 3. The original 8 bit per pixel image, with pixel intensities ranging from 0 to 255, is reconstructed with a root mean square error (per pixel) of 6.17. The compression ratio is 4:1. The reconstructed image is almost as good as the original.

5 Variations

Philanthropic Trees Traditional decision trees are often described as *greedy* . A parent is designed to do the best it can without any consideration of how its child is doing. The overall performance of a tree may be improved if at any given intermediate node a sacrifice in performance is allowed based on knowledge of how its decision affects its children's performance. Neural TSVQ's offer the unique potential of being able to adapt all nodes in a fully interdependent manner. The error of a child can be weighted into the error of its parent. The parent which attempts to minimize its own mean squared error is now directly influenced by how well its children are doing. The parent, which determines the subset of the training set its children see, in turn affects the expected mean squared error of the children. The details of philanthropic trees are left to [6]. Experiments with error passing have shown improvements in MSE on the order of 10 percent. It is our view that various error passing schemes which allow one to adapt the tree as a fully interdependent unit will ultimately provide many advantages over traditional decision trees.

(a) (b)

Figure 3: **Lena** (a) Original (b) Compressed

Pruned Trees Additional improvements to Tree VQs can be obtained by using variable length trees. One can prune selected nodes to cut the tree short along paths which include inefficient nodes. This lowers the average bit rate of a coder without undue loss in mean square error. Equivalently, this procedure can yield lower distortion for any given average bit rate by first growing a uniform tree at a higher bit rate and then pruning back to the desired rate. Pruning results in trees with classification paths similar in nature to entropy based codes and can outperform even full search quantizers. An "optimal" pruning technique can be found in [5].

6 Conclusion

In this paper we have presented a new method for vector quantization which combines decision tree techniques with neural network techniques. While we have concentrated on data compression, the neural tree structure can be applied to a variety of decision problems. Inputs to the network may, for example, correspond to segments of speech. The tree would develop in such a way that decision paths identify phoneme classes. A study of this structure for use in character recognition is already under investigation and will be reported in a forthcoming paper. Finally, other possible variations include alternate error passing algorithms, the inclusion of additional layers in the encoder and decoder, and the use of teacher directed learning. A detailed discussion of this structure can be found in an expanded version of this paper [6].

References

[1] R.M. Gray, "Vector quantization," *IEEE ASSP Magazine*, vol. 1, no. 2, pp. 4-29, April 1984.

[2] Y. Linde, A. Buzo, and R.M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Comm.*, vol. COM-28, pp. 84-95, Jan. 1980.

[3] D.E. Rumelhart, J.L. McClelland, and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, The MIT Press, Cambridge, MA, 1986.

[4] B. Widrow and M.E. Hoff, "Adaptive switching circuits," *1960 IRE WESCON Convention Record*, Part 4, pp. 96-104, August 1960.

[5] P.A. Chou, T. Lookabaugh, and R.M. Gray, "Optimal pruning with applications to tree-structured source coding and modeling," 1987. *IEEE Trans. Information Theory*, to appear.

[6] E.A. Wan, P. Ning, "Neural Tree Structured Vector Quantization" 1988, *Stanford report.*

This research was sponsored by SDIO Inovative Science and Technology Office and managed by ONR under contract #N00014-86-K-0718, and by Rome Air Development Center under contract #F30602-88-D-0025, and subcontract E-21-T22-S1.

Application of Neural Networks to Pulse–doppler Radar Systems for Moving Target Indication

Chia–Jiu Wang & Chwan–Hwa Wu[1] *& Rodger E. Ziemer*
Department of Electrical and Computer Engineering
University of Colorado at Colorado Springs
Colorado Springs, CO 80933

Department of Electrical Engineering[1]
Auburn University
Auburn, AL 36849

Abstract:

A three–layer neural network has been used to detect moving targets in severely cluttered environments for pulse–doppler radar applications. Significant performance improvements have been achieved as compared with the conventional fast Fourier transform method. An example with 16 input signal samples per moving target is used in this paper for demonstration.

1. Introduction

The basic function of pulse–doppler radars is to detect a target by transmitting a pulse of radio frequency energy and receiving a portion of the reflected energy from the target. Medium pulse repetition frequency (PRF) radar systems can be designed to have a greater degree of tactical flexibility in target detection by combining the desirable features of both low and high PRF radars [1,2]. The medium PRF radar systems use time discrimination for ranging and frequency discrimination for range rate calculation. The clutter foldover effect caused by ambiguous range makes the medium PRF radar require greater clutter rejection capability.

In this paper we apply three–layer neural networks for moving target detection in severe clutter environments. The performance of the neural network is compared with that of the conventional filter bank method. The neural network shows much better performance especially in highly cluttered environments.

2. Moving Target Indication

The purpose of moving target indication radars is to reject signals from fixed unwanted targets, such as buildings, hills, and trees, and retain signals for detection of the moving targets such as aircrafts, ships, missiles etc.

The target doppler frequency F_t can be expressed as $F_t = (2V_t/\lambda)\cos\psi_t$ where V_t is the target velocity, ψ_t is the angle between the target–velocity vector and the radar–target line of sight, λ is the radar wavelength. Figure 1 shows a typical medium PRF radar frequency spectrum in the presence of ground clutter. The major function of a moving target indication (MTI) radar is to detect the target doppler shift from severe clutter environments.

2.1 Conventional MTI techniques

Adaptive pulse cancellers process the received signals in time domain. Based on the clutter information the adaptive pulse canceller can change the notch frequency to block clutter from receiving. The adaptive pulse canceller can only distinguish whether this is a moving target or clutter. The target velocity can not be obtained from pulse cancellers directly. In many application situations the target velocity is required. Therefore in advanced radar systems, filter banks are used to further process the output signals from pulse cancellers.

Another technique is to use doppler filter banks directly without using pulse cancellers. Doppler filter

bank technique is used in many modern radar systems to detect not only moving targets but also their associated radial velocities. Fast Fourier transform (FFT) is the most commonly used technique to construct filter banks for target velocity detection and clutter rejection. Based on the sampling theory, the maximum detectable target velocity is 1/2 * PRF. A filter bank covering frequency from —1/2 * PRF to 1/2 * PRF can be constructed from a direct application of fast Fourier transformation. There are many other ways to form filter banks, such as using a set of narrow bandpass filters in parallel. But the most economical and reliable one is to use FFT.

2.2 Limitations in Conventional MTI techniques

The bandwidth of each filter in the filter bank determines the accuracy of target velocity. The number of points used in the FFT processing has to meet the radar system design requirements. The number of stages used in the FFT is $\log_2 n$, n is the number of points and has to be an integer power of 2. The FFT algorithm has reduced the computation load tremendously as opposed to discrete Fourier transform. But it still takes $\log_2 n$ processing steps to finish one complete transformation. Moreover, the spectral leakage effect due to the finite window length in observation will degrade the signal strength in the mainlobe and enhance the signal strength in sidelobes. To overcome this spectral leakage, i.e. to reduce the signal magnitude in sidelobes, many windowing schemes are used such as Hanning, Hamming, Blackman, and Bartlett windows [3]. All these windows can greatly reduce the signal magnitude in the sidelobes, but the mainlobe frequency bandwidth has been increased. In other words, the supression of sidelobe signals sacrifices the resolution due to the widening of mainlobe bandwidth.

3. Neural Network Moving Target Indication

A three—layer neural network has been trained to perform as a moving target indicator. The back propagation training algorithm [4,5] has been used to train a three—layer neural network. This three—layer neural network has 16 neurons in the input layer, 24 neurons in the hidden layer, and 16 neurons in the output layer. For each input signal (pattern), two corresponding output neurons will be activated to generate output signals with magnitude close to 1, and the rest of neurons in the output layer generate almost zero output. The output neurons are numbered from —8,—7,...,0,...,6,7, then training pattern 1 will activate output neurons —1 and 1, training pattern 2 will activate output neurons —2 and 2, and so forth. Training pattern 8 is used to activate output neuron 0 only. Physically this three—layer neural network is trained to perform as a set of bandpass filters with different center frequencies. The outputs of neuron n and —n are corresponding to a bandpass filter with center frequency at n/16 * sampling frequency, where n varies from 0 to 7. For a pulse—doppler radar system, the sampling frequency is the PRF.

3.1 Performance Comparison

The performance of the three—layer neural network is tested with randomly sampled input signals to represent different moving targets with different velocities. The white Gaussian noise has been used to simulate the highly cluttered environments. For the purpose of comparison, the outputs from the FFT filter bank method have also been calculated. Figure 2 shows the output comparison between the neural network and the FFT for signal—to—noise ratio (SNR) = 5 db and no noise cases. Figure 3 presents a complete comparison for eight different target doppler signals with SNR = 0 db. Surprisingly good performance is achieved by the three—layer neural network. The three—layer neural network performs much better in clutter suppression than the conventional FFT filter bank method. Another unique feature we found from the neural outputs is that the mainlobe width remains unchanged, implying that the target velocity detection resolution will not be changed after the suppression of signal magnitudes in sidelobes. This unique feature can not be achieved through the windowed FFT method (such as Blackman, Hamming etc) that can only suppress the signal magnitude in the sidelobes at the expense of widening the mainlobe [3]. (i.e. detection resolution is reduced.)

4. Discussions and Conclusions

The advent of new massively parallel neural network hardware offers the opportunity for implementing special purpose detection processing which has the potential for considerable performance gains. This paper has shown that a three—layer neural network can be trained to detect moving targets in very cluttered environments. The three—layer neural network performs much better than the conventional FFT method in several aspects: 1) suppress the sidelobe clutter significantly without widening the mainlobe bandwidth, 2) the number of input signal samples can be any integer numbers, 3) only two stages are needed for the completion of computation regardless of the number of input signal samples. (many test results cannot be included due to space limitation)

References

[1] M.E. Skolnik,Radar Handbook, Mcgraw—hill book company, 1970
[2] S.A. Hovanessian, Radar System Design and Analysis, Artech House, Inc., 1984.
[3] A.V. Oppenheim, R.W. Schafer Discrete—time Signal Processing, Prentice Hall 1989.
[4] R.Hecht—Nielsen, "Theory of the Backpropagation Neural Network," International Joint Conference on Neural Networks, Vol. 1, pp. 593—605, 1989.
[5] R.P. Lippmann,"An Introduction to Computing with Neural Nets," IEEE ASSP Mag., pp. 4—22, April 1987.

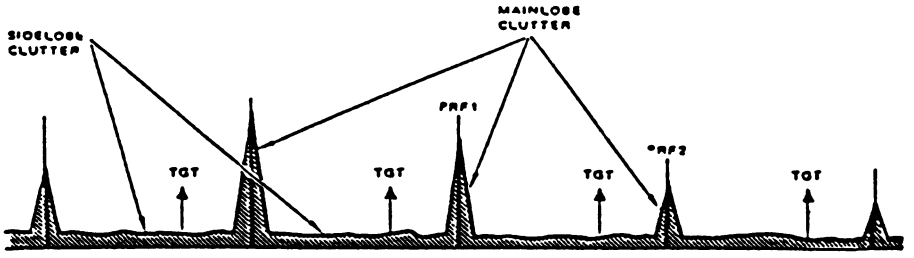


Figure 1 Medium PRF spectrum in the presence of ground clutter

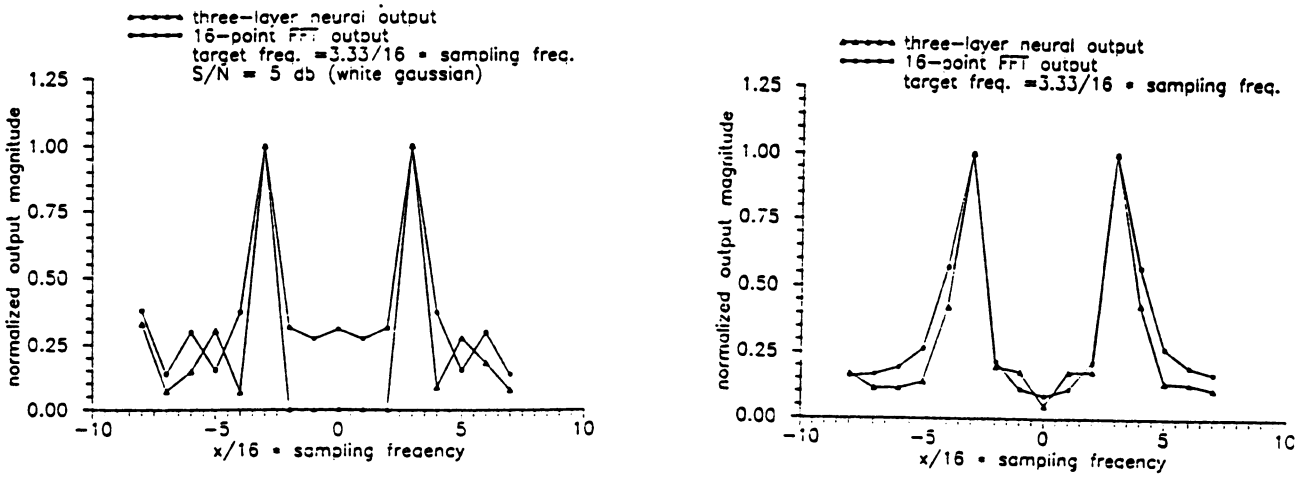


Figure 2 performance comparison between neural and FFT outputs for SNR=5 db (left figure) and zero noise (right figure) cases.

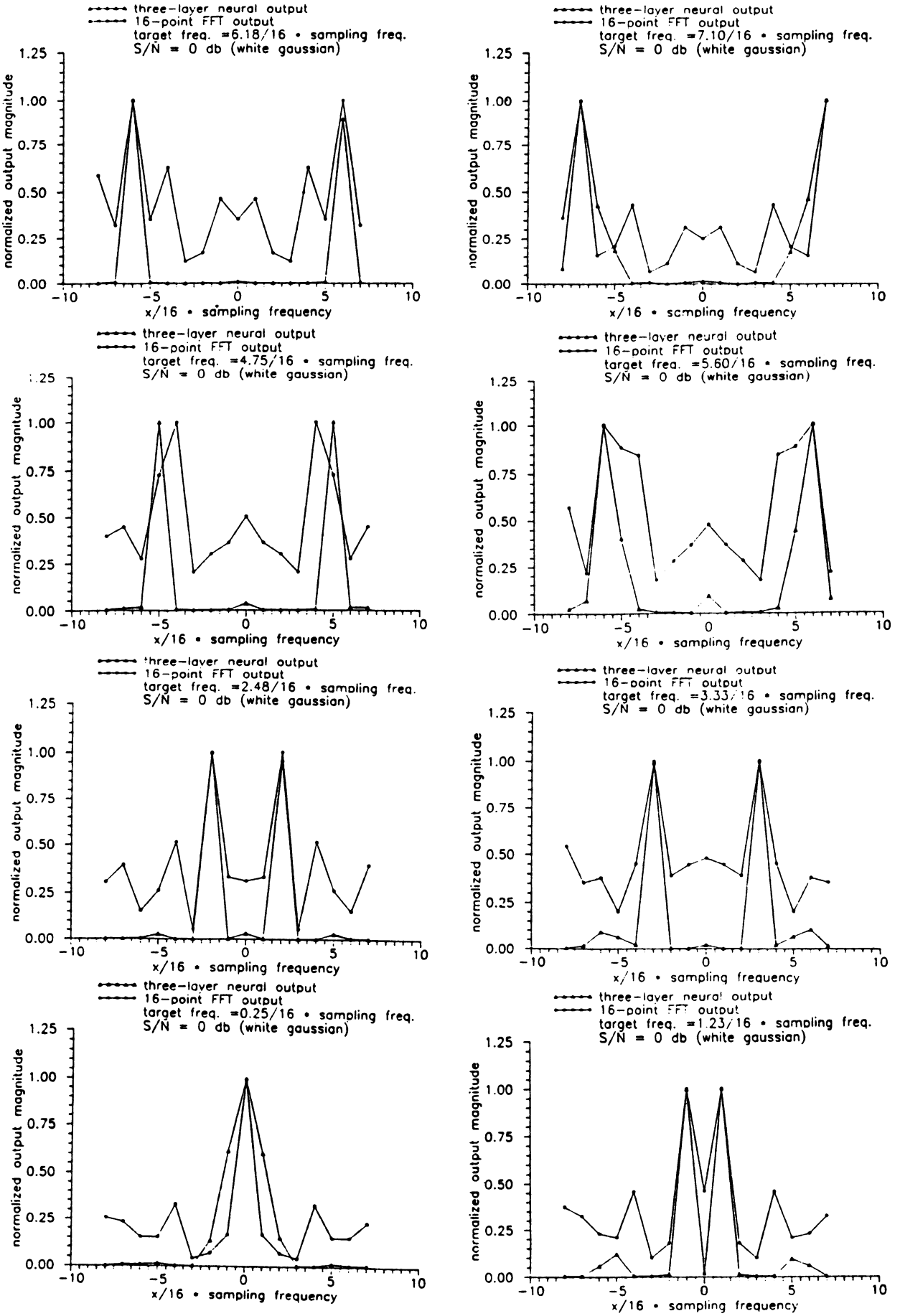


Figure 3 A complete comparison of neural and FFT outputs for SNR=0 db

A VLSI IMPLEMENTABLE HANDWRITTEN DIGIT RECOGNITION SYSTEM

L.C. Agba*, R. Shankar, A.S. Pandya***, and C. Naylor+**

***Department of Electrical Engineering**
****Department of Computer Engineering**
*****Center for Complex Systems**
Florida Atlantic University
Boca Raton, Florida 33431

+Department of Computer Science, University of South Florida

ABSTRACT The development of handwritten character recognition system has been an age old puzzle to researchers in the field. A simple hardware implementable technique that yields very high recognition rate using neural networks is herein discussed.

INTRODUCTION Artificial neural net (ANN) models have evolved to emulate the behavior of biological nervous systems in such tasks as image and speech recognition [1]. The brain performs a large number of operations in parallel with the aid of interconnections (synapses) of a large number of simple processing elements (neurons). Thus these ANNs incorporate parallel distributed processing with dense interconnections of simple computational elements.

Such a highly interconnected ANN model is not easily scaled for large number of nodes, since the area requirement increases as per $O(N^3)$, where N is the number of connections [2]. Two proposals have been made so that different ANNs can be implemented in VLSI without undue demand for interconnection area [2,3]. Bailey and Hammerstrom [2] suggested an architecture based on multiplexing of communication lines and information storage and passing. Akers and Walker [3] have proposed limited-interconnect, multi-layered perceptron like neural net architecture. A limited interconnect would limit the area set aside for communication links and hence would be easily expandable.

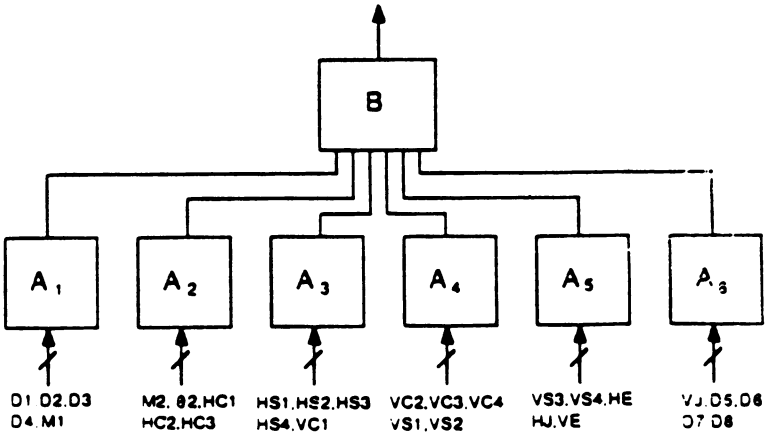
We have explored a limited-interconnect multiple-layer perceptron topology to overcome the interconnection problems. We have successfully trained network architectures for handwritten numerals (ZIP code information) with a maximum of eight inputs and eight outputs per node and nearest neighbor connections. Results on the (handwritten) zipcode database from the United States Postal Service Office of Advanced Technology are presented [4].

METHOD Several techniques such as Fourier transforms, stroke feature distribution and moments [5,6] exist for addressing orientation, translation, and scale invariancy in pattern classification. We have opted for a feature set extracted in a manner that is simplistic, and computationally more efficient. Starting from a binary image frame of a digit, we first skeletonize the image, and then extract a set of features from the image. Eight

parameters (D_i, M_i, and θ_i) are obtained by preprocessing the image using holon dynamics [7]. We define a septon as a group of nine holons (3x3) taken as a unit [7]. Here a holon is assumed to exist for each pixel and neighboring holons cooperate in image processing and information compression. D_i represents the total number of Septons with the ith mode excited. A certain mode is said to be excited if a direction vector exists in a 3x3 pixel area of the digit. M_i counts the number of type i septons with i modes and θ_i is the total angle subsumed by the septon of type i. These parameters provide for orientation independence [8].

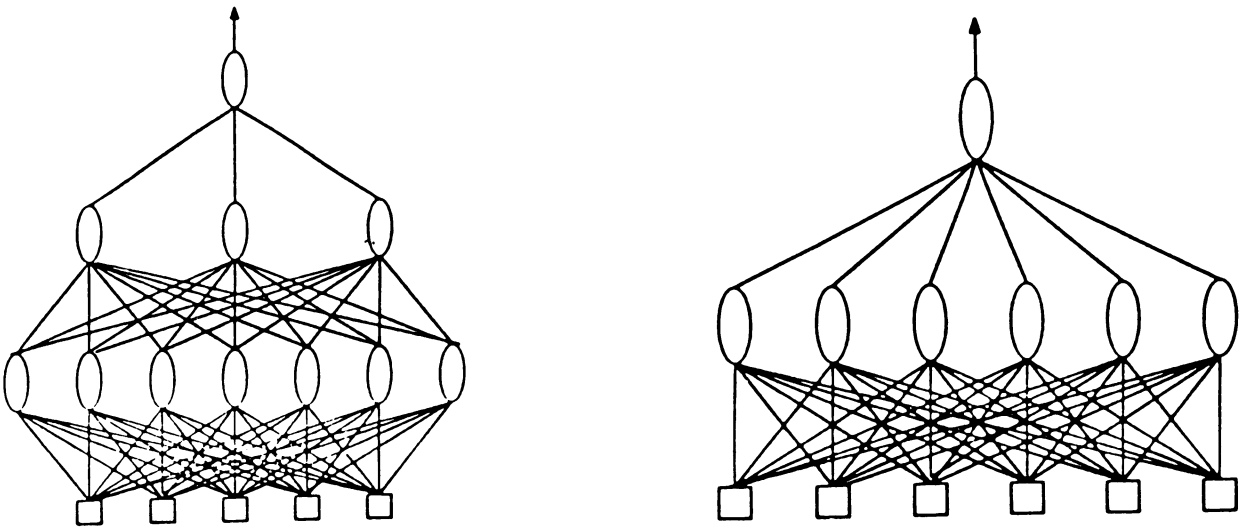
The remaining features in the set (see Fig. 2) account for character distortions. It was noted that most of the character distortions occurred near the image boundaries. So, a set of parameters were extracted from a sub image obtained by disregarding in the final computations 10-20% of the image near the four edges. For computation of horizontal parameters, regions near the vertical boundaries are disregarded and vice versa. HC parameters determine the number of crossings as the subimage is scanned in the horizontal direction from top to bottom. HC_i corresponds to the total number of times the number of crossing change in the subimage from j(j ≠ i) to i crossings, with the i crossings maintained for 10% of the subimage. HS records the sequence in which these HC_i occur up to a maximum of four entries. VC and VS define similar parameters in the vertical direction. HE and HJ (VE and VJ) record the number of end points and junctions, respectively, while scanning the image in the horizontal (vertical) direction.

NETWORK ARCHITECTURE Figure 1(a), represents A730, one of our hierarchical structures with 5 and 6 input perceptron modules. The features were subdivided into 6 groups of five features as shown in Figure 1(a). D_1 through D_4 were repeated as D_5 through D_8 respectively (Figure 1(b)). The outputs of these six subgroups formed inputs to the second higher level B module (Figure 1(c)). A730 thus has 396 wieghts which are locally connected. Two other networks used are of similar architecture with more inputs. One of them has 36 inputs while the other has 64 inputs.



(a)

Figure 1. A730 Network Architecture



(b)

A-Module Network Architecture

(c)

B-Module Network Architecture

Figure 1. A730 Network Architecture (continued)

SIMULATIONS We used the image database that the U.S. Postal Service has assembled to train and test various automatic digit recognition techniques [4]. This database contains approximately 2400 image frames of handwritten ZIP codes. SPIDER package routines among other routines were used to process the images. We have developed our algorithms using only the first 90 frames which yielded a total of 460 digits. Of these 460 digits, a maximum of about 50 digits, as listed below, were used to train the networks. The networks were trained with the back propagation algorithm. The number of iterations varied from about a few hundreds to 2000 depending on the setting of alpha and beta [1].

Table 1. Test results on 460 handwritten digits.

DIGIT	NUMBER IN TRAINING SET	TRUE POSITIVE	FALSE POSITIVE	TRUE NEGATIVE	FALSE NEGATIVE	PERFORMANCE (%)
0	12	90	7	359	4	97.6
1	10	24	3	430	3	98.7
2	28	51	11	393	5	96.5
3	51	35	12	412	1	97.2
4	40	29	2	427	2	99.1
5	35	20	12	424	4	96.5
6	28	45	9	402	4	97.2
7	35	50	8	401	1	98.0
8	15	46	7	401	6	97.2
9	30	37	3	415	5	98.3

RESULT Results for the performance of the networks on the handwritten digits are provided in table 1 above. Our true digit recognition rates of 96% or better are superior to those obtained on the same database by other researcheres who used various techniques [9]. One of the techniques used a 256 input perceptron that was trained with 391 digits 10,000 times and tested with 1173 digits. The better of their various techniques yielded 60-75% performance.

DISCUSSION We have adopted an incremental training technique for all the networks. Initial set included only one of a group of patterns in the training set. Since the digits are handwritten and come in all shapes and sizes, a well written digit and a badly distorted one might not be seen by the network as the same pattern. Consequently, the additions to our training set will come form the false positives and negatives.

CONCLUSIONS We have evolved a system that is simple, and has a high digit recognition rate. Our limited interconnect hierarchical perceptron classifiers are VLSI implementable. Cost and performance may be traded to yeild hardware solutions that range from a fully serial to a fully parallel (one network per digit) implementations. It is also our belief that the system is general enough for application to characters.

ACKNOWLEDGEMENT We would like to thank Dr. Timothy Barnum for allowing us to use the U.S. Postal Service database.

REFERENCES:

1. Lippmann, R.P., "An Introduction to Computing with Neural Nets," IEEE ASSP Magazine, pp. 4-22, April '87.
2. Bailey, J., and Hammerstorm, D., "Why VLSI Implementation of associative VLCNs require connection multiplexing," IEEE Int'l Conference on neural networks, 11, 11-173 to 11-180, July 1988.
3. Akers, L.A., and Walker, M.R., "A Limited-interconnect Synthetic Neural IC," IEEE Int'l Conference on neural networks, 11, 11-151 to 158, 1988.
4. United States Postal Service, Office of Advanced Technology, Technology Resource Dept., 475 L'Enfant Plaza, SW, Washington, D.C. 20260-8100.
5. Shirvaikar, M.V., and Musavi, M.T., "A stroke feature distribution scheme for the recognition of alphanumeric characters," Proc. of the 29th Southeastern symposium on system theory, 192-195, March 1988.
6. Cash, G.L., and Hatamian, M., "Optical character recognition by the method of moments," Computer Vision, Graphics & Image Processing, 39, 3, 291-310, 1987.
7. Shimizu, H., Yamaguchi, Y., Tsuda, I., and Yano, M., "Pattern recognition based on holonic information dynamics: towards synergetic computers," in H. Haken, ed., Complex Systems-Operational Approaches, Springer-Verlag: Berlin, 225-239, 1986.
8. Agba, L.C. and Shankar, R., "A novel method for pattern recognition with special application to alph-numeric characters," First Conference on recent advances in Robotics, FAU, Boca Raton, Florida , 50-54, 1988.
9. Pawlicki, T.F., Lee, D.S., Hull, J.J., and Srihari, S.N., "Neural network models and their application to handwritten digit recognition," IEEE Int'l Conference on neural networks, 11, 11-63 to 11-70, 1988.

Spatio-temporal vs. Spatial Pattern Recognition by the Neocognitron

Kunihiko Fukushima
Department of Biophysical Engineering
Faculty of Engineering Science
Osaka University
Toyonaka, Osaka 560, Japan

Abstract: The neocognitron was previously proposed as a model of visual pattern recognition in the brain. This paper mainly discusses merits and problems which will arise when trying to extend the neocognitron to recognize spatio-temporal patterns. As a first step to extending the neocognitron, a simplified model is proposed. In connection with this model, we concentrate our discussion especially on the problem of tempo-invariant recognition of time sequences, that is, a problem somewhat related to recognition of words in a continuous stream of speech.

1. Introduction

The neocognitron was previously proposed by the author as a neural network model for visual pattern recognition [1]-[3]. The structure of the network was suggested by the visual system of the brain. It can recognize stimulus patterns correctly, even if the patterns are shifted or distorted. It can acquire this ability by unsupervised learning. The repeated presentation of a set of training patterns is sufficient for the self-organization of the network, and it is not necessary to give any information about the categories in which the patterns should be classified. Since the neocognitron has the function of generalization, it is not necessary to teach all the deformed versions of the training patterns. It even comes to recognize correctly a pattern which has not been presented before, provided it resembles one of the training patterns.

The original neocognitron can recognize spatial patterns only, and do not have the ability to process time-varying patterns. We are trying to extend the neocognitron so as to be able to recognize not only spatial patterns but also spatio-temporal patterns, such as speech signals.

The vibration of the sound wave caught by the eardrum generates a traveling wave through the basilar membrane in the cochlea in the inner ear. Different locations along the basilar membrane have different resonance frequencies. Thus the frequency of an acoustic signal is transformed into a particular place of vibration along the basilar membrane. The vibration pattern of the basilar membrane is sensed by hair cells that line along the membrane, and is converted to neural signals. Hence, the input pattern to the auditory nervous system is a one-dimensional spatial pattern that varies with time. We can treat speech recognition as a problem of spatio-temporal pattern recognition.

In the first part of this paper, we discuss merits and problems which will arise when trying to extend the neocognitron to recognize spatio-temporal patterns. After that, we concentrate our discussion mainly on the problem of tempo-invariant recognition of time sequences, that is, a problem somewhat related to recognition of words in a continuous stream of speech. We have proposed several models for spatio-temporal pattern recognition [4]. Among these models, we pick up a simplified model with which we are mainly testing the function of tempo-invariant recognition of time sequences.

2. Spatio-temporal Patterns vs. Spatial Patterns

2.1 Why Neocognitron for Speech Recognition?

We will first consider the similarity between visual pattern recognition and auditory pattern recognition. In order to make the following discussion simple, we will tentatively consider an easiest way of using the neocognitron for spatio-temporal pattern recognition, although this is not the model proposed in this paper.

The process which is tentatively considered is as follows: Using an array of delay lines, the spatio-temporal pattern is transformed into a two-dimensional spatial pattern in which one axis correspond to the time axis in the original spatio-temporal pattern. The neocognitron can be used to recognize this converted spatial pattern.

We, human beings, can recognize speech correctly, even if the frequency of the speech sound is somewhat shifted. In other words, we can recognize speech even if an audio tape is played at a speed different from the recording speed. The frequency shift of the sound corresponds to the shift in position in the spatio-temporal pattern, and consequently to the shift in position in the two-dimensional spatial pattern. Because of the shift-invariant nature of the neocognitron, the effect of frequency shift would be absorbed.

We can also recognize speech even if it is spoken slowly or quickly. The change in speaking speed, that is, the elastic expansion or contraction of time axis, is converted to the change in scale in the two-dimensional spatial pattern. The size-invariant recognition by the neocognitron would also solve this problem.

The ability of deformation-invariant recognition of the neocognitron would be useful to make speaker-independent recognition of speech sounds.

The process of converting spatio-temporal patterns into two-dimensional spatial patterns by delay lines, however, seems to be unnatural as a model of biological systems. Hence, we try to extend the neocognitron to be able to process spatio-temporal patterns directly without transforming them into two-dimensional spatial patterns.

2.2 Peculiarity of Straight Lines

When recognizing a spatial pattern consisting of line drawings by the neocognitron, the cells in a lower stage extract local features such as line segment in a particular orientation, intersection of lines, corner of a bent line, curvature of a line, end of a line, and so on. Among these features, line segments show a peculiar characteristics different from others.

Let a letter "L", which consists of a bent line, be presented to the input layer (or the retina) of a neocognitron. Many cells will be activated in the cell-plane* which extracts horizontal line segments, and also in the cell-plane extracting vertical line segments. However, only one cell (or at most a few cells around it) will be activated in a cell plane which extract corners. The same is true for each end of the bent line. If the output signal from these cell-planes are treated under the same criterion, the influence of the line segments becomes extremely strong because of the large number of activated cells. To make matters worse, the number of activated cells varies considerably with the change in size of the input pattern, depending on the length of the straight lines in the pattern. This causes deformation-invariant recognition difficult.

---

* A cell-plane of the neocognitron is defined as a subgroup of cells which extract the same feature but from different locations in the retina. Cells in a cell-plane are arranged topographically in order in a two-dimensional array.

In the conventional neocognitron which are designed to recognize handwritten numerals [3], training patterns which would generate cell-planes extracting line segments are not given during the supervised learning of the intermediate layer U_{S2}. Layer U_{S2} are trained to extract such features as ends of lines, corners, curvatures, intersections, and so on.

Disuse of line-extracting cells in the intermediate layer plays an important role in endowing the network with the ability of size- and deformation-invariant recognition of line drawings. In order to recognize a straight line, for example, it is enough to detect the existence of both ends of the line, and also examine the absence of all other features between them. If the input pattern is not a straight line, some other features must be detected between the both ends of the line. Thus we can recognize input patterns without detecting line segments explicitly in the middle part of lines. Since cells detecting line segments does not exist in the layer, the number of activated cells does not vary so much even if the size of the input pattern changes.

This does not mean, however, that line extracting cells are completely eliminated from all stages of the network. On the contrary, the first layer U_{S1} consists of cell-planes extracting line segments only, and no other features are extracted in this stage. All the other features are extracted indirectly in higher stages using the output of layer U_{S1}.

In the case of spatio-temporal pattern recognition, a continuous signal which does not change with time corresponds to a straight line in the spatial pattern recognition. By eliminating cells responding to stationary continuous signals, we expect to help endowing our model with the ability to tempo-invariant recognition of temporal patterns.

Hence, in our simplified model for spatio-temporal pattern recognition, transient components (or temporal variation) are first detected from a time sequence of spatial patterns, and the sequence of such transients are processed by layers of cells connected in a hierarchical manner. Transient detecting cells correspond to on-cells and off-cells found in the auditory nervous system. An on-cell is activated by the start of a continuous tone, and an off-cell responds to the end of the tone.

3. A Simplified Model for Sequence Recognition

3.1 Sequence Recognition

The basic idea for sequence recognition in our model resembles somewhat to that proposed by Tank and Hopfield [5] in the sense that sets of delay filters are used. Filters of different delay times detect signals from different parts of the input time sequence. In their model, one set of delay filters is used to detect the entire sequence at a time, while in our model, many sets of delay filters are used to detect fragments of the input sequence. In the next stage of the hierarchical network, the information about fragments of the input sequence is integrated by other sets of delay filters which has longer delay than the first. This kind of processings are repeated in a hierarchical manner in a multilayered network.

In these models, a filter which has a longer delay has a broader tuning curve on time scale. This is effective to make tempo-invariant recognition of sequences. However, the detection of the order of stimuli in a sequence (say, the order of phonemes in a word) becomes ambiguous, especially at the beginning part of the sequence which is detected by a filter of broader tuning curve.

In order to reduce the adverse effect of broad tuning curve of the filters, and still to maintain the ability of tempo-invariant recognition, our

model is designed in such a way that the difference in the tempo of the input sequences is absorbed a little at a time at each stage of the hierarchical network.

3.2 Structure of the Network

Supposing that the deformation-invariant recognition of spatial patterns could be performed by the same mechanism as the neocognitron if necessary, we omit these function from our simplified model.

The network has a hierarchical structure like the neocognitron. Input signal to the model is a time sequence of spatial patterns, and is given to input layer U_0 consisting of receptor cells. Behind U_0, layers of S-cells and C-cells are connected alternately in a hierarchical manner, and they are denoted by Us and Uc, respectively: $U_0 \rightarrow Us_1 \rightarrow Uc_1 \rightarrow Us_2 \rightarrow Uc_2 \rightarrow Us_3 \rightarrow Uc_3 \rightarrow Us_4$.

Layer Us_1 in the first stage consists of on-cells and off-cells. An on-cell responds to the beginning of the response of the corresponding cell in the input layer U_0, and an off-cell responds to the ending of the response.

An S-cell sends signals to a subgroup of C-cells of the next Uc layer. The connection from the S-cell to a C-cell has a characteristic like a delay line of broad tuning curve. C-cells in each subgroup have connections of different delay time ranging from zero to a certain maximum delay. The breadth of the tuning curve increase in proportion to the delay time of the connection.

In an intermediate stage, even the longest delay of the connections to C-cells is shorter compared to the duration time of the entire input sequence. In other words, the Uc-layer of the intermediate stage processes only a part of the input sequence at a time. The higher the stage is, the longer is the time span of the sequence which are processed by the Uc-layer in the stage. In the highest stage, it is as long as to cover the entire input sequence.

S-cells (except those of layer Us_1) receive input connections from all the C-cells in the preceding stage, and extract spatial features at every step of time. The input connection of S-cells do not have any time-delay. These connections have plasticity and reinforced by unsupervised learning using a competitive learning procedure similar to the one used in the neocognitron.

The S-cells in the highest stage works as recognition cells. Each of them observes the entire input information indirectly from the output of the C-cells of the preceding layer, and responds only to one particular input sequence.

Acknowledgement. This work was supported in part by a Bioscience Grant for International Joint Research Project from NEDO, Japan.

References

[1] K. Fukushima: "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position", Biol. Cybernetics, **36**[4], 193-202 (1980).

[2] K. Fukushima, S. Miyake: "Neocognitron: A new algorithm for pattern recognition tolerant of deformation and shifts in position", Pattern Recognition, **15**[6], 455-469 (1982).

[3] K. Fukushima: "Neocognitron: A hierarchical neural network capable of visual pattern recognition", Neural Networks, **1**[2], 119-130 (1988).

[4] T. Ito, K. Fukushima: "Recognition of spatio-temporal patterns with a hierarchical neural network", to be presented in this conference: IJCNN-90-WASH DC (Jan. 1990).

[5] D. W. Tank, J. J. Hopfield: "Neural computation by concentrating information in time", Proc. Natl. Acad. Sci. USA, **84**, 1896-1900 (1987).

TEXTURED IMAGE SEGMENTATION USING LOCALIZED RECEPTIVE FIELDS [1]

Joydeep Ghosh, Nanda Gopal, and Alan C. Bovik
Department of Electrical and Computer Engineering,
University of Texas, Austin, TX 78712-1084

Abstract: We present an approach to texture analysis that uses spatially localized filters and cooperative-competitive mechanisms for determining emergent boundaries. Gabor filters that closely resemble cortical receptive fields are used to activate columns of cells that selectively respond to localized frequency and orientation attributes of an image. For each cell column, a winner-take-all network that is moderated by the activities of neighboring columns is used to gradually segment the image. All the mechanisms used are biologically plausible and typically yield results that are superior to those reported previously for real images.

1. Texture-based Segmentation.

Both biological systems and computational vision models use texture for perceptual tasks such as segmentation of surfaces, classification of surface materials and computation of shape. An image texture can be interpreted as a pattern of image intensities projected from a surface of uniform surface radiance attributes. Several models of perception use spatial filtering or frequency analysis to perform segmentation based on characteristics of the image spectrum. Since frequency estimates are inherently global, identifying significant spectral components in an image does not remove the burden of determining their spatial supports. Indeed, it is possible to generate textures having identical power spectra but with different local structures that are preattentively discriminable [9].

An alternative approach developed recently regards an image as a "carrier" of region/surface information which can be encoded into several narrowband channels[1,2]. A localized multi-frequency multi-orientation channel decomposition of an image yields local orientation and spatial frequency estimates that are used as key parameters for classifying texture. Bovik *et al.* [1] have proposed texture segmentation methods based on the recovery and analysis of the amplitude envelopes of Gabor-filtered images with two predominant textures. Daugman [6] has shown that the 2-D Gabor functions uniquely achieve the lower bound of the conjoint uncertainty relationship - $\Delta u \times \Delta x \geq 1/4\pi$ and $\Delta v \times \Delta y \geq 1/4\pi$, where (x, y) and (u, v) represent 2D spatial and spectral coordinates respectively. Thus these functions optimally achieve simultaneous localization in space and frequency. Furthermore, these functions closely resemble the receptive field profiles of simple cells in the visual cortex of cats, and their coefficients can be determined by a neural network [5,6]. A slight modification of the Gabor transform has been used by Perry and Lowe [13] for segmenting textured images through an iterative growth of seed regions. In this paper we extend this methodology to cater to images with arbitrary types of textures, and present a novel neural network for the adaptive segmentation of images.

2. Determination of Localized Spatial Frequencies.

A continuous, monochromatic image containing only a narrow range of frequencies concentrated near (U_k, V_k) can be described by:

$$t_k(x, y) = a(x, y)cos[2\pi(U_k x + V_k y) + p(x, y)], \tag{1}$$

where $a(x, y)$ and $p(x, y)$ are slowly varying amplitude and phase terms. A natural texture $t(x, y)$, can be modeled as composites of sub-images of the form (1):

$$t(x, y) = \sum_{i=1}^{N} a_i(x, y)cos[2\pi(U_i x + V_i y) + p_i(x, y)] + t'(x, y),$$

where t' is a residual containing shared or insignificant frequencies and any dc term.

The image $t(x, y)$ is the input to our *texture discrimination network*. The assembly of 'neural' cells that forms this network contains N layers of cells. The cell at position (x, y) in the ith layer has a localized receptive field centered at coordinates (x, y) of the image. The synaptic strengths in this field are selected

[1] This research is supported by the Texas Advanced Research Program under Grant No. 3456 and by URI Project R-145.

so as to mimic a 2D Gabor filter with center frequency (U_i, V_i). Such a filter is described by: $h(x,y) = g(x',y')e^{2\pi j[U_i x + V_i y]}$, where $(x',y') = (xcos\phi + ysin\phi, -xsin\phi + ycos\phi)$, and $g(x,y) = \frac{1}{2\pi\lambda\sigma^2} exp\{-[(x/\lambda)^2 + y^2]/2\sigma^2\}$. Here, h is a complex sine grating modulated by a 2D Gaussian with aspect ratio λ, scale parameter σ, and major axis oriented at an angle ϕ from the x-axis.

Let $m_i(x,y), 1 \leq i \leq N$, be the activation of the cell in the ith layer with retinotopic coordinates (x,y). When an image is presented to the discrimination network, the N cell activations at each point (x,y) are proportional to the amplitude responses of N Gabor filters. Within a component texture, the activation vector formed by such a column of cells is expected to vary smoothly. At texture boundaries, on the other hand, the activation vectors will differ significantly. It can be shown that, if the textures are highly narrowband, then segmentation may be obtained using the region assignment:

$$(x,y) \in \Re_n \text{ if } n = \arg\{\max [m_i(x,y)]\}, 1 \leq i \leq N. \tag{2}$$

Usually, the envelopes m_i are not smooth enough to allow consistent segmentation using Eq.(2). If some texture does not have narrow bandwidths, leakage will occur. This can be reduced by post-filtering the amplitude envelopes with a Gaussian filter having the same shape as the channel filters but a greater spatial extent. The modified region assessment

$$(x,y) \in \Re_n \text{ if } n = \arg\{\max [g_i(x/\gamma, y/\gamma) * m_i(x,y)]\}, 1 \leq i \leq N; \;\; 0 < \gamma < 1, \tag{3}$$

will then yield a smoother segmentation. In our experience, we have found $\gamma = 2/5$ to be effective in most cases.

Even the modified region assessment gives inadequate performance for textures that vary smoothly in frequency and/or orientation. By using a *max* function in Eq.(2) and (3), information about the differentials in the cell responses is not fully utilized. Region assignment at each point (x,y) should be dependent not only on the strength of each hypothesis indicated by the column of cells at that point, but also on the activation in neighboring columns. The next section describes how these column vectors are used for adaptively detecting emergent texture boundaries.

3. Detecting Boundaries through Cooperative-Competitive Mechanisms.

On the presentation of an image, the feedforward network using local receptive fields enables each cell plane to reach an activation level corresponding to the amplitude envelope of the Gabor filter that it represents. A cooperative-competitive feedback network is then used as a Smoothing, Adaptive Winner-Take-All Network (SAWTA) so that only one cell gradually becomes predominant in each column.

To implement this mechanism, each cell receives constant inhibition from all other cells in the same column, along with excitatory inputs from neighboring cells in the same row or plane. The synaptic strengths of the excitatory connections exhibit a 2D Gaussian profile centered at (x,y). The network is mathematically characterized by shunting cooperative-competitive dynamics [8] that model on-center off-surround interactions among cells which obey membrane equations [10]. Thus, at each point (x,y), the evolution of the cell in the ith layer is governed by:

$$\tau \frac{d}{dt} m_i = -m_i + (A - m_i)J^+ - (B + Cm_i)J^-, \tag{4}$$

where J^+, J^- are the net excitatory and inhibitory inputs respectively, and are given by

$$J^+ = \alpha \sum_{(x_n,y_n)\in R} m_i(x_n,y_n) e^{\frac{-[(x-x_n)^2+(y-y_n)^2]}{2\sigma^2}} \;\; ; \;\; J^- = \sum_{j\neq i} f(m_j(x,y)).$$

Here, R is the neighboring region of support and f is a sigmoidal transfer function. The stability of such systems has been shown by Cohen and Grossberg. The network is allowed to run for t_n iterations before region assignment is performed using Eq.(2).

Experimental Results: A variety of natural and synthetic images have been segmented using the SWATA network, and compared to previous approaches that use Eq.(2) or (3). The 256x256 gray level images are prefiltered using a Laplacian-of-Gaussian to remove high dc components, low-pass phase functions, and

to suppress aliasing. Sixteen circularly-symmetric Gabor filters are used. Sets of three filters with center frequencies increasing in geometric progression (ratio = 2.1) are arranged in a daisy-petal configuration along 5 orientations, while the sixteenth filter is centered at the origin. Figure 1 shows the segmentation achieved by the three approaches for a synthetic texture. Figure 2 uses an aerial view of tilled land that can be easily segmented by the human eye, and compares the same three approaches. For both Fig. 1.d and 2.d, the constants A, B and C in Eq.(4) were taken to be 1, 0 and 10 respectively. The activation function used is $f(x) = tanh(2x)$. Figure 3 shows the effect of varying the number of iteration steps t_n and the inhibition factor C, on the segmentation obtained. We observe that the SAWTA network achieves a more smooth segmentation in regions where the texture shows small localized variations, while preserving the boundaries between drastically different textures. Usually, 10 iterations suffice to demarcate the segment boundaries, and any changes after that are confined to arbitration among neighboring filters.

4. Concluding Remarks.

Our method does not require a feature extraction stage as in [7] or computationally expensive masking fields [4]. The incremental and adaptive nature of the SAWTA network enables it to avoid making early decisions about texture boundaries. The dynamics of each cell is affected by the image characteristics in its neighborhood as well by the formation of more global hypotheses. Typically, four spatial frequencies are dominant in human visual systems. This suggests the use of a mechanism for post-inhibitory response [12] that suppress cells with activation below a threshold and speeds up the convergence of a SAWTA network. Also, the adaptive learning network of Kohonen [11] can be used to change both excitatory and inhibitory synaptic strengths (J^+, J^-) in response to a teaching input. This approach is being currently studied. The use of cell responses for instantaneous frequency estimates leading to shape-from-patterns analysis also merits active investigation[3].

References

[1] A. C. Bovik, M. Clark, and W. S. Geisler. Computational texture analysis using localized spatial filtering. In *Proc. IEEE Comput. Soc. Workshop on Computer Vision*, pages 201–206, Miami Beach, Fl., 1989.

[2] A. C. Bovik, M. Clark, and W. S. Geisler. Multichannel texture analysis using localized spatial filters I: Segmentation by channel demodulation. *IEEE Trans. PAMI*, to appear, 1989.

[3] A. C. Bovik, T. Emmoth, and N. Gopal. *Numerical Analysis of Visual Patterns.* Technical Report TR-88-6-51, Computer and Vision Research Center, Univ. of Texas, Austin, December, 1988.

[4] G. A. Carpenter and S. Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*, 37:54–115, Jan. 1987.

[5] M. Clark and A. C. Bovik. Texture discrimination using a model of visual cortex. In *Proc. IEEE Conf. on Systems, Man and Cybernetics*, Atlanta, GA, 1986.

[6] J. G. Daugman. Complete discrete 2-D Gabor transforms by neural networks for image analysis and compression. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 36:1169–1179, July 1988.

[7] N. R. Dupaguntla and V. Vemuri. A neural network architecture for texture segmentation and labeling. In *International Joint Conference on Neural Networks*, pages 127–144 (I), June 1989.

[8] S. Grossberg and E. Mingolla. Neural dynamics of perceptual grouping: textures, boundaries and emergent segmentations. *Perception and Psychophysics*, 38:141–171, 1985.

[9] B. Julesz. Spatial nonlinearities in the instantaneous perception of textures with identical power spectra. *Phil. Trans. R. Soc. London, B*, 290:83–94, 1980.

[10] E. R. Kandel and J. H. Schwartz. *Principles of Neuroscience.* Elsevier/North-Holland, Amsterdam, 1981.

[11] T. Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, Berlin, 1984.

[12] J. Malik and P. Perona. A computational model of texture segmentation. In *Proc. Intl. Conf. on Computer Vision and Pat. Recognition*, pages 326–332, 1989.

[13] A. Perry and D. G. Lowe. Segmentation of textured images. In *Proc. Intl. Conf. on Computer Vision and Pat. Recognition*, pages 319–325, 1989.

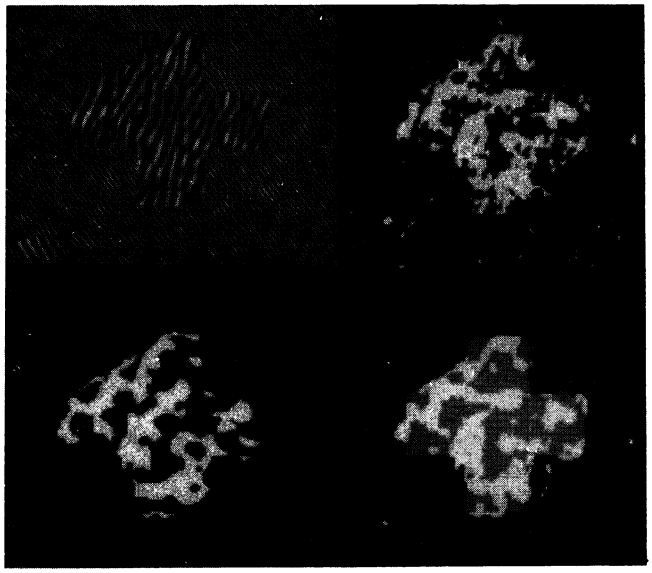


Figure 1

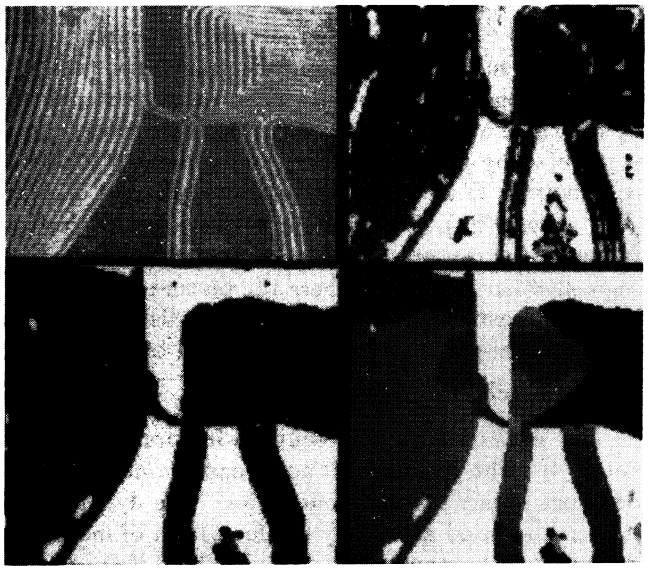


Figure 2

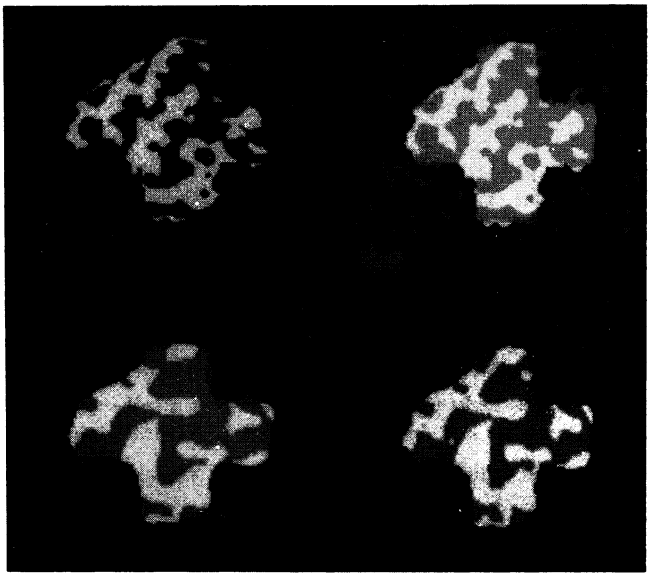


Figure 3

Fig. 1: Segmentation of a synthetic texture. (clockwise from top left): (a) original (b) segmentation using (2) (c) segmentation using (3) (d) segmentation after 10 iterations of the SAWTA network.

Fig. 2: Segmentation of an aerial image. (a) original (b) segmentation using (2) (c) segmentation using (3) (d) segmentation after 10 iterations of the SAWTA network.

Fig. 3: Effect of iteration steps and inhibition factor on segmentation. (a) same as Fig. 1(d) (b) segmentation using C = 3 (c) after 100 iterations (d) after 50 iterations.

Computational Framework and Neural Networks for Low and Intermediate 3D Computer Vision

Ziqing Li
Department of Artificial Intelligence, University of Edinburgh
5 Forrest Hill, Edinburgh EH1 2QL, U.K.
zql%uk.ac.edinburgh.edai@ucl.cs.nss

1. Introduction

Computer vision via range image analysis has potential use in automated vision tasks because of the richness of the explicit surface information contained in range data. Surface representation is one of the important ways used to characterize objects. Surface curvature features, such as mean curvature and Gaussian curvature, are invariant to rotations, translations and changes in surface parameterization and are therefore desirable properties for 3-D object recognition and location[1]. The surface curvature based representations facilitate many 3-D vision problems.

According to *Differential Geometry* [9], mean curvature and Gaussian curvature at a point on a graph surface can be computed from the five partial derivatives of the graph surface function which is the range data in the problems considered in this paper. However, numerical differentiation is mathematically an ill-posed problem [10],[12]. Regularization techniques have been used to make an ill-posed problem well-posed [3],[6]. Some researchers have introduced discontinuities in finding regularization solutions for early vision processing such as surface reconstruction, edge detection [2],[7],[11].

In this paper, we present from regularization theory a computational framework for low level and intermediate level processing of 3D computer vision based on invariant surface curvature features, covering *from a range image to invariant surface descriptions* [8]. We then map the corresponding computational algorithms onto neural network paradigms. The final interests of this work is the understanding of range images.

2. Low Level: From a Range Image to Surface Curvature Images

The problem we are facing in this level is to estimate surface curvature from noisy range data. It is done through minimization of some energies involving discontinuities, with the result of regularized data. Partial derivatives are estimated from the regularized data.

2.1 Computational Framework

We chooses for our problem energy functions of the form as:

$$E(f \mid d,\pi,\Omega_n) = E^p(f \mid d) + \lambda E^s(f \mid \pi,\Omega_n) \tag{1}$$

In this function, $d=\{d(i,j) \mid (i,j)\in \mathbf{Z}_M\}$ is the given data, where $\mathbf{Z}_M=\{(i,j) \mid 1\le i,j\le M\}$ is an integer lattice, $f=\{f(i,j) \mid (i,j)\in \mathbf{Z}_M\}$ a function to be found, $\lambda>0$ a weight, $\Omega_n=\{\omega_n(i,j) \mid (i,j)\in \mathbf{Z}_M\}$ is the homogeneous neighborhood system in which $\omega_n(i,j)=\{(u,v) \mid 0<(u-i)^2+(v-j)^2\le n\}$ is the set of neighbors of (i,j) where n is an integer (note $(i,j)\notin \omega_n(i,j)$). $\pi=\{\pi(i,j) \mid (i,j)\in \mathbf{Z}_M\}$ is a characteristic function indicating (the absence of) depth discontinuities at (i,j) with respect to $\omega_n(i,j)$. At this level, The neighborhood parameter $n=1$ and the cardinal number of the neighborhood set $\#\omega_1(i,j)=4$. $E^p(f \mid d)$ is a function that measures the discrepancy between f and d:

$$E^p(f \mid d) = \sum_{(i,j)} [f(i,j)-d(i,j)]^2$$

$E^s(f \mid \pi,\Omega_n)$ is a membrane energy that measures the (lack of) smoothness of the function f with respect to Ω_n controlled by π:

$$E^s(f \mid \pi,\Omega_n) = \sum_{(i,j)} \sum_{(u,v)\in\omega_n'(i,j)} \pi(i,j)[f(u,v)-f(i,j)]^2$$

where $\omega_n'(i,j)=\{(u,v) \mid (u,v)\in\omega_n(i,j);\ u\ge i, v\ge j;\ (i,j)\in \mathbf{Z}_M\}$ is a subset of $\omega_n(i,j)$.

Solution f^* minimizes E:

$$E(f^* \mid d,\pi,\Omega_n) = \min_f\{E(f \mid d,\pi,\Omega_n)\} \tag{2}$$

E is quadratic with π fixed and therefore there is a unique solution for the minimization problem (2).

2.2 Neural Networks

By substituting V for f, I for d, we can write (1) as:

$$E(V) = \sum_{(i,j)}[V(i,j)-I(i,j)]^2 + \sum_{(i,j)}\sum_{(u,v)\in\omega_n'} T(i,j;u,v)[V(u,v)-V(i,j)]^2$$

where $T(i,j;u,v)=T(i,j)=\lambda\pi(i,j)$. A simple dynamic system with characteristic $\frac{dE}{dt}\le 0$ is determined by a system of differential equations:

$$\frac{dV(i,j)}{dt} = -\mu\frac{\partial E}{\partial V(i,j)} \qquad \textit{for all } i,j$$

where $\mu>0$ is a constant. From this, we can write the dynamic equations for the network:

$$C\frac{dV(i,j)}{dt} = I(i,j) - \frac{V(i,j)}{R} + \sum_{(u,v)\in\omega_n(i,j)} T(i,j;u,v)\,[V(u,v)-V(i,j)] \qquad \textit{for all } i,j \tag{3}$$

where $C = -\mu > 0$ and $R = 1$. Since $\frac{dE}{dt} = \sum_{(i,j)}\frac{\partial E}{\partial V(i,j)}\frac{dV(i,j)}{dt} = -C\sum_{(i,j)}[\frac{dV(i,j)}{dt}]^2 \le 0$, with $\frac{dE}{dt}=0$ only when $\frac{dV(i,j)}{dt}\equiv 0$ *for all* (i,j), and E is lower-bounded (from 0), the system will converge to the minimum energy state and E is therefore a Liapunov function. Once initialized, *i.e.* once I, C, R and T are set, the network will evolve as to settle into a stable state [5], that of the least power dissipation.

According to Kirchhoff's current law, (3) is the dynamic equation of of a massive network at node (i,j). Fig.1 illustrates the circuit for neuron (i,j) of this network and its connection.

The stable state of $\{V(i,j)\}$ is the minimum solution f^* of (2). Partial derivative estimates f_x, f_y can be computed from the minimization solution f^* using the finite difference formulation. Fig.2 illustrates the derivative estimation process.

3. Intermediate Level: From the Curvature Images to Invariant Surface Descriptions

The problem we are facing in this level is to segment an estimated curvature image g into a curvature sign (+,0,-) map which we expect to possesses some surface-patch coherence. If g is a noise-free image, this problem can be well solved by a thresholding operation at thresholds $\pm\tau(\tau>0)$, resulting in an initial 3-level segmentation. However, in the noisy case, the problem is not so simple. In what follows, an appropriate energy function will be constructed as the cost of segmentation to achieve coherent and consistent segmentation from noisy images, and a piecewise-constant configuration is found as the solution of the segmentation problem by minimizing the energy function.

3.1 Computational Framework

The energy functions is defined by:

$$E(h \mid g,\pi,\tau,\Omega_n) = E^p(h \mid g,\tau)+\lambda E^s(h \mid \pi,\Omega_n) \tag{4}$$

In this function, $h=\{h(i,j) \mid (i,j)\in \mathbf{Z}_M\}$ is the map of the curvature sign; $h(i,j)\in\Phi$, where $\Phi=\{+,0,-\}$ is a set whose elements correspond to the positive, zero and negative sign of curvature. $g=\{g(i,j) \mid (i,j)\in \mathbf{Z}_M\}$ the input curvature data. λ, π and Ω_n are as illustrated in section 2.1. $\tau>0$ is a pre-determined threshold [8]. $E^p(h \mid g,\tau)$ is a function that measures the plausibility of a mapping from the curvature function g into the curvature sign function h subject to the threshold τ:

$$E^p(h\,|\,g,\tau) = \sum_{(i,j)} e^p_{i,j}(h(i,j)\,|\,g(i,j),\tau)$$

where $h(i,j)\in\Phi$, $g(i,j)\in\mathbf{R}$ and $e^p_{i,j}(\phi\,|\,\eta,\tau)$ is a function that measures the plausibility of a mapping from $g(i,j)$ to $h(i,j)$. $e^p_{i,j}(\phi\,|\,\eta,\tau)$ is a function defined by:

$$e^p_{i,j}(+\,|\,\eta,\tau) = \begin{cases} 1 & \eta<0 \\ 1-\eta/2\tau & 0\le\eta<2\tau \\ 0 & \eta\ge 2\tau \end{cases} \qquad e^p_{i,j}(0\,|\,\eta,\tau) = \begin{cases} |\eta/2\tau| & |\eta|<2\tau \\ 1 & \textit{otherwise} \end{cases} \qquad e^p_{i,j}(-\,|\,\eta,\tau) = e^p_{i,j}(+\,|\,-\eta,\tau)$$

$E^s(h\,|\,\pi,\Omega_n)$ is a function that measures the smoothness of the function h with respect to the neighborhood system Ω_n

$$E^s(h\,|\,\pi,\Omega_n) = \sum_{(i,j)} e^s_{i,j}(h(i,j)\,|\,\pi(i,j),\omega_n(i,j)) = \sum_{(i,j)} \pi(i,j)\ \#\{(u,v)\in\omega_n(i,j)\ |\ h(u,v)\neq\phi\}/\#\,\omega_n$$

The neighborhood parameter n=5 at this intermediate level processing stage. The solution h^* of the problem minimizes function E:

$$E(h^*\,|\,g,\pi,\tau,\Omega_n) = \min_h\{E(h\,|\,g,\pi,\tau,\Omega_n)\} \tag{5}$$

3.2 Neural Networks

Replace h by a 3-layer net $V = \{\,[V_+(i,j), V_0(i,j), V_-(i,j)]^T\ |\ (i,j)\in\mathbf{Z}_M\}$. The state of a neuron takes boolean value $V_\phi(i,j)\in\{0,1\}$. Among layers, states of neurons have to satisfy a consistency constraint: $\sum_\phi V_\phi(i,j)=1$. This multi-layer network is an extension of [3]. Let $I_\phi(i,j)=e^p_{i,j}(\phi\,|\,g(i,j),\tau)$ and $T=\lambda\pi$. Equation (4) can be written in terms of the new notations as:

$$\begin{aligned} E(V) &= \sum_\phi\sum_{(i,j)} I_\phi(i,j)V_\phi(i,j) + \frac{\lambda}{N}\sum_\phi\sum_{(i,j)} \pi(i,j)V_\phi(i,j)\,[N - \sum_{(u,v)\in\omega_n(i,j)} V_\phi(u,v)] \\ &= \sum_\phi\sum_{(i,j)} I_\phi(i,j)V_\phi(i,j) - \sum_\phi\sum_{(i,j)}\sum_{(u,v)} T(i,j;u,v)V_\phi(i,j)V_\phi(u,v) + \textit{constant} \end{aligned}$$

where $N=\#\,\omega_n$ and $T(i,j;u,v)=T(i,j)=(\lambda/N)\pi(i,j)$.

The energy change $\Delta E^{(t)}(i,j)$ due to the state change $\Delta V^{(t)}(i,j) = [\Delta V^{(t)}_+(i,j), \Delta V^{(t)}_0(i,j), \Delta V^{(t)}_-(i,j)]^T$ of a neuron at (i,j) is:

$$\Delta E^{(t)}(i,j) = \sum_\phi H^{(t)}_\phi(i,j)\,\Delta V^{(t)}_\phi(i,j)$$

where

$$H^{(t)}_\phi(i,j) = I_\phi(i,j) - 2\sum_{(u,v)} T(i,j;u,v)V^{(t)}_\phi(u,v)$$

The updating rule for the gradient descent is that *for all* (i,j):

$$V^{(t+1)}_p(i,j)\leftarrow 0$$
$$V^{(t+1)}_q(i,j)\leftarrow 1$$
$$\text{if}$$
$$H^{(t)}_q(i,j) < H^{(t)}_p(i,j) \qquad \textit{for all } p\neq q$$

According to the updating rule, we have $\Delta V^{(t)}_p=-1$ and $\Delta V^{(t)}_q=1$ and therefore the local energy change:

$$\Delta E^{(t)}(i,j) = H^{(t)}_q - H^{(t)}_p$$

is always negative if any change occurs. $V^{(t)}$ will finally converge. $V^{(\infty)}$ is a map representing the solution h^* of (5) from either a mean curvature image or a Gaussian curvature image.

In this network, the neurons within a layer cooperate whereas those among layers compete. The circuit for neuron (i,j) of this network and its connection is illustrated in fig.3.

A mean curvature sign map and a Gaussian curvature sign map are derived from the present intermediate process. The combination of the two maps yields a new map that tells up to eight different surface types. This description is invariant to rotations, translations and scale changes. Fig.4 shows results from a Renault part image. Upon this, an adjacency graph can be derived with surface patch attributes attached to each node of the graph, with which a third network for object recognition and location can start working.

References

[1] P.J. Besl and R.C. Jain, "Invariant surface characteristics for three dimensional object recognition in range images," *Computer Vision Graphics and Image Processing*, vol.33, no.1, pp.33-80, Jan, 1986.

[2] A. Blake and A. Zisserman, *Visual Reconstruction*, Cambridge, MA: MIT press, 1987.

[3] W.E.L. Grimson, *From Images to Surfaces: A Computational Study of the Human Early Visual System*, MA: MIT Press 1984.

[4] J.J Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Natl. Acad. Sci. USA*, 79:2554-2558, 1982.

[5] J.J Hopfield, "Neurons with graded response have collective computational properties like those of two state neurons," *Proc. Natl. Acad. Sci. USA*, 81:3088-3092, 1984.

[6] B.K.P Horn, *Robot Vision*, Cambridge, MA:MIT press, 1986.

[7] C. Koch, J. Marroquin and A. Yuille, "Analog 'neuronal' networks in early vision," *A.I. Lab. Memo*, No.751, MIT Cambridge, Mass., 1985.

[8] Z. Li, "Invariant surface segmentation through energy minimization with discontinuities," *submitted to Intl. Journal of Computer Vision.*, April 1989.

[9] M. Lipschutz, *Differential Geometry*, McGraw-Hill, 1969.

[10] T. Poggio, V. Torre and C. Koch, "Computational vision and regularization theory," *Nature*, 317, pp.314-319, 1985.

[11] D.T. Terzopoulos, "Regularization of inverse problems involving discontinuities," *IEEE Trans. Pattern Analysis Machine Intell.*, vol. PAMI-8, pp.129-139, 1986.

[12] A.N. Tikhnov and V.A. Arsenin, *Solutions of Ill-posed Problems*, Winston & Sons, Washington, D.C., 1977.

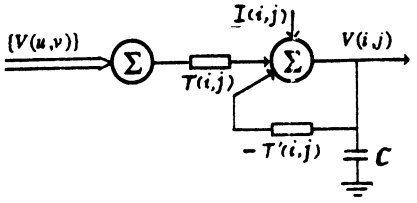


Fig.1 A circuit for neuron (i,j) of the network for low-level processing and its connection, where $\{V(u,v)\}=\{V(u,v) \mid (u,v)\in\omega_n(i,j)\}$ and $T'(i,j)=1+4T(i,j)$.

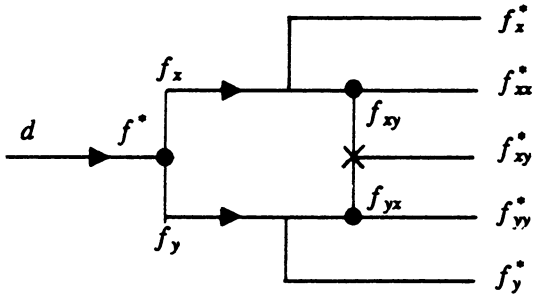


Fig.2 An illustration of the partial derivative estimation process. A triangle represents a regularization operation, a filled circle a finite difference operation and the cross the mean-averaging operation.

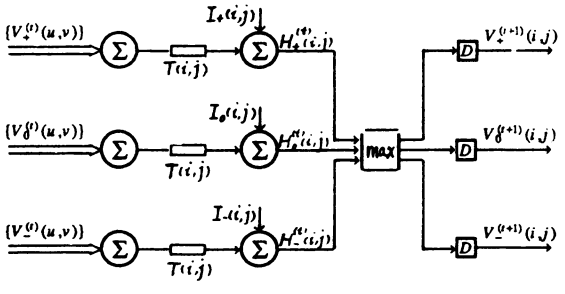


Fig.3 A circuit for 3-layered neuron (i,j) of the network for intermediate-level processing and its connection where $\{V_\phi^{(t)}(u,v)\}=\{V_\phi^{(t)}(u,v)\mid(u,v)\in\omega_n(i,j)\}$ and "max" operation selects the maximum component from the input and outputs in 0/1.

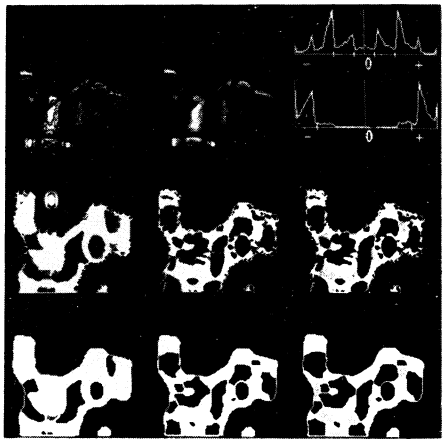


Fig.4 Experimental results from a Renault part image

DESIGNING A SENSORY PROCESSING SYSTEM: WHAT CAN BE LEARNED FROM PRINCIPAL COMPONENTS ANALYSIS?[1]

Ralph Linsker
IBM Research, T. J. Watson Research Center, Yorktown Heights, NY 10598
(Electronic mail: linsker@ibm.com)

INTRODUCTION

How can we understand the rich and complex organization of processing functions found in many biological perceptual systems? The visual system of macaque monkey, for example, is composed of dozens of anatomical regions which we may think of as processing stages, interconnected in a manner that suggests extensive serial and parallel processing of visual input, with both feedforward and feedback interactions between regions (Van Essen 1985). Are there rules that might account for the specific processing functions found in each such region, and for the ways in which cell properties develop and become organized to perform these functions?

In earlier work (Linsker 1986) I showed that in a layered feedforward network of linear-response cells whose connections develop according to a Hebb-type rule, cells can develop feature-analyzing properties (including center-surround and orientation-selective receptive fields) that are qualitatively similar to properties found in the early stages of visual processing in cat and monkey. Analysis of the Hebb-type rule used in that study showed that the rule causes each cell's set of connection strengths $\{c_i\}$ to change with time so as to maximize the quadratic form $V \equiv \Sigma_{i,j} c_i Q_{ij} c_j$ subject to the constraints Σc_i = constant and $c_- \le c_i \le c_+$ for all i. (The constraint on Σc_i is not explicitly imposed, but emerges from the particular form of Hebb-type rule used.) Here Q_{ij} is the covariance of the signal activities at a pair of connections i and j that provide input to the developing cell. The quantity being maximized, V, is the statistical variance of the output activity of the developing cell. To visualize the optimization problem geometrically, note that: (1) The connection strength vector $\mathbf{c}$ lies within a hypercube bounded by c_- and c_+. (2) The covariance matrix Q has no nonnegative eigenvalues. Therefore, apart from the possibility of zero eigenvalues, $V(\mathbf{c})$ has a unique local minimum at the origin. (3) V is to be maximized on the hyperplane defined by Σc_i = constant. This maximum is guaranteed to occur at a point on the boundary of the hypercube at which all, or all but one, of the cs are extremal (Linsker 1986). V is not in general guaranteed to have a unique local maximum (owing to the constraints).

If we instead use a rule that maximizes V subject to the different constraint Σc_i^2 = constant (with no additional constraint on each c_i), we are maximizing V on a hypersphere. The well-known solution is then that $\mathbf{c}$ should lie in the direction of the leading eigenvector of Q (that having maximal eigenvalue), or in the subspace spanned by the eigenvectors having identical maximal eigenvalue, if degeneracy exists. Such a cell performs principal components analysis (PCA) on its inputs, in the sense that the cell's output value is the leading principal component of the input vector.

I emphasize that the solutions obtained in (Linsker 1986) are not the PCA solutions, owing to the difference in constraints. However, the particular constraints can be changed by using a different Hebb-type rule (e.g., Oja 1982), and there is no compelling reason to choose one such rule rather than another. It is therefore worthwhile to explore what feature-analyzing properties emerge when a cell develops to perform PCA on its inputs.

Why is PCA of interest in this context? (1) PCA, or the Karhunen-Loève expansion, is a classic and important tool of statistical analysis (Watanabe 1985). (2) If one uses the cell's output value to optimally reconstruct the input values, the reconstruction has smaller mean squared error if the output is the leading principal component than if the output is any other linear function of the input values. (3) Under special

[1] This paper covers in greater detail a portion of the talk entitled "Using Information Theory to Guide the Design of a Perceptual System," presented at the IJCNN Winter 1990 Meeting (Washington, DC, Jan. 1990).

conditions (e.g., a linear response cell, a Gaussian distribution of input vectors, and Gaussian noise) a cell whose output is the leading principal component of the input vector conveys maximum Shannon information about the input vector (Linsker 1988).

Why, on the other hand, might we expect that PCA does not suffice as a principle for determining what response property a cell develops to analyze an ensemble of input "scenes"? (4) Cell response is in general nonlinear. (5) Biological constraints are in general complex, and the particular PCA constraint Σc_i^2 = constant is not a necessary (or even a particularly plausible) one. (6) A processing layer contains many mutually interacting cells, and there is no reason to require each cell to develop according to an optimization principle, independently of its neighbors. (7) There is no apparent reason to "reconstruct" the input values from a cell's output value in a biological system, so that the optimal reconstruction property of PCA (which is important in a data compression context) is of unclear value here. Also, if the biological circuitry does not in fact perform such a reconstruction, the appropriate choice of constraints to impose on such a hypothetical reconstruction (in order that one may compute the optimal connection strengths for the encoding stage) is obscure. (8) Feedback from "later" processing stages may be important in determining the cells' response properties.

An information-theoretic principle, "maximum information preservation," has been proposed (Linsker 1988) as an optimization principle for a processing stage of a perceptual system. [For a review of earlier ideas relating information theory and perception, see (Linsker 1990).] This principle reduces to PCA in restricted cases [see item (3) above], yet is applicable to the more general situations described in items (4)-(6) of the preceding paragraph. A new criterion -- namely, that the output values from a layer should optimally discriminate among the input vectors -- replaces the minimum reconstruction error criterion of item (7). (Further extension of the principle is needed to accommodate feedback from later stages.) Some consequences of this information-theoretic principle for biological systems and model networks are described in (Linsker 1988, 1989a,b). In particular, (Linsker 1989a) shows how the principle selects an information-theoretically optimal solution from the low-dimensional subspace spanned by the first several principal components, in certain types of linear systems.

With this background, let us explore some general properties of PCA solutions, and the character of these solutions for input ensembles of biological interest. The goal is to see what we can learn about biological response properties from studying PCA solutions, and what properties remain unexplained in the context of PCA.

The main analytic, and some of the numerical, results presented here appeared earlier as part of a technical report (Linsker 1987). Relationships between PCA and the emergence of feature-analyzing properties in biological systems have also been studied for the case of orientation selectivity by Sanger (1989) and Yuille et al (1989), and (for ocular dominance) by Miller et al (1989).

SPECIFICATION OF THE EIGENFUNCTION PROBLEM

Suppose we are given an ensemble of input "presentations" $\{L^\pi(\mathbf{x})\}$, where $L^\pi(\mathbf{x})$ is the signal activity of presentation π at position $\mathbf{x}$ in the two-dimensional layer L. Without loss of generality we may assume that $<L^\pi(\mathbf{x})> = 0$ for all $\mathbf{x}$, where the angle brackets denote the ensemble average (over π). (Replace each $L^\pi(\mathbf{x})$ by $L^\pi(\mathbf{x}) - <L^\pi(\mathbf{x})>$.) Consider an arbitrary cell of the following layer M, whose output is a linear function of the activities in layer L: $M^\pi = \int L^\pi(\mathbf{x}) a(\mathbf{x}) d^2\mathbf{x}$. The covariance function of $L^\pi(\mathbf{x})$ is $Q(\mathbf{x}, \mathbf{x}') \equiv <L^\pi(\mathbf{x}) L^\pi(\mathbf{x}')>$. If Q is a function only of $(\mathbf{x} - \mathbf{x}')$, then the eigenfunctions of Q are the Fourier components of $Q(\mathbf{x} - \mathbf{x}')$, which extend over all of layer L (Linsker 1989a, Yuille et al 1989). To obtain localized solutions in such cases, we will introduce a weighting function $g(\mathbf{x})$ that falls off with increasing $|\mathbf{x}|$ and in effect limits the spatial region of L to which an M cell will respond. [Yuille et al (1989) study the unlocalized case (of translation-invariant Q) analytically, and the localized case by numerical simulation. We will show that the eigenfunction problem in the localized case is separable. This leads to useful analytic results concerning the geometric form of the eigenfunctions, simplifies the numerical computation, and helps to avoid computational problems arising from near-degeneracy of eigenvalues.]

There are several different ways to introduce such a localizing function. We state three ways, motivated by different considerations, all of which lead to the same eigenfunction problem and subsequent analysis.

(1) Discrete connections with density function: Consider an M cell (centered at $\mathbf{x} = 0$) that receives discrete connections from layer L, with a density of connections given by $g(\mathbf{x})^2$. The strength of each

connection i located at $\mathbf{x}_i$ is $c(\mathbf{x}_i) \equiv c_i \equiv a(\mathbf{x}_i)/g(\mathbf{x}_i)^2$. The output value is $M^\pi = \int L^\pi(\mathbf{x})g(\mathbf{x})^2 c(\mathbf{x}) d^2\mathbf{x} = \Sigma_i L^\pi(\mathbf{x}_i)c_i$. In terms of the discrete connections, the PCA criterion is that we maximize the output variance $V = \langle (M^\pi)^2 \rangle = \Sigma_{i,j} c_i Q_{ij} c_j$ subject to $\Sigma c_i^2 = 1$, where the discrete covariance matrix $Q_{ij} \equiv \langle L(\mathbf{x}_i) L^\pi(\mathbf{x}_j) \rangle$. In the continuum representation, the output variance is $V = \int c(\mathbf{x})g(\mathbf{x})^2 Q(\mathbf{x}, \mathbf{x}')g(\mathbf{x}')^2 c(\mathbf{x}') d^2\mathbf{x} d^2\mathbf{x}'$, and the constraint is $\int g(\mathbf{x})^2 c(\mathbf{x})^2 d^2\mathbf{x} = 1$. Define $z(\mathbf{x}) \equiv a(\mathbf{x})/g(\mathbf{x}) = c(\mathbf{x})g(\mathbf{x})$. Then $V = \int z(\mathbf{x}) S(\mathbf{x}, \mathbf{x}') z(\mathbf{x}') d^2\mathbf{x} d^2\mathbf{x}'$ and the constraint is $\int z(\mathbf{x})^2 d^2\mathbf{x} = 1$, where $S(\mathbf{x}, \mathbf{x}') \equiv g(\mathbf{x})Q(\mathbf{x}, \mathbf{x}')g(\mathbf{x}')$. In the continuum representation, PCA therefore consists of determining the eigenfunctions $z(\mathbf{x})$ of $S(\mathbf{x}, \mathbf{x}')$.

(2) Masking of input by a localizing function: A simple way to introduce the localizing function g is to premultiply $L^\pi(\mathbf{x})$ by $g(\mathbf{x})$ before providing it as input to the M cell. (The origin $\mathbf{x} = 0$ is defined appropriately for each M cell in turn.) This approach could be appropriate if the goal is to use PCA to design an array of filters in a synthetic network, for example. Then $M^\pi = \int L^\pi(\mathbf{x})g(\mathbf{x})a(\mathbf{x}) d^2\mathbf{x}$ and $V = \int a(\mathbf{x})g(\mathbf{x})Q(\mathbf{x}, \mathbf{x}')g(\mathbf{x}')a(\mathbf{x}') d^2\mathbf{x} d^2\mathbf{x}'$. If we impose the constraint $\int a(\mathbf{x})^2 d^2\mathbf{x} = 1$ then we should define $z(\mathbf{x}) \equiv a(\mathbf{x})$ and (as before) $S(\mathbf{x}, \mathbf{x}') \equiv g(\mathbf{x})Q(\mathbf{x}, \mathbf{x}')g(\mathbf{x}')$. In this case PCA again consists of determining the eigenfunctions of $S(\mathbf{x}, \mathbf{x}')$.

(3) The criterion of minimum weighted mean squared reconstruction error: In this example we do not fix either the density of connections or any constraint on their strengths. Instead, we ask: For a linear response function $M^\pi = \int L^\pi(\mathbf{x})a(\mathbf{x}) d^2\mathbf{x}$, how should the $\{a(\mathbf{x})\}$ be chosen so as to minimize the <u>weighted</u> mean square reconstruction error

$$\text{WMSE} \equiv \int g(\mathbf{x})^2 \langle [L^\pi(\mathbf{x}) - L^\pi_{\text{est}}(\mathbf{x})]^2 \rangle \, d^2\mathbf{x}, \tag{1}$$

where the weighting factor $g(\mathbf{x})^2$ is applied to the MSE at $\mathbf{x}$? Here $L^\pi_{\text{est}}(\mathbf{x})$ is the optimal estimate of $L^\pi(\mathbf{x})$ for <u>given</u> $\{a(\mathbf{x})\}$; it is given (Liebelt 1967) by

$$L^\pi_{\text{est}}(\mathbf{x}) = M^\pi \times \left[\int Q(\mathbf{x}, \mathbf{x}')a(\mathbf{x}') \; d^2\mathbf{x}' \right] / \left[\int a(\mathbf{x}')Q(\mathbf{x}', \mathbf{x}'')a(\mathbf{x}'') \, d^2\mathbf{x}' \, d^2\mathbf{x}'' \right]. \tag{2}$$

Therefore WMSE = $\{\int g(\mathbf{x})^2 \langle [L^\pi(\mathbf{x})]^2 \rangle d^2\mathbf{x}\} - H$, where the first term on the right-hand side is independent of $\{a(\mathbf{x})\}$, and

$$H \equiv \left\{ \int g(\mathbf{x})^2 \, [Q(\mathbf{x}, \mathbf{x}')a(\mathbf{x}') \, d^2\mathbf{x}']^2 \, d^2\mathbf{x} \right\} / \left[\int a(\mathbf{x})Q(\mathbf{x}, \mathbf{x}')a(\mathbf{x}') \, d^2\mathbf{x} \, d^2\mathbf{x}' \right]. \tag{3}$$

The minimum-WMSE criterion therefore requires that we choose $\{a(\mathbf{x})\}$ so as to maximize H. [Cf. (Linsker 1988) for the unweighted case.] Now define $z(\mathbf{x}) \equiv a(\mathbf{x})/g(\mathbf{x})$ and $S(\mathbf{x}, \mathbf{x}') \equiv g(\mathbf{x})Q(\mathbf{x}, \mathbf{x}')g(\mathbf{x}')$. Algebraic manipulation gives

$$H = \left[\int z(\mathbf{x}')S(\mathbf{x}', \mathbf{x})S(\mathbf{x}, \mathbf{x}'')z(\mathbf{x}'') \, d^2\mathbf{x} \, d^2\mathbf{x}' \, d^2\mathbf{x}'' \right] / \left[\int z(\mathbf{x})S(\mathbf{x}, \mathbf{x}')z(\mathbf{x}') \, d^2\mathbf{x} \, d^2\mathbf{x}' \right]. \tag{4}$$

Let $z_k(\mathbf{x})$ and λ_k be the kth eigenfunction and eigenvalue in a complete set of orthonormal eigenfunctions of the kernel S; that is,

$$\int S(\mathbf{x}, \mathbf{x}') \, z_k(\mathbf{x}') \, d^2\mathbf{x}' = \lambda_k \, z_k(\mathbf{x}) \tag{5}$$

for all $\mathbf{x}$ and each k. For any eigenfunction $z_k(\mathbf{x})$, the H value of the corresponding M cell is λ_k. For any linear combination of eigenfunctions $z(\mathbf{x}) = \Sigma_k \, b_k \, z_k(\mathbf{x})$ [hence for arbitrary $z(\mathbf{x})$], $H = (\Sigma b_k \lambda_k^2) \; / \; (\Sigma b_k \lambda_k)$. H is maximal for the eigenfunction of S having largest eigenvalue. The linear combinations of the first few eigenfunctions (in order of decreasing eigenvalue) give the M cell types having large H value.

All of the approaches described lead to the same problem: that of determining the eigenfunctions of $S(\mathbf{x}, \mathbf{x}')$. We proceed to derive some analytic results of general interest, and numerical results for particular types of input ensembles that arise in models of early visual processing.

ANALYTIC RESULTS: GEOMETRIC FORM OF EIGENFUNCTIONS

The situation of principal interest to us is that in which (1) $Q(\mathbf{x}, \mathbf{x}')$ is a function $Q(s)$ only of the separation $s \equiv |\mathbf{x} - \mathbf{x}'|$, and (2) the localizing function $g(\mathbf{x})$ is a function $g(r)$ only of the distance r of $\mathbf{x}$ from the origin (the position of the origin is specific to each output cell). Under these conditions $S(\mathbf{x}, \mathbf{x}')$ depends only on $(u, u', |\theta' - \theta|)$, where $u \equiv r^2$, $u' \equiv r'^2$, and θ, θ' are the angles that $\mathbf{x}$, $\mathbf{x}'$ make with a reference axis. [To see this, note that $S(\mathbf{x}, \mathbf{x}') = g(r)Q(|\mathbf{x} - \mathbf{x}'|)g(r')$ and that the argument of Q is a function only of $(u, u', |\theta' - \theta|)$.]

For a covariance function S of the form $S(u, u', |\theta' - \theta|)$, the eigenfunction problem of Eqn. 5 is separable in polar coordinates. To see this, assume a product form $z(u, \theta) = f(u) \sin(n\theta + \delta)$. Then Eqn. 5 becomes (with $\phi \equiv \theta' - \theta$)

$$(1/2)\int S(u,u', |\phi|) f(u') \sin [n(\phi + \theta) + \delta] du'd\theta' = \lambda f(u) \sin(n\theta + \delta). \tag{6}$$

The left-hand side equals $(1/2)\sin(n\theta + \delta)\int du'f(u')\int d\phi \cos(n\phi)S(u,u', |\phi|)$. Therefore this $z(u, \theta)$ is an eigenfunction of S if and only if $f(u)$ satisfies $\int du'T_n(u, u')f(u') = \lambda f(u)$ where

$$T_n(u, u') \equiv (1/2)\int d\phi \cos(n\phi)S(u, u', |\phi|). \tag{7}$$

For each $n \geq 0$, we denote the (complete) set of eigenfunctions of T_n by $\{f_{ni}(u)\}$ $(i \geq 0)$ and the corresponding eigenvalues by $\{\lambda_{ni}\}$. A complete set of eigenfunctions of S is then given by $\{f_{0i}(r), f_{ni}(r) \cos n\theta, f_{ni}(r) \sin n\theta\}$ for all $n > 0$, $i \geq 0$. For all $n > 0$ and all i, the eigenfunctions occur as degenerate orthogonal pairs (with angular dependence $\sin n\theta$ and $\cos n\theta$). In particular cases, special symmetries of S may give rise to additional degeneracies, or "accidental" additional degeneracies may occur.

In the absence of such additional degeneracies, we see that each eigenfunction of S must have a "sectored" geometric form. That is, each eigenfunction is either circularly symmetric ($n = 0$), or has two semicircular regions of opposite sign ($n = 1$), or has $2n$ sectors of alternating sign, with angular dependence of the form $\sin(n\theta + \delta)$ for arbitrary phase δ. Note that the only eigenfunctions that can be described as having a form similar to that of a Hubel-Wiesel orientation-selective "simple" cell are the $n = 1$ solutions. We shall return to the comparison with "simple" cells later.

NUMERICAL RESULTS: RELATION TO FEATURE-ANALYZING CELLS

To proceed further we must assume specific forms for S. We shall treat two different types of input ensemble $\{L^\tau(\mathbf{x})\}$.

(1) Random field convolved with Gaussian filter: Here each presentation $L^\tau(\mathbf{x})$ is obtained by convolving a random field of identically and independently distributed (i.i.d.) values with a two-dimensional Gaussian filter $\exp(-s^2)$. The resulting covariance function is (apart from an overall constant factor) $Q(s) = \exp(-s^2/2)$. We obtain for the kernel T_n of the radial eigenfunction problem:

$$\begin{aligned} T_n(u,u') &= (1/2)g(u^{1/2})g(u'^{1/2}) \exp[-(u + u')/2]\int_0^{2\pi} d\phi \cos(n\phi) \exp[(uu')^{1/2} \cos\phi] \\ &= \pi g(u^{1/2})g(u'^{1/2}) \exp[-(u + u')/2] I_n[(uu')^{1/2}] \end{aligned} \tag{8}$$

where I_n is a modified Bessel function.

Figure 1a shows the two largest eigenvalues of T_n (for each of $n = 0, 1, 2$) as functions of the parameter r_0, for a Gaussian localizing function $g(r) = \exp(-r^2/2r_0^2)$. The corresponding six radial eigenfunctions are shown in Fig. 1b for the case $r_0 = 1.5$. Note that the leading eigenfunction is the circularly symmetric ($n = 0, i = 0$) solution. The eigenvalues of the broken-symmetry eigenfunctions ($n \geq 1$) are considerably smaller in value (for this choice of S and g).

(2) Random field convolved with "Mexican-hat" filter: Here each input presentation is obtained by convolving an i.i.d. random field with a filter that is proportional to the Laplacian of a two-dimensional Gaussian (a type of "Mexican-hat" function). This filter function is $(1 - s^2)\exp(-s^2)$, and the resulting covariance function is $Q(s) = (1 - s^2 + s^4/8)\exp(-s^2/2)$ (obtained by calculating the overlap integral of

two identical filter functions whose centers are separated by distance s). The kernel T_n has a form similar to Eqn. 8, but with I_n replaced by a sum of terms involving $I_{n\pm0,1,2}$.

Figures 1c,d show the eigenvalues and eigenfunctions of T_n for $n = 0, 1, 2$ and $i = 0, 1$. Recall that each eigenfunction of S is the product of a radial eigenfunction of T_n and an angular factor $\sin(n\theta + \delta)$ for arbitrary phase δ. Thus the leading eigenfunction of S has $(n,i) = (0,0)$ and is of circularly symmetric center-surround form. The next two orthogonal eigenfunctions of S [with $(n,i) = (1,0)$] have angular dependence $\sin(\theta + \delta)$ and $\cos(\theta + \delta)$ for arbitrary δ, and are therefore "orientation-selective" in the sense of Hubel-Wiesel "simple" cells of odd symmetry. It is important to note that the eigenfunctions of S for $n \geq 2$ are necessarily of "sectored" form (having $2n$ "pie slice" regions of alternating sign), and are <u>not</u> orientation-selective in the sense of "simple" cells (although, clearly, symmetry breaking occurs for all $n \geq 1$). As stated above, the only exception to this statement arises if the eigenfunction problem happens to have additional symmetries or "accidental" degeneracies, in which case one may be able to construct an eigenfunction of simple-cell form as a linear combination of degenerate "sector-shaped" eigenfunctions of S.

Some notes and caveats are in order before proceeding further. (1) In certain cases (particularly in the large-r_0 regime of Fig. 1c, where several angular eigenmodes have similar but not identical eigenvalues), one must be particularly careful when solving the two-dimensional eigenfunction problem numerically, to avoid computing spurious "eigenfunctions" that can have a banded appearance reminiscent of orientation-selective cells, but are in fact not eigenfunctions of S (R. Linsker, unpublished observations, 1987). The recognition that the eigenfunctions of S must have a sinusoidal dependence on polar angle -- unless true degeneracy exists between different modes n -- helps in avoiding this difficulty. [Sanger's (1989) method for computing transfer functions gives banded, non-"sectored" solutions; the reason for this is not described. Yuille et al (1989) provide insight into the reason for the occurrence of near-degeneracy in a related problem.] (2) Two of the

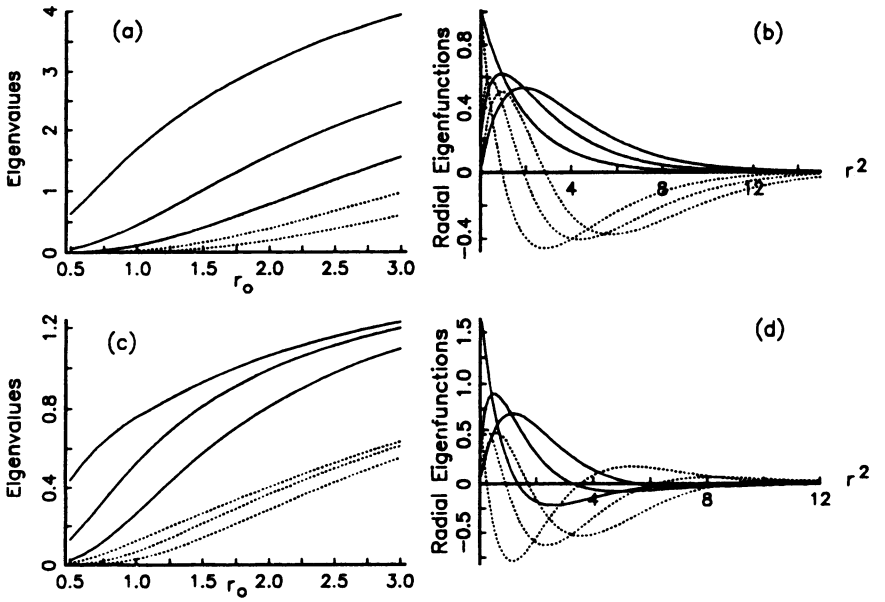


Figure 1. PCA solutions: (a) Eigenvalues for angular eigenmode indices $n = 0, 1, 2$ and radial eigenmode indices $i = 0, 1$, where each input presentation is a random field convolved with a Gaussian filter, and the localizing function $g(\mathbf{x}) = \exp(-r^2/2r_0^2)$. Curves starting at top: $(n,i) = (0,0)$; $(1,0)$; $(2,0)$ and $(0,1)$ (degenerate); $(1,1)$; $(2,1)$. (b) Radial eigenfunctions for the same six modes, for $r_0 = 1.5$ (solid, $i = 0$; dotted, $i = 1$; within each triplet of curves, the value of r^2 at the peak or trough of the curve shifts to the right as n increases). (c) Same as (a) but where each input presentation is a random field convolved with a "Mexican-hat" filter (see text). Curves starting at top: $n = 0,1,2$ for $i = 0$, then $n = 0,1,2$ for $i = 1$. (d) Same as (b) but for the input presentations described in (c). Note that the leading solution ($n = 0$, $i = 0$) has "center-surround" geometry.

lower-lying eigenmodes in Fig. 1a, $(n,i) = (2,0)$ and $(0,1)$, happen to be degenerate. This degeneracy depends upon the particular forms of S and g; it vanishes if, for example, the form of $g(r)$ is changed from Gaussian to $g(r) \propto \exp(-r^{2.4}/2r_0^2)$. (3) The $(n,i) = (1,0)$ eigenmode is not necessarily lower-lying than the (0,0) mode. For example, choosing $g(r) = \{1 + \exp[(r - r_0)/\Delta r]\}^{-1/2}$ (a "flattop" function in two dimensions with a smoothed sigmoid-like falloff), with $r_0 = 1.6$ and $\Delta r = 0.1$, gives eigenvalues $\lambda_{10} = 1.17 > \lambda_{00} = 1.05 > \lambda_{20} = 0.92$, for the S function used in Figs. 1c,d. That is, an orientation-selective solution is favored over the circularly symmetric one for that $g(r)$.

We return to the solutions discussed earlier and illustrated in Figs. 1c,d. The geometric form of the $(n,i) = (1,0)$ and $(2,0)$ eigenfunctions of S is shown explicitly in Fig. 2a,b. The function plotted is the eigenfunction of S (denoted z_{ni}) multiplied by $g(r)$; this equals the transfer function from layer L to the M cell for the three problems introduced in the "Specification ..." section above. [The transfer function is $a(\mathbf{x})$ for problems (1) and (3), and $g(\mathbf{x})a(\mathbf{x})$ for problem (2).] An orientation-selective cell of even "-+-" symmetry (Fig. 2c) is <u>not</u> itself an eigenfunction of S, but can be constructed as the sum of two eigenfunctions (which are not degenerate), $z_{00} + z_{20}$. Other linear combinations of the leading eigenfunctions have a variety of shapes (Figs. 2d-g) which are not in general "orientation-selective" in the sense of Hubel and Wiesel's (1962) classic receptive field (RF) plots. [To obtain RF plots for the "cells" discussed here, one must convolve the transfer function shown with the receptive field of the cells in the input layer L. However, for the cases shown, the transfer function gives a good qualitative indication of the RF shape. Cf. (Linsker 1986) and (1988) for an illustration of this distinction for an orientation-selective cell.]

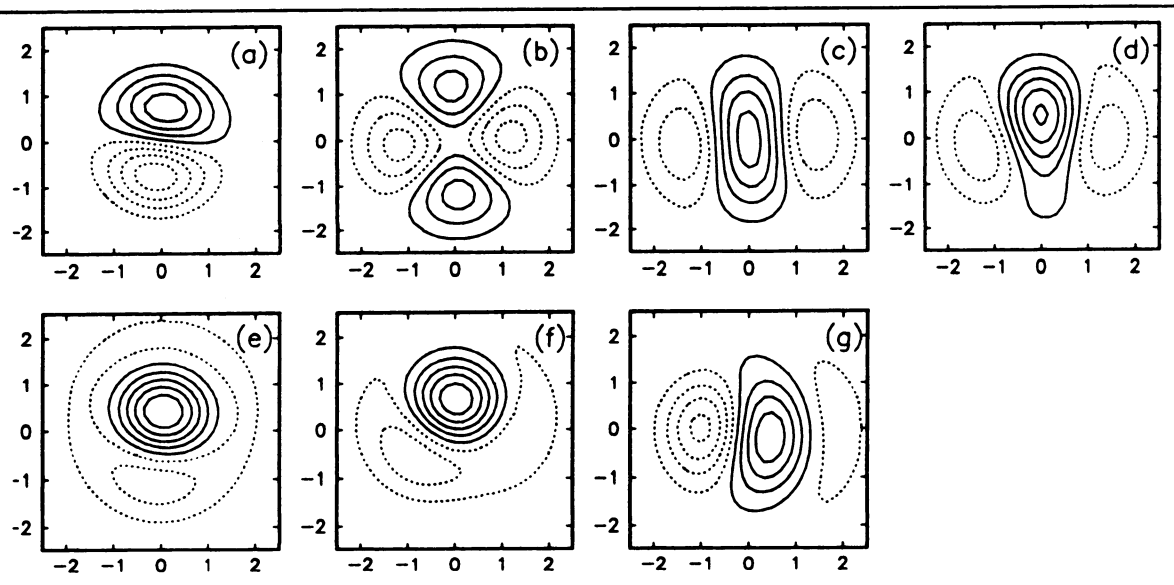


Figure 2. Transfer functions $g(\mathbf{x})z(\mathbf{x})$ where $z(\mathbf{x})$ are linear combinations of the first several eigenfunctions $z_{ni}(\mathbf{x})$ of S, for input consisting of random field convolved with a "Mexican-hat" filter, and Gaussian $g(r)$ with $r_0 = 2.5$: (a) $z = z_{10}$ (eigenvalue $\lambda_{10} = 1.10$). (b) $z = z_{20}$ ($\lambda_{20} = 0.96$). (c) $z \propto z_{00} + z_{20}$ (z_{00} is center-surround eigenfunction, not shown, having $\lambda_{00} = 1.15$). (d) $z \propto z_{00} + (z_{10}/2) + z_{20}$. (e) $z \propto z_{00} + z_{10}$. (f) $z \propto z_{00} + 2z_{10} + z_{20}$. (g) $z \propto z_{00} + 2z'_{10} + z_{20}$, where z'_{10} is the $(n,i) = (1,0)$ eigenfunction orthogonal to z_{10}. Solid (resp. dotted) contour curves denote positive (resp. negative) values; contours are spaced every 0.15 units (starting with $\pm$ 0.075). All z functions normalized to $\int z(\mathbf{x})^2 d^2\mathbf{x} = 1$.

CONCLUSIONS

Principal components analysis is a useful tool for understanding some aspects of the feature-analyzing properties of cells found in at least the first few stages of a sensory processing pathway. This is illustrated for the case of center-surround and orientation-selective cells. In particular, we (i) start with a simplified model in which the input activities correspond to a random field convolved with either a Gaussian or a "Mexican-hat" filter, (ii) compute the transfer functions for model "cells" whose output activity has high (but not necessarily maximal) statistical variance, and (iii) find that some of those "cell" types are qualitatively similar to those found biologically. These "cell" types lie in a subspace spanned by the first several eigenfunctions in a PCA or Karhunen-Loève expansion. The relevant eigenfunction problem is shown to be separable in polar coordinates; this fact leads to useful constraints on the geometric form of the eigenfunctions. A connection between PCA and a criterion for the minimization of a geometrically-weighted mean squared reconstruction error is derived.

Although the properties of a low-dimensional PCA subspace may provide some insight into observed cell response properties, PCA (at least as used here) is insufficient to determine which of the transfer functions lying within such a subspace will be found in a biological system. To make more detailed predictions of cell responses, and of the organization of these response properties within a processing stage, a more powerful set of principles or constraints appears to be needed. An information-theoretic optimization criterion that is related to PCA in certain cases, but has greater generality, is cited and briefly discussed as a candidate for such a principle.

References

Hubel, D. H., Wiesel, T. N. 1962. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. J. Physiol. 160:106-54

Liebelt, P. B. 1967. An Introduction to Optimal Estimation. Reading, MA: Addison-Wesley

Linsker, R. 1986. From basic network principles to neural architecture (series). Proc. Natl. Acad. Sci. USA 83:7508-12, 8390-94, 8779-83

Linsker, R. 1987. Towards an organizing principle for perception: Hebbian synapses and the principle of optimal neural encoding. IBM Res. Rep. RC12830, IBM Research, Yorktown Heights, NY. [Note the following corrections: At p.12, change "$S(\mathbf{x} - \mathbf{x}')$" to "$S(\mathbf{x}, \mathbf{x}')$." Eqn. 7 (p.12) should read: $S(\mathbf{x}, \mathbf{x}') \equiv w(\mathbf{x})^{1/2} Q(\mathbf{x} - \mathbf{x}') w(\mathbf{x}')^{1/2}$. These changes have no effect on the rest of the analysis.]

Linsker, R. 1988. Self-organization in a perceptual network. Computer 21(3):105-17

Linsker, R. 1989a. An application of the principle of maximum information preservation to linear systems. In Advances in Neural Information Processing Systems (Denver, CO, 1988), ed. D. S. Touretzky, 1:186-94. San Mateo, CA: Morgan Kaufmann

Linsker, R. 1989b. How to generate ordered maps by maximizing the mutual information between input and output signals. Neural Computation 1:396-405

Linsker, R. 1990. Perceptual neural organization: Some approaches based on network models and information theory. Annu. Rev. Neurosci. 13, in press

Miller, K. D., Keller, J. B., Stryker, M. P. 1989. Ocular dominance column development: Analysis and simulation. Science 245:605-15

Oja, E. 1982. A simplified neuron model as a principal component analyzer. J. Math. Biol. 15:267-73

Sanger, T. D. 1989. An optimality principle for unsupervised learning. In Advances in Neural Information Processing Systems (Denver, CO, 1988), ed. D. S. Touretzky, 1:11-19. San Mateo, CA: Morgan Kaufmann

Van Essen, D. C. 1985. Functional organization of primate visual cortex. In Cerebral Cortex, ed. A. Peters, E. G. Jones, vol. 3. New York: Plenum

Watanabe, S. 1985. Pattern Recognition: Human and Mechanical. New York: John Wiley & Sons

Yuille, A. L., Kammen, D. M., Cohen, D. S. 1989. Quadrature and the development of orientation selective cortical cells by Hebb rules. Biol. Cybern. 61:183-94

A Real Time ART-1 Based Vision System for Distortion Invariant Recognition and Autoassociation

J. C. Rajapakse, O. G. Jakubowicz and R. S. Acharya
Department of Electrical & Computer Engineering
SUNY at Buffalo
Buffalo, NY 14260.

Introduction

A neural architecture is presented which can recognize and reconstruct the traces of previously learned input patterns. The recognition and reconstruction properties of the network are invariant under input patterns that are translated, distorted, noisy, incomplete, scaled and slightly rotated with respect to the trained patterns. If novel training patterns are to be incorporated into the network , it is not necessary to relearn the previously trained patterns.

This network utilizes feature combination and coagulation of information in a manner related to Fukushima's neocognitron[1] and similar to multilayer and multifeature map architecture described by Jakubowicz in [2]. Feature categories at each stage are learned using the Adaptive Resonance Theory introduced by Grossberg in [3].

Adaptive resonance architectures self-organize stable recognition codes in real time in response to arbitrary sequence of input patterns. ART-1 generates the codes of categorization for binary input patterns as connection weights between the neurons in the input plane and the output plane. The vigilance parameter determines the coarseness of categorization. The network achieves its stability using the feedback connections and with a test and reset mechanism. Although the ART-1 is completely characterized as a system of ordinary differential equations, fast learning equations are sufficient for real time applications. All ART networks used in pattern recognition require either a preprocessor or the input pattern to be identical in shape, size and location and to be noise free.

Neocognitron is a hierarchical model of a competitive learning paradigm capable of selectively recognizing position shifted or shape distorted patterns. The variable connections between the stages grow according to a simplified Hebbian learning rule for the input stimuli appearing in the lower stages. The different features recognized by different stages are determined during the self organization or training phase. Positional error is tolerated bit by bit at each stage by reduction of the size of planes. The forward connections of the network manage the function of pattern recognition and the backward connections manage the function of selective attention, segmentation and associative recall. The forward connections are identical to the backward connections. The efferent signal flow is guided by the afferent signal flow. The gain control signals make even the vague traces of the input to be recognized by lowering the thresholds for the afferent signal flow.

ART learning improves the stability and reduces the complexity of a neocognitron like architecture. The architecture can learn and incorporate new training exemplars in real time. The network is capable of extracting even the vague traces of input features during afferent signal flow. The new architecture is not unstable unlike other hierarchical architectures based on Kohenen's feature map learning algorithm and does not need a lot of iteration for training. Extension to recognize analog patterns and to reduce the size of large weight matrices with ART-2 learning principles is possible.

Description of the Architecture

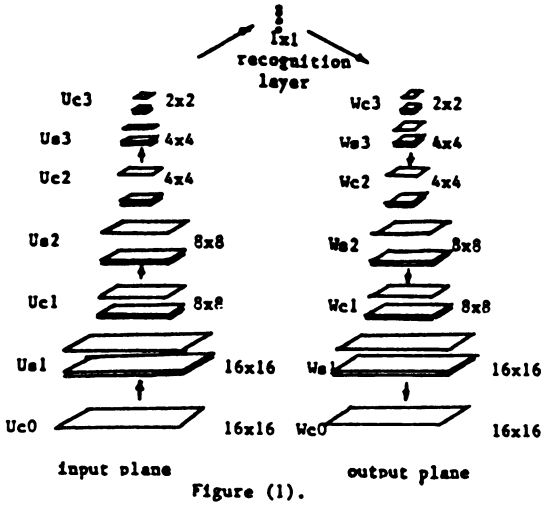


Figure (1).

Figure (1) shows the basic architecture for the system. The architecture consists of five stages. The input patterns are to be sensed by the Uc_0 layer consisting of 16x16 neurons. Each path in each stage has two layers s and c namely and each layer consists of one or more planes of artificial neurons. The layers in the afferent path are represented by U and in the efferent path by W. Each plane in every Us layer represents a feature that appeared in the previous Uc stage during the training phase. The input to a Us layer at the spatial location $\mathbf{n} = (n_x, n_y)$, is restricted to windows of size $D = 5\text{x}5$ in all planes of the previous Uc layer. $Uc_l(k, \mathbf{n})$ represents the l^{th} stage activity of neurons in the k^{th} Uc plane at the location $\mathbf{n}$. The pattern at $\mathbf{n}$ is said to be centered and selected for training if

$$1.5 \leq \frac{\sum_{v \epsilon D} \sum_{k=0}^{K} n_x . Uc_l(k, \mathbf{n}+v)}{\sum_{v \epsilon D} \sum_{k=0}^{K} Uc_l(k, \mathbf{n}+v)} \leq 2.5 \text{ and } 1.5 \leq \frac{\sum_{v \epsilon D} \sum_{k=0}^{K} n_y . Uc_l(k, \mathbf{n}+v)}{\sum_{v \epsilon D} \sum_{k=0}^{K} Uc_l(k, \mathbf{n}+v)} \leq 2.5$$

Only centered input patterns are selected for training. This avoids training redundant patterns and increases the number of features that a particular stage can learn (see [2]).

Feature categorization in each stage is done according to the ART-1 learning algorithm. If the input within a window at the spatial location $\mathbf{n}$ is selected for learning, and if the K^{th} neuron in the $l+1^{th}$ layer wins then the connection weights are modified as shown below. The weights connected between $l+1^{th}$ stage neurons in K^{th} plane and l^{th} stage neurons within a window, $v \epsilon D$, of k^{th} plane, is represented by $a_l(K, k, v)$. If the K^{th} node has already learned a feature and the input satisfies the vigilance test.

$$a_l(K, k, v) = \frac{L . (U_l(k, \mathbf{n}+v) \cap a'_l(K, k, v))}{L - 1 + \sum_{v \epsilon D} \sum_{k=0}^{K} U_l(k, \mathbf{n}+v)}$$

where the constant $L > 1.0$ and the efferent connection $a'_l(K, k, v) = 1$ if $a_l(K, k, v) > 0$ and 0 otherwise. If a K^{th} node has not been allocated to any feature previously, then

$$a_l(K, k, v) = \frac{L . U_l(k, \mathbf{n}+v)}{L - 1 + \sum_{v \epsilon D} \sum_{k=0}^{K} U_l(k, \mathbf{n}+v)}$$

During propagation, at each point, the input within the localized window is multiplied by the same learned weight matrix and thresholded to get the output at U_s layers.

$$U_{s_l}(K, \mathbf{n}) = \varphi\left(\sum_{k=0}^{K}\sum_{v\epsilon D} a_l(K, k, v).U_{c_{l-1}}(k, \mathbf{n}+v)\right)$$

where the threshold function $\varphi(x) = 1.0$ if $x > \theta_l$ and otherwise 0. θ_l represents the threshold value at l^{th} stage to the afferent signal flow..

The size of each plane undergoes approximately a two to one reduction at the U_c layer. The purpose of this is to allow the recognition of distorted inputs and also to cause a compressed representation at the recognition layer. Each neuron at the U_c plane has fixed constant connections from a small localized window D' of 9 neurons arranged in 3x3 array from the U_s layer just below it. Neurons at the U_c layer perform ORing of inputs.

$$U_{c_l}(k, n) = \cup_{v\epsilon D'} U_{s_l}(k, \mathbf{n}+v)$$

The weights connected to the recognition layer are learned using a teacher. Once the system recognizes the pattern after propagation through the afferent connections, the efferent signal flow reconstructs the input pattern. The backward propagation of signals is governed by the following equations.

$$W_{c_l}(k, \mathbf{n}) = \sum_{K=0}^{K}\sum_{v\epsilon D} a'_{l+1}(K, k, v).W_{s_{l+1}}(K, \mathbf{n}-v)$$

$$W_{s_l}(k, \mathbf{n}) = min\{U_{s_l}(\mathbf{n}, k).W_{c_l}(k, \mathbf{n}/2)\}$$

The hierarchical structure of the interconnections between different kinds of cells are indicated in the figure(2). The • indicates a threshold control. At each stage the threshold to the afferent signal was reduced until at least 50% of expected features are extracted. If a stage fails to extract sufficient amount of features, then the threshold of the very first stage is lowered.

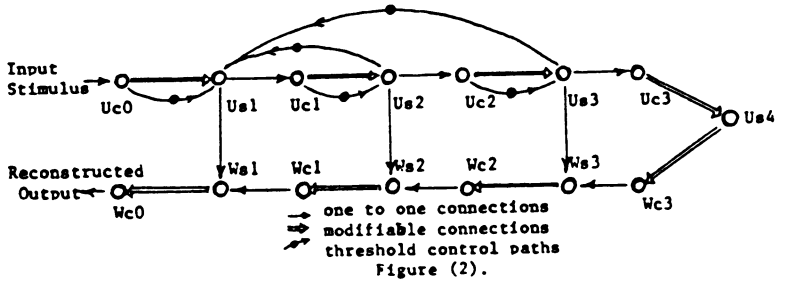
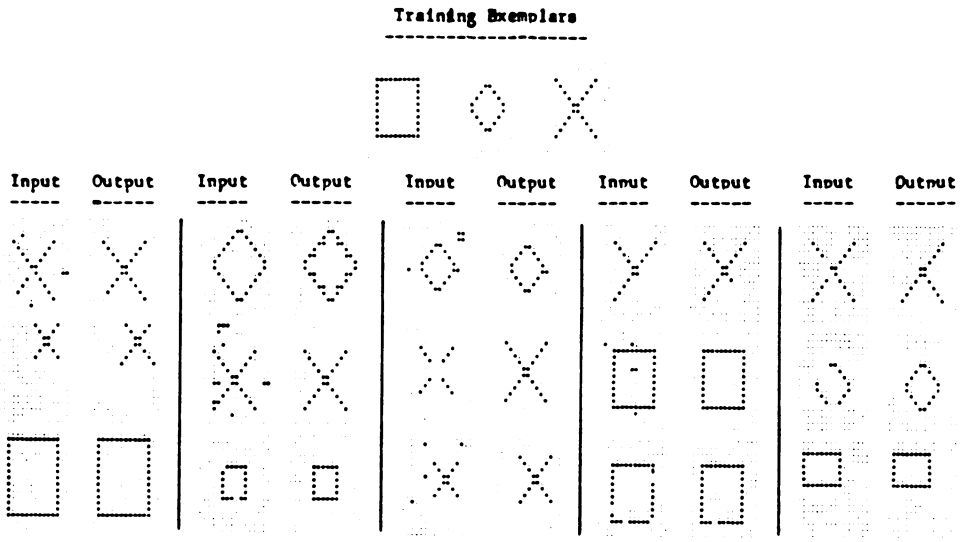


Figure (2).

Simulation Results

Exp 1: The network was trained to recognize three patterns with vigilance 0.8. The trained patterns when presented at the same location, were recognized and reproduced with 100% accuracy. It recognized all test inputs and reconstructed perfectly about 60% of the inputs. Others were either partially reconstructed or produced with added noise. The reconstructed outputs of 15 test patterns are shown in figure (3).



Fipure (3).

Exp 2: The network was trained to identify 10 digits 0...9 with a low vigilance of 0.6. The network recognized all trained digits and reproduced them with 100% accuracy when presented at their original locations. It recognized about 90% of the tested patterns and reproduced perfectly about 50% of recognized patterns. This distortion is due to the low vigilance used because of memory limitation of the computer. If the uncompletely reconstructed patterns were fedback at the input. most of them were improved by the network.

Exp 3: The network was tested to see its capability to incorporate novel training patterns. The network was trained with digit 1 first. then 2. 3 and so on. The network was tested against its performances to recognize and reconstruct previously learned digits after each digit was introduced. It was capable to recognize and reproduce all trained digits even after learning all 10 digits irrespective of the sequence they were trained.

Conclusion

Introduction of ART-1 learning has improved the stability. speed and complexity of Fukushima's neocognitron related network. The new architecture can automatically incorporate novel training patterns without disturbing previous trained information. The network approaches an ideal real time pattern recognizer and autoassociator when the vigilance used for training approaches unity. The ideal network performance is independant of the number of patterns incorporated as far as the memory is available. Our present study is to recognize analog patterns and to reduce the number of weighted connections in the architecture using ART-2 learning principles.

References

[1]. Fukushima K. 'A Neural Architecture for Selective Attention in Visual Pattern Recognition'. Biological Cybentics. 55. 5-15. 1986.

[2]. Oleg G. Jakubowicz. 'Multi-layer Multi-feature Map Architecture for Situational Analysis'. Proceedings of IEEE International Joint Conference on Neural Networks. June 1989.

[3]. Grossberg S. and Carpenter G.A.. 'The ART of Adaptive Pattern Recognition by A Self Organizing Neural Network'. Computer. March. 1988.

A Decoder for Block-Coded Forward Error Correcting Systems

Michael D. Alston
Dr. Paul M. Chau
Electrical & Computer Engineering Department
University of California, San Diego
Mail Code R-007
La Jolla, CA 92093

The hardware model of a neural network for decoding block-encoded digital data is presented. This neural network decoder is designed for use in channel coding digital communication systems to ensure that bits transmitted over the communications channel are received correctly in the presence of noise or interference. Unlike a classical biological neural network, this artificial neural network requires no training: all of its synapses have equal weights. It is the topology of the network, which is intimately related to the topology of the Encoder, which embues this Decoder with its error correcting capability.

The purpose of a forward error-correcting digital communications system, such as the one represented in Fig. 1, is to accomplish the transfer of information bits from Location A to Location B with consistent accuracy in the presence of noise. By encoding the information to be transmitted, at location A, a certain amount of redundancy (in the form of additional bits) is added to the information data. The decoder, at location B, applies its knowledge of exactly how this redundancy was introduced to process the received data reconstructing the original information bits with as much accuracy as possible. Within a particular digital communication system, the relationships which exist between the complexity of the Encoder & Decoder, the level of noise on the Channel, the maximum attainable data throughput (no. of bits/sec), and the minimum attainable Bit Error Rate (no. of errors/no. of bits) are intuitive.

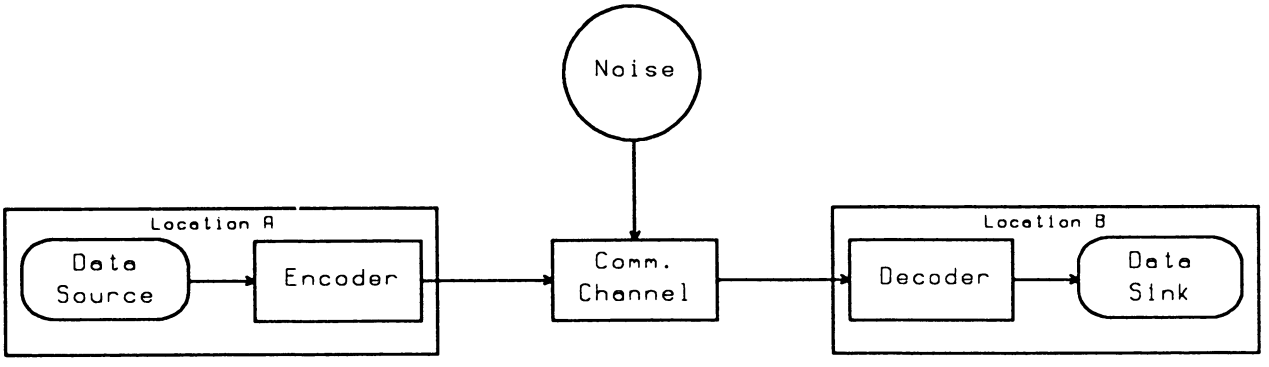


Figure 1 A Forward Error-Correcting Communications System

Diagrammed in Figures 2 and 3 are a block encoder and a neural network decoder for a simple block code. At the Encoder, for every 4 information bits input, a total of 7 channel bits are output. These 7 code bits (n) are comprised of the 4 information bits (k) and 3 parity bits (m). This (7,4) particular code is known as a Hamming Code[1].

Those block codes (n,k) which exactly meet the inequality: $2^m >= n+1$, where $m = n-k$ are called Hamming Codes. By carefully studying the connectivity of the Encoder's parity bit generator and the connections between the Decoder's Data Bit Neurons and the Ex-OR Logic, a definite relationship can be recognized. This Decoder has been "taught" to decode this particular Encoder's output by the way it was constructed. As in an evolving biological neural network system, a fully interconnected system is not implemented, rather only those connections which are of importance in accomplishing the task at hand are present. This artifical neural network will change the states of its neurons until attaining its the "lowest energy level." If the number of errors introduced by the noise on the communications channel is within the error-correcting capability of the block code used, the neurons will settle to the those values representing the original information output by the Encoder. If the number of errors introduced by the noisy communications channel is more than the error-correcting capability of the code, then as with conventional decoders, decoding errors will be observed.

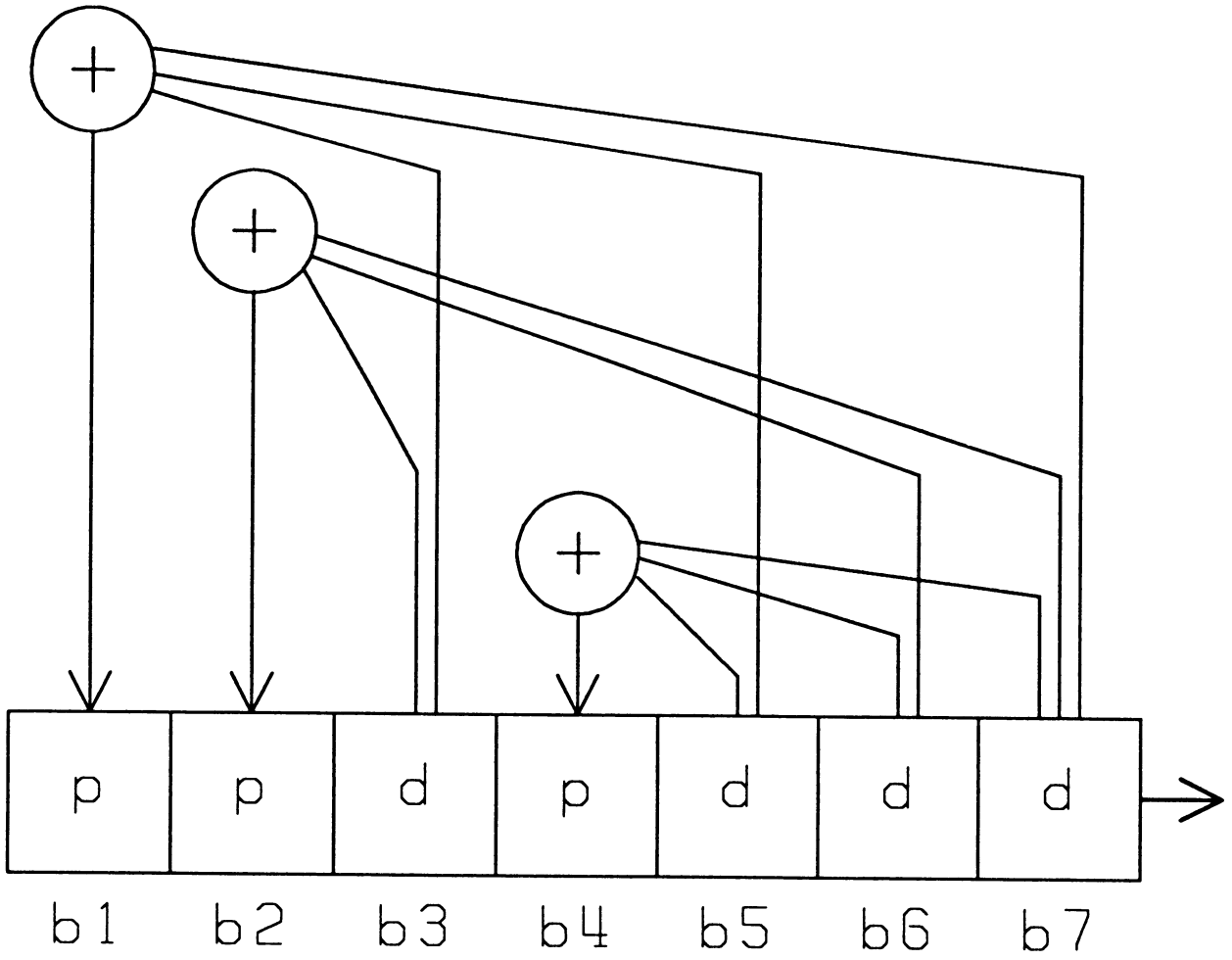


Figure 2 A Block Encoder

This neural network decoder is composed both analog and digital circuits[2]. By implementing the computations required in the type of circuitry most suited to to the function being performed, an area efficient VLSI system can be realized. One of the fundamental operations required within encoders and decoders of binary data is modulo-2 addition, or the exclusive-OR operation. To accomplish this an EX-OR "neurally" would require a multi-layered neural network[3]. In performing the required exclusive-OR operations digitally, the VLSI implementation is more efficient. In contrast, the summing

of the inputs to each Data Bit Neuron is best accomplished with analog circuitry.

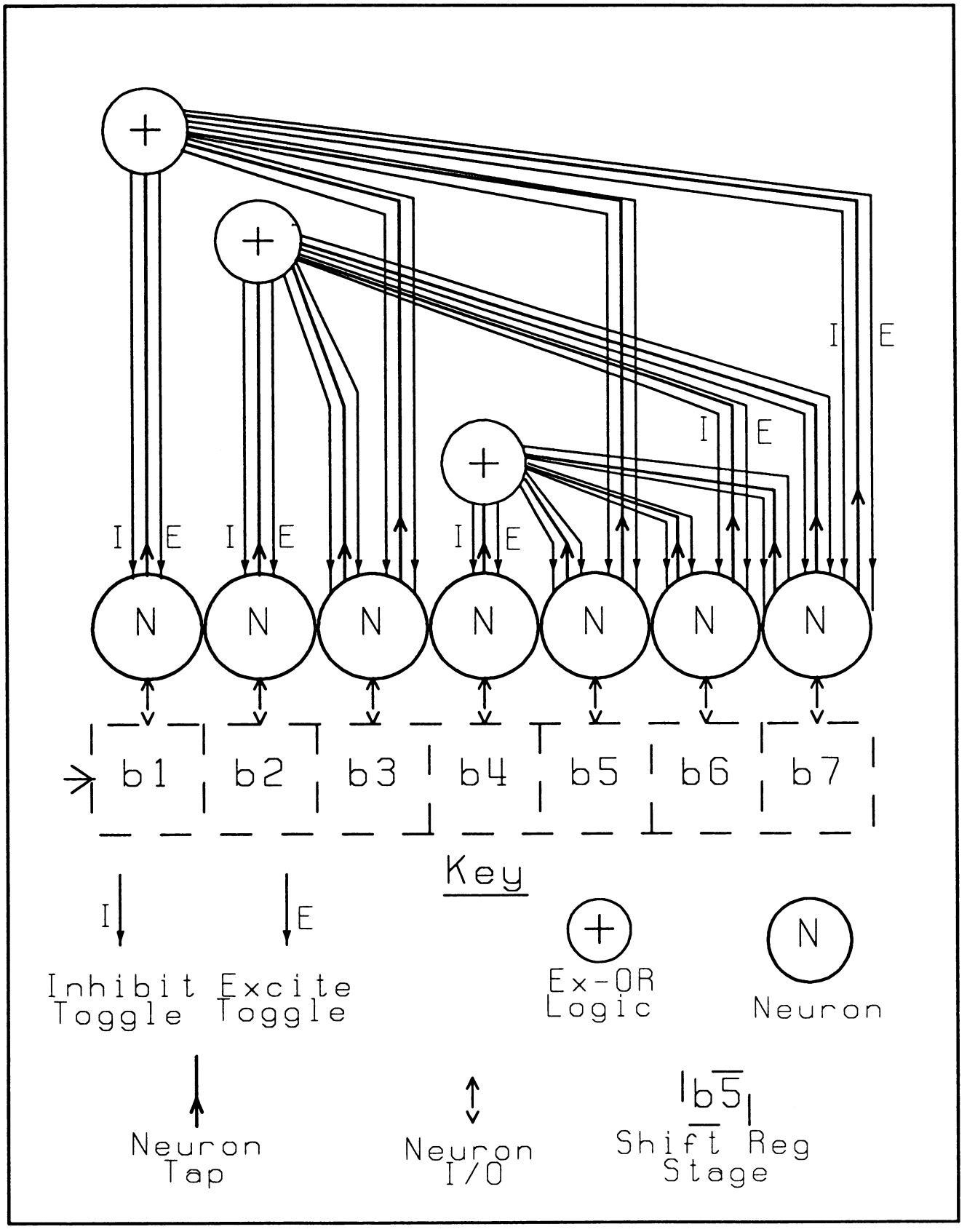


Figure 3 A Neural Network Decoder for Block-Coded Data

To allow tens and even hundreds of synapses[4] to be connected to a single neuron, two separate lines are used to sum the excitatory and inhibitory currents. Each synapse is a programmable current source which controls a differential amplifier. In this manner a two-state synapse can be realized. The small (on the order of 10 uA) excitatory and inhibitory currents generated by the Input Cells are respectively summed (as per Kirchoff's Current Law[5]) and compared within the Data Bit Neurons. Whenever the excitatory current exceeds the inhibitory current, the state of the binary Data Bit Neuron is toggled. The quiescient states of the Neurons, subsequently achieved, represents the decoded data.

Analogous to the synergism of a system which combines the storage capabilities of a Direct Storage Neural Network with the discrimination abilities of the Hopfield Network, the design of this Decoder combines the exclusive-ORing efficiency of digital circuits with the (current) summing capability of analog circuits[6].

References

1. Hamming, Richard W., (1986). Coding and Information Theory. New Jersey: Prentice-Hall, p.5; pp. 39-42.

2. DARPA Neural Network Study, (1988). Virginia: AFCEA International Press, pp. 360-363.

3. McClelland, J. L. and Rumelhart, D. E., (1988). Explorations in Parallel Distributed Processing. Massachusetts: The MIT Press, pp. 123-126.

4. Verleysen, M., Sirletti, B., Vandemeulebroecke, A., Jespers, P., (1989). "Neural Networks for High-Storage Content-Addressable Memory: VLSI Circuit and Learning Algorithm," IEEE Journal of Solid-State Circuits, Vol. 24, No. 3, pp. 562-569.

5. Mead, Carver, (1989). Analog VLSI and Neural Systems. California: Addison-Wesley, pp. 87-88.

6. Kinser, J. M., Caulfield, H. J., Hester, C., (1988). "Error-Correcting Neural Networks," Abstracts of the First Annual INNS Meeting, Boston, New York: Pergamon Press, p. 190.

A Continuous Speech Recognizer Using Two-Stage Encoder Neural Nets

M.T. Anikst, W.S. Meisel, R.E. Newstadt, S.S. Pirzadeh, J.E. Schumacher, P. Shinn, M.C. Soares, D.J. Trawick
Speech Systems Incorporated
18356 Oxnard Street
Tarzana, California 91356

Abstract

In this paper we present a continuous speech recognizer which combines a two-stage hierarchy of neural networks with a dynamic programming (DP) beam-search model. The neural networks encode short-time speech units. The DP beam-search selects the output text by time-aligning the obtained code sequence with the phonetic transcription of all allowable text for an entire continuously-spoken sentence.

Binary-tree structured neural nets, which are trained to maximize the average mutual information between a code alphabet and an alphabet of phonetic classes, are used as encoders. The topology of the networks (i.e., the number of interior nodes and their connections) is learned along with the set of weights from the training data.

Two stages of speech representation, the acoustic frame and the acoustic segment stages, are formed. Each stage utilizes an encoder attempting to retrieve the highest amount of information about the underlying phonetic classes of speech units of that stage. Acoustic-phonetic features are extracted over moving time-context windows and used as input into each encoder network. This allows the networks to form time-shift-invariant decision regions. The 2nd-stage encoder takes as a part of its input the output of the 1st-stage encoder. Time compression occurring between the frame- and segment stages allows wider time-context to be included into the 2nd-stage features thus making them more informative.

Binary-tree structured neural nets scale up well to high-complexity problems such as speech recognition. The amount of labelled speech data and training time required are reasonable.

1. Introduction

Recently there has been a growing realization that combining neural-net based classifiers/vector quantizers (using local time-context acoustic-phonetic features) with DP-based time-alignment models such as HMMs (using global time-sequences of phonetic codes with a language model) can lead to synthesis of a superior speech recognition system [Lippmann 89]. In this paper, we describe a continuous speech recognizer which attempts such a synthesis.

The recognizer uses local time-context acoustic-phonetic features for the encoding (in the information-theoretic sense) of speech by means of two stages of binary-tree structured networks, with intervening segmentation between the encoding stages. Segmentation can be viewed as forming blocks of the adjacent codes thus allowing for a more efficient block-coding. It can also be viewed as a time compression allowing incorporation of more time context into the acoustic-phonetic features.

To get the text of the spoken sentence, the resulting segment-code time-sequence is then time-aligned with the phonetic transcriptions of the grammar-constrained sentences by means of the DP beam-search algorithm using an alignment score statistical model.

Binary-tree structured networks are trained as optimal encoders assigning maximally-informative (on the average) codes to speech patterns at the appropriate stages. The task of training such networks has been extensively addressed in the theory of binary decision trees. [Breiman 84] systematically considered binary decision trees applied to various classification tasks. The processing elements used in the decision nodes of the binary decision trees included the sigma-type units with hard limiter non-linearities. Training criteria ("impurity" criteria) for the binary decision trees included the average leaf-node-conditional class entropy. Training was performed in a top-down node-at-a-time fashion, adding new leaf nodes and maximizing reduction in the average leaf node impurity attained by such additions. It was demonstrated on many practical classification problems that the above procedure results in a suboptimal but "good enough" tree.

[Koutsougeras 88] and [Sun 88] reformulated binary decision trees as neural networks and further developed training based on the minimization of the average node-conditional class entropy (or, equivalently, maximizing the average node-class mutual information). [Sun 88] also extended the type of the processing element used in the decision tree to the single-layer perceptron with n output units. Most recently, [Bichsel 89] discussed usage of the minimum conditional class entropy criterion for training of general layered feedforward networks with hard limiters.

At SSI, we have been using binary-tree structured networks for maximum code-class mutual information encoding in our speech recognition research for some time now. We describe here one of the more successful recognizer architectures and present some test results.

The remainder of the paper is organized as follows. Section 2 contains a brief recognition system overview, Section 3 describes system training aspects, and Section 4 presents test results and discussion. Finally, Section 5 recaps major features of the system and outlines a possible generalization.

2. System Overview

A speech recognizer based on frame- and segment-encoder neural networks is briefly described below. A continuously-spoken sentence's audio is sampled at 16 kHz and is converted into a time-sequence of pitch-synchronous frames of 24 acoustic parameters. At this stage, frames are encoded in such a way as to convey maximum information about their underlying segmentation class identities (the set of 9 broad phonetic classes called the "segmentation" classes is used to characterize acoustic-phonetic segmentation of speech).

To perform frame encoding, the frame time-sequence is scanned by a "moving time window" covering 3 frames. A set of pre-defined feature vectors is extracted from the acoustic parameters of the frames accessed by the window. The frame-encoder neural network (further called "Frame Encoder") takes as input this set of the feature vectors (as a single block-vector extracted from the 3-frame window) and outputs a code for the frame at the center of the window. The Frame Encoder is trained to maximize the average mutual information between its code alphabet and the alphabet comprised of the segmentation classes.

The resulting time-sequence of coded acoustic frames is processed by a Segmenter which forms acoustic segments by merging time-contiguous blocks of frames using their codes and certain code-class statistics. The Segmenter also assigns the most-likely segmentation class to each formed segment. The time-sequence of the acoustic segments with the assigned segmentation classes constitutes the input to the segment-coding stage.

The segment-coding stage processing is similar to that of the frame-coding stage. Namely, segments are encoded in such a way as to convey maximum information about their underlying dictionary class identities (the set of 50 phonetic classes called "dictionary classes" are used for transcribing words in the system's dictionary).

To perform segment encoding, the time-sequence of segments is scanned by a moving time window covering 3 segments. A set of pre-defined feature vectors is extracted from the acoustic parameters of all the frames encountered in the segments accessed by the window. Also the most-likely segmentation classes assigned by the Segmenter to each of the 3 segments in the window comprise an additional, "phonetic feature" vector (with binary components encoding the local representation of the class triplet). The segment-encoder neural network (further called "Segment Encoder") takes as input this set of feature vectors (as a single block-vector extracted from the 3-segment window) and outputs a code for the segment in the center of the window. The Segment Encoder is trained to maximize the average mutual information between its code alphabet and the alphabet comprised of the dictionary classes.

Finally, the time-sequence of coded segments is processed by the Linguistic Decoder which retrieves the text of the recognized sentence. It uses the DP beam-search algorithm to select the phonetic transcription variant of a sentence best matching the sequence of segment codes. The match score statistical model is based on a variant of the information-theoretic decoding model using the Fano metric, see [McEliece 77]. The set of allowable sentences (word sequences) is constrained by a finite-state grammar. Word transcriptions are selected from a dictionary. The sentence text is read out from the matched sentence transcription.

3. Training

The Frame and Segment Encoder neural networks (as well as the Segmenter and the Linguistic Decoder statistical models) are trained using labelled frame data for the isolated continuously-spoken sentences. To initiate the training process, one of the existing Speech Recognizer models (originally trained with a large body of hand-labelled speech collected at SSI) is used as a "bootstrap" model to produce the labelling of the training speech.

The training sentence's audio is processed by the bootstrap Speech Recognizer model. The Linguistic Decoder is then given the known sentence text and is instructed to directly output the sentence transcription best matching the segment-code sequence in the form of segments labelled with dictionary classes (where several class labels may be attached to a single segment). By an off-line process using certain segment code and dictionary class statistics, the dictionary-class labels are propagated from segments to their constituent frames. The obtained labelled data is subsequently used for supervised training of the Frame Encoder, the Segmenter, the Segment Encoder, and the Linguistic Decoder parameters (the encoder training is described in detail further).

Once training is performed, the bootstrap Speech Recognizer model can be replaced with the one just trained, and the training process is repeated iteratively until no improvement in recognition accuracy is observed. In our research, we performed two iterations, but we also added more training data and slightly modified the labelling algorithm, between the first and the second iterations.

The Frame and Segment Encoders are trained as binary-tree structured networks [Breiman 84] using maximization of the average mutual information between the set of leaf nodes and the set of target classes as the

training criterion. Training is performed top-down, starting from the root of the binary decision tree. The processing element associated with each decision node of the tree is a sigma-type unit with hard limiter nonlinearity [Rumelhart 86, Lippmann 87] which accepts the block-vector (of features extracted from the moving time window) as input for each speech pattern. A set of training patterns which was routed to the node by the decision nodes along the path of the tree already built is used for the weight computation.

Training of the processing element of a decision node is limited to subspaces of the weight space corresponding to the feature (sub)vectors which form the input block-vector. Every such subspace is considered in turn, and optimal weight vector components are computed for that subspace while holding the rest of the weight vector components equal to zero. The optimization criterion (i.e., the reduction in the node's average class entropy) is evaluated using the node's set of training patterns. The optimal weight subvector yielding the highest value of the optimization criterion is selected among the ones for the weight subspaces considered. This (suboptimal!) training procedure gives sufficiently good results. It effectively limits dimensionality of the weight space and, thus prevents data overfitting (especially, in the "near-leaf" nodes). Also, it selects the most-informative feature at the given decision node of the binary decision tree.

To train the weights of a given weight subspace, we use the conjugate gradient based search [Press 86] where the gradient of the criterion function with respect to the weights is computed by replacing the hard limiter with the linear threshold non-linearity and gradually annealing that non-linearity to the hard limiter. Once the optimal weights are estimated, we use the hard limiter to send the patterns to the left or the right child node. We may decide to declare the node a leaf node if the highest reduction in the class entropy attained by the "node split" is less than a certain fraction of the node's class entropy.

After the entire binary tree network is created, its performance criterion (i.e. the average mutual information between the set of the leaf nodes and the set of target classes) is evaluated with an independent set of labelled data.

4. Test Results and Discussion

We used 8844 continuously-spoken utterances from 8 female speakers (about 1105 utterances per speaker, 5.7 words per utterance, 41.7 segments per utterance) to train the Speech Recognizer model described in the previous sections. The known content of the training sentences was used to label the frame/segment data with their phonetic class targets. The set of training sentences was specifically designed to be phonetically balanced. Estimates of the average code-class information criterion for the Frame Encoder and Segment Encoder are given in Table 1.

Table 1: Training of Frame & Segment Encoders

	#codes	#classes	avg. class entropy, bits	avg. code-class info, bits	info as % of class entropy
Frame Encoder	712	9	2.66	1.83	68.6%
Segment Encoder	1016	50	4.85	3.18	65.5%

The time compression factor (avg. #frames per utterance / avg. # segments per utterance) was 14.05. Model building took 10 weeks of processing on a Sun-4/280 (for the two iterations of labelling, see section 3).

The trained Speech Recognizer model was tested using independent speech data collected from the 8 female speakers used in training (dependent speakers) and from 4 additional female speakers (independent speakers). We used 130 continuously-spoken test utterances from each of the above 8 speakers. Average utterance length was 10.8 words. Recognition was performed with a finite-state grammar with a perplexity of about 10 (a difficult artificial test grammar), and with a vocabulary of 12195 words (containing about 15 distinct transcription variants for each word entry on the average, i.e., about 180,000 transcription variants). Of the 2187 distinct words occurring in the training set and of the 646 distinct words occurring in the test set, there were 261 common words. A summary of the test results is given in Table 2.

Table 2: Test Results

	avg. word % correct	avg. # segments/utt	avg. decoding time, sec/utt (Sun-4/280)
8 training speakers	87.1%	65.9	34.8
4 independent test speakers	87.9%	67.2	35.0

The results given in Table 2 indicate that a reasonable recognition accuracy can be obtained for a speaker-independent and training vocabulary-independent Speech Recognition model. The encoding is performed in real time

on SSI's Phonetic Engine [Meisel 89]. A decoding speed of about 7 times the utterance duration is achieved in a general purpose computing environment (a Sun-4/280).

5. Conclusion

We have described a Continuous Speech Recognizer based on the two-stage encoding performed by binary tree structured neural networks. Using acoustic-phonetic features extracted over the moving time window accessing certain time-shifted context allowed the neural networks to form time-shift-invariant decision regions. The average code-class mutual information criterion applied to the binary tree structured neural network allowed a node-at-a-time training of the network, with the data-driven network topology design obtained simultaneously with the weight estimation. Finally, the time compression that occurred with segment formation allowed more time-context to be included in the scope of the segment-stage acoustic-phonetic features, and reduced the number of codes conveying phonetic information by a factor of 14. It also allowed staged encoding where the acoustic-phonetic features of the codes output at the frame stage are used as input into the segmentation stage. The final time alignment between the segment-code sequence and the sentence transcription was performed by a DP beam-search algorithm. The text was retrieved from the best alignment obtained.

One possible generalization of the given architecture is a multistage hierarchy of binary-tree structured encoders which performs segment formation more gradually. The decision to declare the stage terminal is made by analyzing the time compression factor of that stage as well as the increase in the average code-class mutual information extracted by that stage's encoder network. Also, the increase in the "bottom-line" performance, the speech recognition results obtained by DP, should be used in making that decision.

Acknowledgement

We would like to acknowledge significant contributions to our research made by J.E. Lloyd, W.A. Wittenstein and J. Hemela, former members of the SSI's R&D group. Also, we would like to thank L. Breiman and J.H. Friedman for their early support of our work with the binary decision trees. Finally, we are greatful to the rest of the SSI staff for their work on the Speech Recognizer.

References

M. Bichsel, and P. Seitz. "Minimum Class Entropy: A Maximum Information Approach to Layered Networks," *Neural Networks*, Vol. 2, pp. 133-141, 1989

L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*, Wadsworth International Group, Belmont, Calif., 1984.

C. Koutsougeras, and C.A. Papachristou. "Training of a Neural Network for Pattern Classification Based on an Entropy Measure," in *Proceedings of the 2nd International Conference on Neural Networks*, San Diego, Calif., 1988, Vol. I, pp. 247-254.

R.P. Lippmann. "Review of Neural Networks for Speech Recognition", *Neural Computation*, Vol.1, pp. 1-38, 1989

R.P. Lippmann. "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, pp. 4-22, April 1987.

R.T. McEliece. *The Theory of Information and Coding*, Addison-Wesley Publishing Company, Inc., Reading, Mass., 1977.

W.S. Meisel, P.C. Shinn, D.J. Trawick, and W.A. Wittenstein. "Speech Representation and Speech Understanding", Final report under DARPA Contract DAAH01-87-C-0953, April 1988.

W.S. Meisel, M.P. Fortunato, W.D. Michalek. "A Phonetically-Based Speech Recognition System", *Speech Technology*, Apr/May 1989, pp. 44-48

W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling. *Numerical Recipes*, Cambridge University Press, 1986

D.E. Rumelhart, and J.L. McClelland. *Parallel Distributed Processing*, Vol. 1 & 2, MIT Press, Cambridge, Mass., 1986.

G.Z. Sun, H.H. Chen, and Y.C. Lee. "Parallel Sequential Induction Network: A New Paradigm of Neural Network Architecture," in *Proceedings of the 2nd International Conference on Neural Networks*, San Diego, Calif., 1988, Vol. I, pp. 489-496.

A NEURAL NETWORK ARCHITECTURE FOR SILHOUETTE COMPLETION

E. Ardizzone, A. Chella*, S. Gaglio, F. Sorbello.

DIE - Dipartimento di Ingegneria Elettrica
University of Palermo
Viale delle Scienze, 90128 Palermo, Italy
Tel. +39.91.595735

**CRES - Centro per la Ricerca Elettronica in Sicilia*
Via Regione Siciliana 49, 90046 Monreale (Palermo), Italy
Tel. +39.91.6404501
E-mail: chella@ipacres.bitnet

ABSTRACT: A two-layered neural network architecture for completing silhouettes from open edges extracted by classical early vision algorithms is proposed. The main processes involved in the proposed architecture are the short-range competition, the center-surround competition and the long-range cooperation between the orientations of the existing edges, along with the hypothesis of locality, isotropy, symmetry. An example of the implementation of the architecture is reported.

1. INTRODUCTION

Classical early vision algorithms of edge detection from real images give rise to many problems about continuity and regularity of extracted edges: the edge of an object is generally not well defined in an image and it depends on many conditions like position, illumination, background, and so on [1].

Instead, the human eye clearly perceives the edges of an object, also when they are not well defined: uncertainty and fuzziness are eliminated by a great number of low-level elaborations of the perceived image. Moreover, in the process of edge perception some edges not really present in the image are clearly seen, and other edges that are present are not clearly seen, such as in Kanizsa triangles [2].

Ullman [3], Grossberg and Mingolla [4], Sejnowski and Hinton [5], among others, developed theories about the generation and completion of edges and suggested implementations by neural networks.

A simple and efficient two-layered neural network suitable to complete edges detected by early vision algorithms is proposed. The network is based on a competitive-cooperative architecture and it is a refinement of architectures characterized by more layers and connections previously developed by the authors [6,7].

A simulation of the network architecture proposed with a limited angular resolution has been tested on simple patterns of applications and the results seem to be promising.

2. THE ARCHITECTURE OF THE NETWORK

The process of completion of the edges is assumed to be a local process which is also isotropic and symmetric: the generated completing edges must be invariant with respect to the position and the direction of existing edges. Completing edges are also locally generated: they are generated only from the knowledge of tendency of the end parts of the local existing edges and not from the knowledge of the whole edge of the object.

The neural network architecture proposed is based on competitive-cooperative interactions between units. The network is made up of two layers: in the first layer only the competitive interactions are carried out, while in the second layer only the cooperative interactions are carried out.

The guide-lines of the network architecture are: the short-range competition between orientations to recognize local prevalent orientation; the short-range center-surround competition between neighbouring orientations to ensure line cuttings at the corners; the long-range cooperation between the like-oriented units to force completion of non continuous edges.

The building block of the architecture is a cluster of some number of simple units, each of them is sensitive to a specific orientation; the number of these units depends on the angular resolution. Fig. 1 shows a cluster related to six orientations.

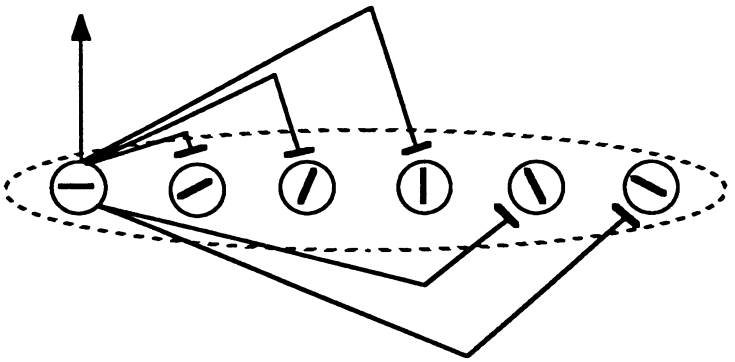


Fig.1: A cluster related to six orientations; the connections refers to the horizontal sensitive units.

A mechanism of lateral inhibition is implemented allowing competition between orientations be performed by units in the cluster: if the output value of an unit emerges, the other units are inhibited; so a strong unit tends to become much stronger and a weak unit tends to become much weaker.

Fig. 2 shows the basic scheme of the whole network architecture; for the sake of simplicity the connections refer only to the horizontal sensitive units and the internal units and connections of the clusters are not reported.

The first layer of the network is made up of orientation fields: each unit in the cluster is sensitive to orientations in a small window of the picture under consideration then it competes with other units in the same cluster: winning orientation tends to emerge. The orientation fields overlap in the picture; this is to ensure a better interpolation of the change of directions of edges.

A short-range center-surround competition within each orientation is also performed in this layer: each unit in a cluster inhibits all like-oriented units in the neighbouring clusters (fig. 3a); this is to prevent output lines flow out beyond their endings when a corner of the edge is found.

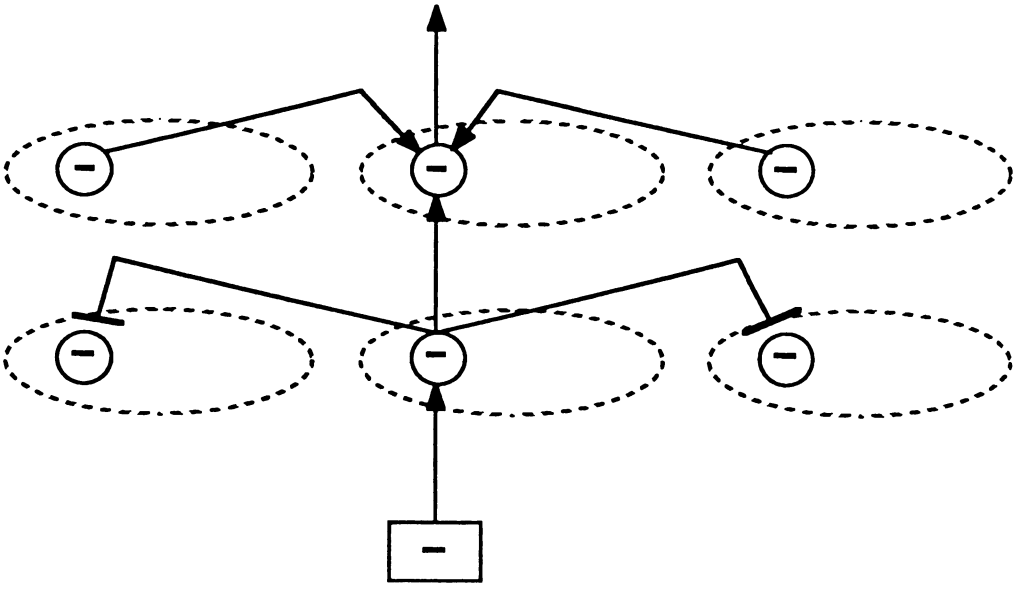


Fig.2: The neural network architecture; the connections refer to the horizontal sensitive units; the other internal units and connections of clusters are not reported.

The second layer performs the long-range oriented cooperation between units: in the cooperation process, each unit receives a strong activation from the like-oriented units of the clusters in the neighbouring position of the previous layer (fig. 3b), to force a completion of the open edge.

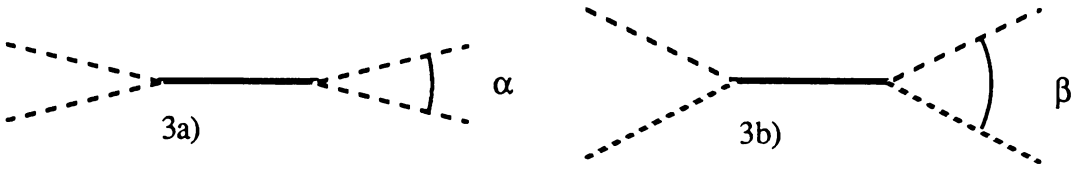


Fig.3: Angles related to the connections of like-oriented units.

The transfer function of the units is chosen to be linear with a low threshold to ensure the noise rejection: the units responding to short random lines do not cooperate with other neighbouring like-oriented units so their low activations are rejected.

For the sake of isotropy and symmetry the weights of the connections are invariant with respect to the position and the direction: several weights of the networks are linked together, so there are only a few weights to fix. At this stage of the development of the network architecture these weights are hard-wired with values empirically fixed through a process of trial and error, so there is not a learning phase of the network.

3. THE SIMULATION OF THE NETWORK ARCHITECTURE

An implementation of the neural network architecture with an angular resolution of six orientations has been tested on simple patterns by using a neural network simulator developed in VMS COMMON LISP running on a VAXStation II/GPX [8].

All the weights are real numbers in the range [-1,1]; as described in the previous chapter, all the weights are initially hard-wired so they not change.

The weights of the connections between each orientation at the layers are fixed at 0.9, to obtain a little loss of information between layers; the weights of the inhibitory connections between units inside the cluster are fixed at -0.6, to obtain a strong inhibitory process.

In the center-surround interaction process, each unit inhibits the like-oriented units at the neighbouring positions. The inhibitory strength of the connections is a bell-shaped function of the angular difference between orientations; the max amount of the inhibition strength is fixed at -0.3 to obtain a weak inhibitory process.

In the cooperative process the weights of the excitatory connections between the like-oriented units in the previous layer are also a bell-shaped function of the angular difference between orientations, but the max amount of the excitation is now fixed at 0.8, to ensure strong cooperative interactions.

The transfer function of the units in the layers are linear with a low threshold fixed at 0.2 to ensure the noise rejection.

Results of simulations of the network architecture are reported in fig. 4. Fig. 4a shows the effect of oriented cooperation: the lines are correctly completed along their directions and there are no activations between them in perpendicular directions. Fig. 4b shows the effect of the center-surround interactions: the cooperation is actually inhibited around the corners of the edges. Fig. 4c shows an example of cooperation between like but not equal orientations, along with the noise rejection.

4. CONCLUSIONS

The neural networks architectures based on competitive-cooperative interactions, together with the hypothesis of locality, isomorphy and symmetry, have been currently developed in many machine vision tasks other to the silhouette completion, as in the stereo matching algorithms and in the interpretation of apparent motion [1]; these architectures are generally quite simple and they seem to be very promising.

Future developments of the architecture proposed are oriented to build a more adaptive architecture, with a learning phase to fine tune the weights and the thresholds by an appropriate adapting law. Other developments are oriented to complete the architecture with some hidden units which may build some sort of internal representation of the task described.

5. REFERENCES

[1] Marr, D.,*Vision*. W. H. Freeman and Co., New York,1982.

[2] Kanizsa, G., Subjective contours, *Scientific American* **234**, 48-64, 1976.

[3] Ullman, S., Filling-in the Gaps: The Shape of Subjective Contours and a Model for Their Generation, *Biological Cybernetics* **25**, 1-6, 1976.

[4] Grossberg, S., Mingolla, E., Neural Dynamics of Form Perception: Boundary Completion, Illusory Figures, and Neon Color Spreading, *Psychological Review* **92**,173-211, 1985.

[5] Sejnowski, T.J., Hinton, G.E., Separating Figure from Ground with a Boltzmann Machine, in: Arbib, M.A., Hanson, A.R. (eds.), *Vision, Brain and Cooperative Computation*. MIT Press, Cambridge, Massachusetts, 1987.

[6] Ardizzone, E., Chella, A., Gaglio, S., Sorbello, F., Vassallo, G., A Neural Network Architecture For Boundary Contour Cleaning, in: Caianiello, E. (ed.), *Parallel Architectures and Neural Networks. Second Italian Workshop*. World Scientific Publishing Co., Singapore (in press).

[7] Ardizzone, E., Chella, A., Gaglio, S., Sorbello, F., 3-D Computer Graphics,Vision and Expert Systems, in Nicolini, C. (ed.), *Towards the Biochip*. World Scientific Publishing Co., Singapore (in press).

[8] Chella, A., A Neural Networks Simulator and its Interface with Semantic Networks, CRES Technical Report, 1989.

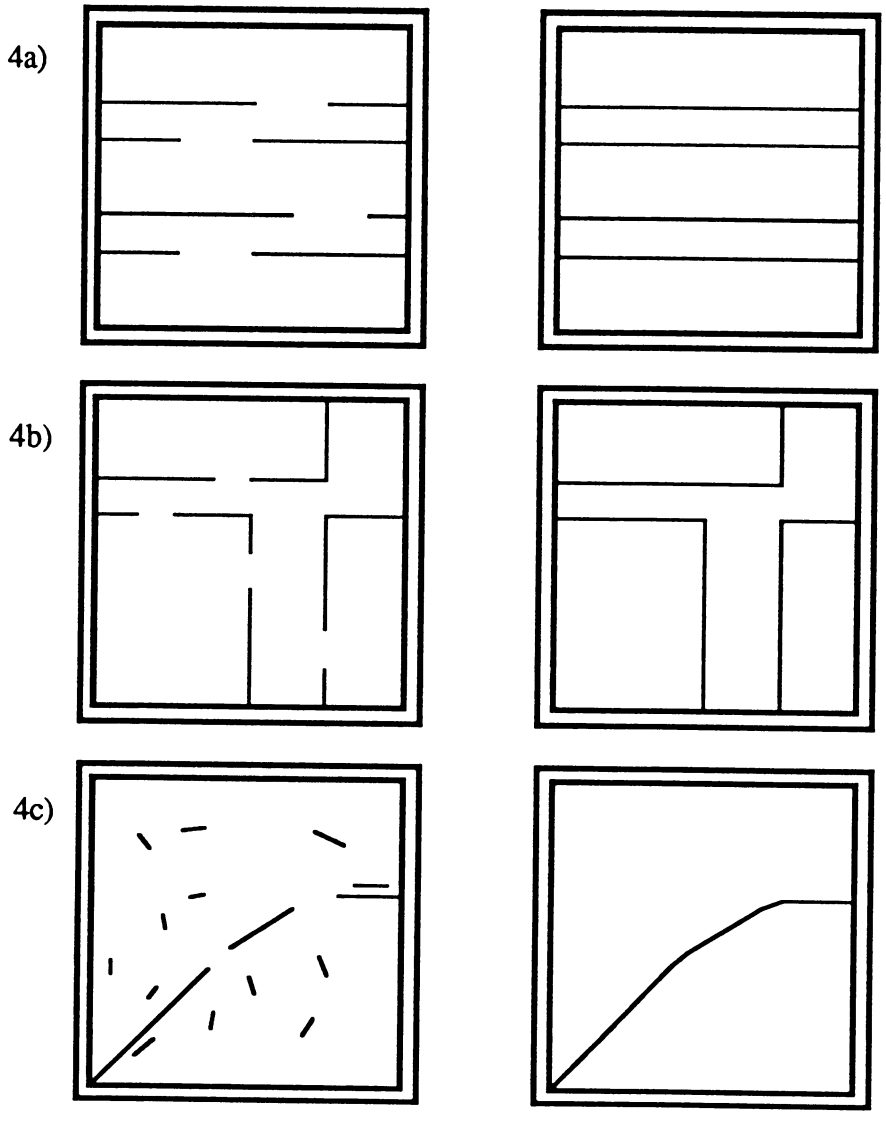


Fig.4: Input and output of the simulation.

ENHANCEMENT OF DETECTION OF DENSE MULTIPLE TARGETS THROUGH LATERAL SUPPRESSION AMONG OVERLAPPING NEURAL NETWORKS

Mohammed Arozullah and William J. Semancik
The Catholic University of America
Washington, D.C. 20064

Abstract

The problem of detecting point targets in images takes on many aspects of the equivalent radar problem. The main difference is the dimensionality of the problem. Common signal processing techniques place an evaluation window around a point or pixel of interest and make an estimate of the background noise intensity to determine a detection threshold. Many of these techniques suffer considerable degradation in the probability of detection of the desired signal or increased false alarm rate in the presence of multiple strong targets within the evaluation window. This paper utilizes a neural network approach to processing the input data whereby detection decisions are used to suppress the effect of strong interfering targets in the decisions made in adjacent windows. This enhances the detection of weak targets.

Introduction

This paper addresses the problem of detecting multiple closely spaced targets in infrared images against a homogeneous background. One typical method of determining the decision threshold is to evaluate the arithmetic average of the signal intensity of all pixels within a window. Nitzberg [1] analyzed this type of processor for its performance versus the performance of the optimum Bayesian detector as a function of false alarm rate. Gandhi and Kassam [2] furthered the analysis of this as well as other types of signal processors when interfering targets or clutter edges are present. The latter effort showed vulnerability of the arithmetic mean processor to clutter edges and multiple targets. We propose to modify the arithmetic mean processor through the use of the neural network shown in figure 1. The advantages of this technique are that it can process an entire image array with a processor with a complexity on the order of the input array itself and that it will process the entire input array simultaneously thus reducing the time required to reach target decisions.

Network Description and Principles of Operation

The two output classes from the problem as stated can be separated linearly. Therefore only a single processing layer is needed in figure 1. The processing nodes are allocated one per evaluation window. Only the pixel at the center of the evaluation window directly connects to the processing node for that evaluation window. Output decisions from the processing node are fed back to a gating layer node. The pixel at the center of the evaluation window also connects to the gating layer. The gating layer turns off the output of the pixel if a positive target detection decision is made. Since all other processing nodes that require the data from a pixel receive it from a node in the gating layer, the effect of a target detection is to remove that pixel from all other evaluation windows.

The statement of the problem is such that it is was possible to derive the exact performance of the network using known weights. This enabled the theoretical comparison of results using this network topology against those achieved with more common processors. This does not imply that only the fixed weight case is of interest.

Theory of Operation

On the initial pass of data through the network, each subnet will act exactly as the arithmetic mean processor that has been analyzed by others. For a single fluctuating target in homogeneous background with an exponentially distributed background, the performance is given by: [2]

1) $P_d = [1+T/(1+S)]^{-N}$

where P_d = the probability of detection, T = the proportionality constant (the weight in the neural network, S = the signal to noise ratio of the target, and N = the nominal number of pixels in the evaluation window. The probability of false alarm is given by setting S equal to zero:

2) $P_{fa} = [1+T]^{-N}$.

Figure 2 shows the performance of this processor.

Reference [2] also covers the case of r interfering targets:

3) $P_d = [1+(1+I)T/(1+S)]^{-r}[1+T/(1+S)]^{r-N}$

4) $P_{fa} = [1+(1+I)T]^{-r}[1+T]^{r-N}$

where I is the signal to noise of the interfering target.

For the neural network model with lateral suppression, equations 3 and 4 are modified as shown below for the two interfering target case. The overall probability of detection for target 1

is then given by equation 5.

5) $P_{d1}' = [1-P_{d2}]P_{d1} + P_{d2}[1+T/(1+S_1)]^{1-N}$.

Let P_{d1} be the probability of detection of target 1 without suppression and is of the form of 3) where S_1 refers to target 1, r = 1, and I refers to target 2. Similarly P_{d2} is of the same form where S_2 refers to target 2 and I refers to target 1. T is given again by equation 2). P_{d1}' is plotted in Figure 3 with fixed power in the primary target and varying power in the secondary target. Figure 5 shows the results of a simulation where the primary target power was 3 dB below the secondary target. Without suppression, the weaker target was not found.

The three target case was also evaluated. The resulting P_d contains terms covering cases where neither secondary target is detected, either is detected but not both, and when both secondary targets are detected. This is plotted in Figure 4 with fixed power in the primary target and varying but equal powers in the secondary target. This situation was simulated where one strong target and two weak but unequal power secondary targets were present. In cases where the neural network processor found all three targets, the arithmetic mean processor could find only the strongest target. A sample simulation output is shown in figure 6.

Discussion of Results

The neural network processor did increase the probability of detection of multiple targets over that of the arithmetic mean processor. As can be seen from Figures 3 and 4, initially there is a decreased probability of detection as compared with the single target case as the secondary target increases in power. As the power continues to increase in the secondary target, the probability of detection rises and in the limit, the probability of detection surpasses that of the single target case. This is actually reasonable to expect since the threshold will decrease due to fewer points being averaged to form the threshold. The cost of this rise in detection is an increase in the probability of false alarm.

Conclusions

The use of neural networks and lateral suppression techniques appear to have promise to handle the multiple target detection problem. Further work must be done to adapt the network to handle non-homogeneous background.

References

[1] R. Nitzberg, "Analysis of the Arithmetic Mean CFAR Normalizer for Fluctuating Targets," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-14, no. 1, pp. 44-47, Jan. 1978.

[2] P. Gandhi and S. Kassam, "Analysis of CFAR Processors in Nonhomogeneous Background," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-24, no. 4, pp. 426-445, July 1988.

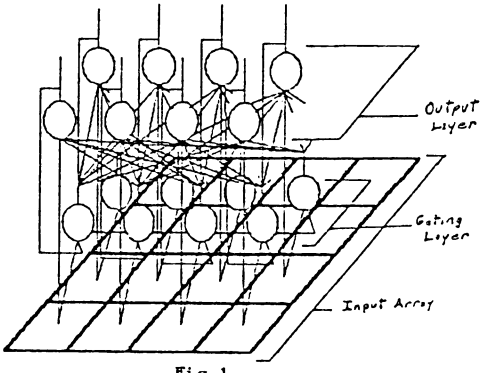


Fig. 1
Network Diagram Showing Input Array, Gating Layer, and Output Layer

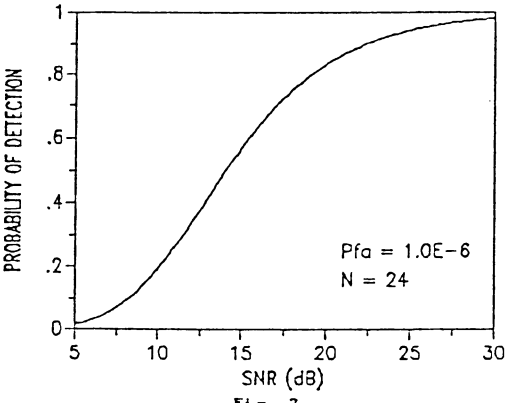


Fig. 2
Detection Performance of Arithmetic Mean Processor in Homogeneous Background

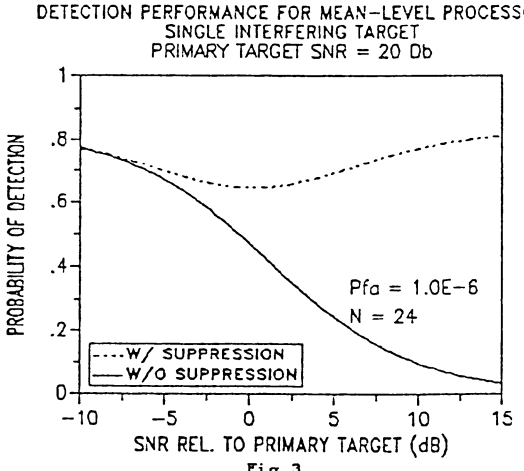


Fig. 3
Primary Target Detection Performance of Arithmetic Mean Processor versus Neural Net Implementation Using Lateral Suppression Single Interfering Target Case

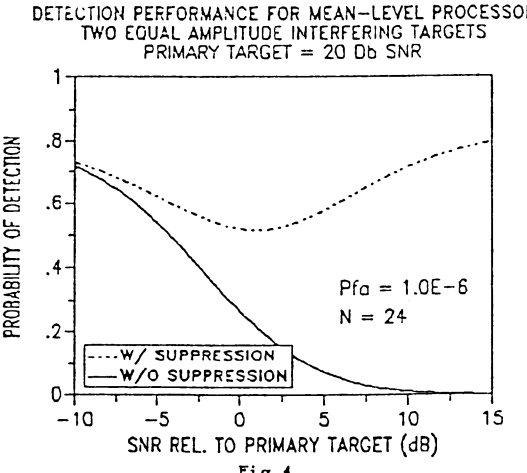


Fig. 4
Primary Target Detection Performance of Arithmetic Mean Processor versus Neural Net Implementation Using Lateral Suppression Dual Interfering Target Case

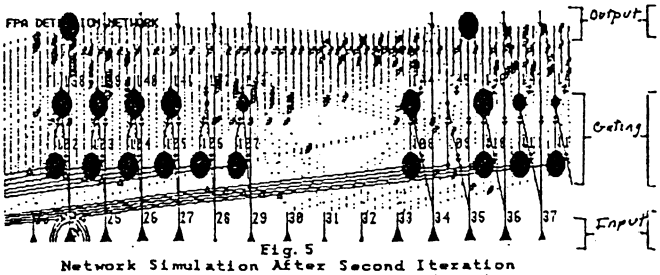


Fig. 5
Network Simulation After Second Iteration One Strong Target and One Weak Target

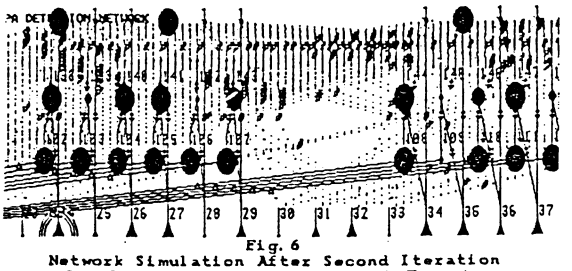


Fig. 6
Network Simulation After Second Iteration One Strong Target and Two Weak Targets

Multilayer Back-Propagation Network for Learning The Forward and Inverse Kinematics Equations

Francisco J. Arteaga-Bravo
School of Information Technology and Engineering
George Mason University
Fairfax, Virginia 22030

Abstract

Forward and Inverse Kinematic Equations (FKE and IKE) are very important in Robotic Manipulators. An approach is presented for solving them by means of Multilayer Neural Networks (MLN) and the Back Error Propagation (BEP) Algorithm [1]. The Solution for the analytical equations is compared with the Network Output. The Simulation is performed in a Two Degrees-of-Freedom (DOF) Robot Arm and the Results are plotted. The Neural Nets learn successfully the indicated path . The Network Architectures are shown as well as the parameters involved during the Simulation. The approach given here can be expanded for other types of Manipulators by adapting the Network Architecture.

1 Introduction

The Solutions of the FKE and IKE are time-consuming tasks. For high number of DOF the IKE can result very complicated to solve. Different methods for solving this equation have been developed [2],[3]. In simple cases, a closed form solution can be accomplished by geometric relationships between joint angles and cartesian coordinates. For non-redundant manipulators, or for manipulators with redundant DOF, numerical methods can be implemented to determine the joint displacements [4],[5]. This type of solution is generally much slower than the corresponding closed-form solution. An alternative solution is the application of Neural Network Processing for solving the FKE and the IKE.

An approach for the solution of the differential motion relationship

$$\dot{x} = (\partial f/\partial q)\dot{q} \tag{1}$$

where $\dot{x}$ and $\dot{q}$ are cartesian and joint velocities, respectively, and $\partial f/\partial q$ is the $m \times n$ Jacobian matrix J, has been presented by Guo and Cherkassky [6]. This approach was based on the Jacobian Control Technique. A similar approach by Yeung and Bekey [7] shows a Neural Network which learned to control a robot arm by learning independently the different entries of the inverse Jacobian matrix. Another approach, presented by Guez and Ahmad [4], consisted in a hybrid solution using a Multilayer Feedforward Network to obtain an initial value and providing it to an iterative algorithm.

The FKE can be expressed as

$$x = f(q) \tag{2}$$

where f is a continous non-linear function with known parameters and structure, and each joint vector $q(n \times 1)$ is associated with a cartesian vector $x(m \times 1)$, and also deals with the solution of the Inverse Kinematics Equation (IKE)

$$q = f^{-1}(x) \tag{3}$$

where this inverse mapping may have many q's associated with each x.

The main objective in the paper is to demonstrate the capability of learning different trajectories by using the BEP Algorithm. The MLN is trained to learn a particular path (input vector x) with the corresponding vector q. Then, the network is tested with an input x_2, in the same circular path as x_1. The values for the output vector q given by the MLN are very close to the given by the IKE. Good results are also obtained for the case of the FKE.

2 Statement of the Problem

A representation for the two DOF manipulator is shown in Figure 1.

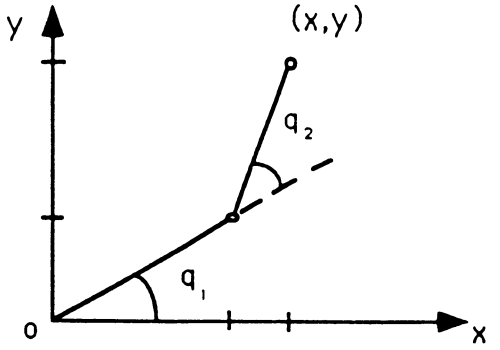


Figure 1. Two DOF Planar Manipulator.

2.1 Forward and Inverse Kinematics

For a given joint vector q the corresponding coordinates x and y can be determined by:

$$x = l_1 cos q_1 + l_2 cos(q_1 + q_2) \tag{4}$$

$$y = l_1 sin q_1 + l_2 sin(q_1 + q_2) \tag{5}$$

A closed form solution for q_2 can be derived from the manipulator geometry as,

$$q_2 = cos^{-1}[(x^2 + y^2 - l_1^2 - l_2^2)/2l_1 l_2] \tag{6}$$

and q_1 can be determined in terms of q_2, x, and y as:

$$q_1 = tan^{-1}(y/x) - tan^{-1}(l_2 sin q_2/(l_1 + l_2 cos q_2)) \tag{7}$$

2.2 Neural Network Learning

A circular path for the 2 DOF manipulator is determined from the expressions:

$$x = r cos\beta \tag{8}$$

$$y = sin\beta \tag{9}$$

For increments $\Delta\beta$ in the angle β (and a fixed r), a circular path is derived. For different $\Delta\beta$ (same r), different values for x and y are obtained, and q is determined from the IKE.

The Network can be trained to learn a first vector x_1 and the corresponding vector q_1. For the same path, but different $\Delta\beta$, the network is given a vector x_2 as input. The MLN output (vector q_2), is expected to be similar to the one yielded by the IKE. Same task is performed for the FKE and the MLN.

Once the Network has learned a specific set of input and output vectors and we make an appropriate distribution of the values in the desired data universe, this Network can give very approximate output results for other set of input vectors. This is the main motivation for this paper.

3 Technical Approach

A three layer feedforward Network with 2,4,and 2 nodes for the input, hidden, and output layers, respectively is used for simulating the FKE. A Multilayer Feedforward Network with 2 hidden layers, 4 nodes in each one, is used for simulating the IKE. For the case of FKE, q is the input vector and x is the output vector. The Network is trained to learn the cartesian coordinates for a specified set of joint angles q_1 and q_2. For the case of the IKE, the 4 layers network is given x as input and q as desired output (training data) , so the Network learns joint angles for a specified set of coordinates. The BEP algorithm is applied in both simulations. The BEP Multilayer Simulator is available on the VAX 8530 (Computer Science Dept).

4 Examples and Results

A set of data for the 2 DOF manipulator (with $l_1 = 2$ and $l_2 = 1$) was obtained from (8) and (9) letting $r = 3$ and β varying from 0° to 90° in steps of 10 degrees. The 2-4-2 and 2-4-4-2 Networks were trained to learn the FKE and IKE respectively, for the specified circular trajectory. The MLN learned these training data very well with η (learning rate) equal to 0.9 in both cases. The momentum term constant α was 0.9 for the FKE case and 0.3 for the IKE case. In the same path (r=3), β was incremented from 5° to 85° ($\Delta\beta = 5°$) and last value $\beta = 90°$. A set of data (x, y) obtained from (8) and (9) was given to the 2-4-4-2 Network just trained. The results were very approximate to the obtained with the IKE, as we can see in Figures 2 and 3. The maximum magnitude difference is about 0.08 (for x coordinate) when $\beta = 90°$. The results for the comparison FKE and the 2-4-2 Network are shown in Figures 4 and 5. The maximum difference was about 0.3° (for q_1) when $\alpha = 60°$. The training data was mapped to values in the range [0,1] in order to make more efficient use of the activation function and to obtain better results in trainig the Networks.

5 Conclusions

An approach for the solution of the FKE and IKE has been presented. The solution is based on Multilayer Network for learning different paths with the BEP algorithm. The simulation results for the two DOF manipulator indicate the possibility of applying this approach to other types of manipulators. Training the MLN appropriately yields very good results and can save the effort of computing the FKE and IKE each time. Future research tasks will focus on modifying the BEP algorithm and adapt it to the particular manipulator environment in order to obtain better results.

References

[1] David E. Rumelhart, James L. McClelland, and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Vol. 1 and 2, Cambridge, MA: MIT Press, 1986.

[2] K.S. Fu, R.C. Gonzalez, and C.S.G. Lee, *Robotics Control, Sensing, Vision and intelligence.* New York, NY: McGraw-Hill Book Company, 1987.

[3] Richard P. Paul, *Robot Manipulators: Mathematics, Programming, and Control.* Cambridge, MA: MIT Press, 1981.

[4] Allon Guez and Ziauddin Ahmad, "Accelerated Convergence in the Inverse Kinematics Via Multilayer Feedforward Networks", *Proc. Int. Joint. Conf. on Neural Networks*, Washington D.C., June 18-22, 1989, pp. II341-II344.

[5] J.J. Hopfield and D.W. Tank,"Neural Computation of Decision Optimization Problems",*Biological Cybernetics*, 52, 1-12,1985.

[6] J. Guo and V. Cherkassky, "A Solution to the Inverse Kinematic Problem in Robotics Using Neural Network Processing", *Proc. Int. Joint. Conf. on Neural Networks*, Washington D.C., June 18-22, 1989, pp. II299-II304.

[7] Dit-Yan Yeung and George A. Bekey, "Using a Context-Sensitive Learning Network for Robot Arm Control", *Submitted to IEEE International Conf. on Robotics and Automation*, Scottsdale, Arizona, May 14-19, 1989.

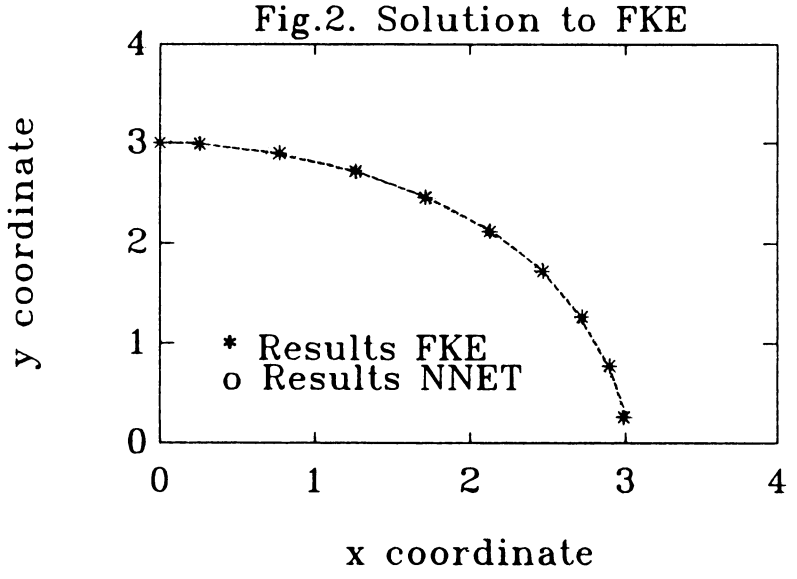


Fig.2. Solution to FKE
y coordinate
x coordinate
* Results FKE
o Results NNET
0
1
2
3
4

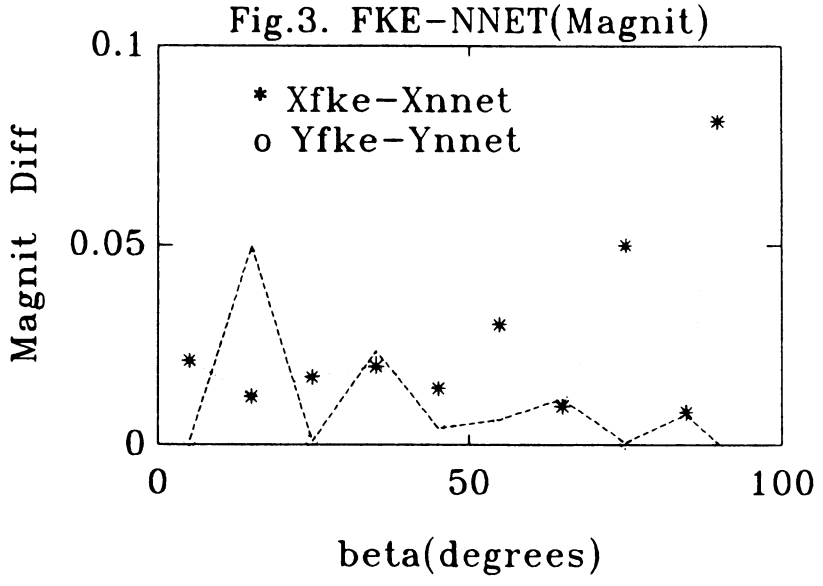


Fig.3. FKE-NNET(Magnit)
Magnit Diff
beta(degrees)
* Xfke-Xnnet
o Yfke-Ynnet
0.1
0.05
0
50
100

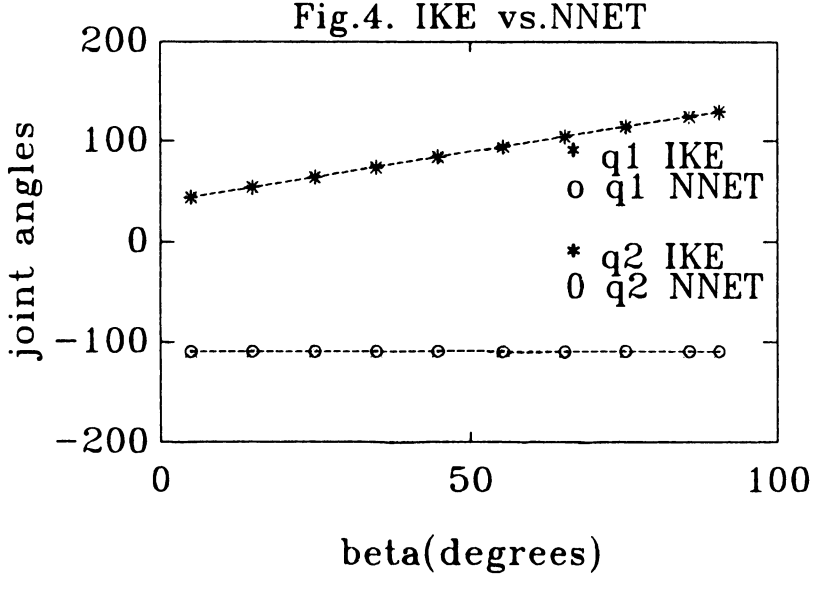


Fig.4. IKE vs.NNET
joint angles
beta(degrees)
* q1 IKE
o q1 NNET
* q2 IKE
0 q2 NNET
200
100
0
-100
-200
50

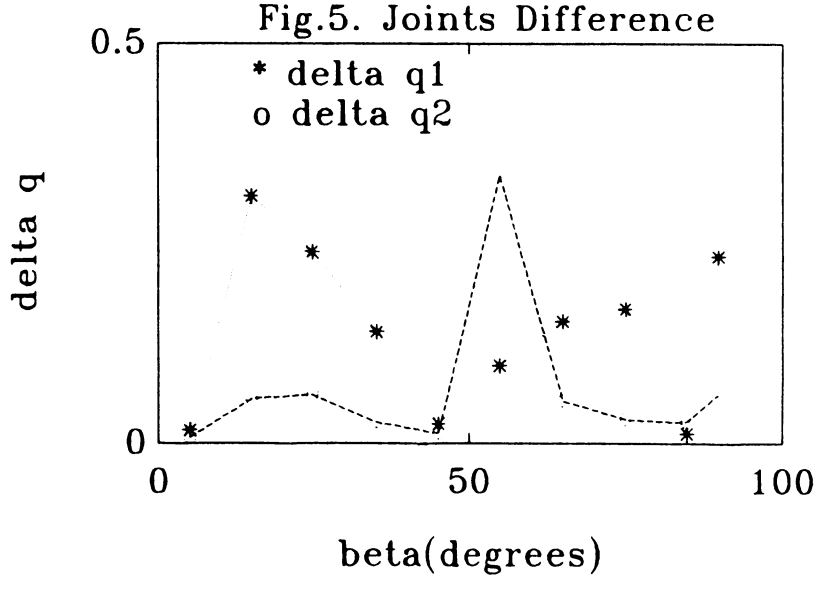


Fig.5. Joints Difference
delta q
beta(degrees)
* delta q1
o delta q2
0.5
0
50
100

NEUROMORPHIC COMPUTING ARCHITECTURE FOR ADAPTIVE CONTROL

Izhak Bar-Kana and Allon Guez
ECE Dept., Drexel University, Philadelphia, PA 19104

ABSTRACT

A neuromorphic unsupervised parallel distributed adaptive controller for nonlinear systems is proposed. It is shown to provide bounded tracking and asymptotic regulation following an arbitrary 'teacher'. A two degrees of freedom robotic simulation example is provided.

1. PROBLEM FORMULATION

An unsupervised distributed parallel computing architecture is proposed for the adaptive control of nonlinear dynamic systems of the class

$$\dot{x}(t) = A(x)x(t) + B(x)u(t); \quad y(t) = C(x)x(t) + D(x)u(t) \tag{1}$$

The proposed controller is depicted in Figure 1 below: It consists of a teacher model which contains apriori knowledge regarding the desired input/output plant response as well as the repertoir of reference commands that the system may be subjected to.

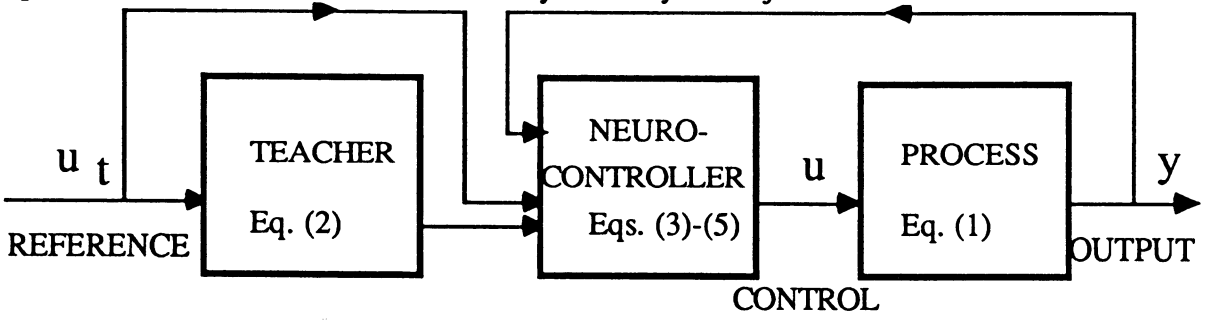


Fig. 1: Proposed Neurocontroller

The teacher dynamic model is assumed to have the following representation:

$$\dot{x}_t(t) = A_t(x_t)x_t(t) + B_t(x_t)u_t(t); \qquad y_t(t) = C_t(x_t)x_t(t) + D_t(x_t)u_t(t) \tag{2}$$

It is emphasized that the dimension of the model is unrestricted, except that $\dim(y_t) = \dim(y)$.The parallel distributed adaptive controller has structure similar to the LMS adaptive layer [Widrow and Stearns, 1985], and to many other neurocontroller architectures [Guez, 1988]. It receives the input 'features' vector $f(u_t, x_t, y)$ and generates as an output the process contorl vector u, where $u(t) = K(t)f(u_t, x_t, y)$ and where K(t) is the adaptive gain matrix of appropriate dimension. Each K_{ij} gain monitors the sensitivity of the i-th control loop, namely u_i, to the j-th feature of the system, namely $f_j(u_t, x_t, y)$. The K_{ij} gain adjusts its value independently of, and simultaneosly with all other gains, according to:

$$K_{ij}(t) = M_{ij}(t) + N_{ij}(t) \tag{3}$$

$$M_{ij}(t) = \alpha_{ij}(y_{t_i} - y_i)f_j \tag{4}$$

$$\frac{d}{dt}N_{ij}(t) = -\beta_{ij}N_{ij}(t) + \gamma_{ij}(y_{t_i} - y_i)f_j \tag{5}$$

where α_{ij}, β_{ij} and γ_{ij} are positive constants. We emphasize that the adaptation law (3)-(5) is similar in structure to the Widrow-Hoff rule [Widrow and Stearns, 1985] modified with a momentum term. As we show later in the paper, the modified structure also includes a supplementary term which takes care of the stabilization of unstable system. It is performed in parallel and in a **distributed** fashion, that is, the **K_{ij} gain only needs data from the j-th feature and i-th output components**. Thus, very large scale dynamic systems may be considered for this controller. The adaptive gains consist of two terms: a "proportional" term, $M_{ij}(t)$, and an "integral" term, $N_{ij}(t)$. Notice that the role of the teacher (2) is to demonstrate to the adaptive controller what

should be the appropriate and desired response y_t for any specified reference input u_t. Since the teacher's model is incorporated in the controller structure (Figure 1), it yields the so-called 'unsupervised' learning or adaptation.

For the above described control architecture we use recent results by [Bar-Kana, 1989] to show that, under some realistic assumptions, large classes of nonlinear plants of the form (1) with adaptive controllers of the form (3)-(5) can perform good trajectory tracking and also guarantee robust adaptive stabilization in the presence of *any* bounded input commands and input or output disturbances. Furthermore, this controller shows good graceful degradation (or fault resistant) properties [Morse and Ossman, 1989].

MAIN RESULTS

We describe below a summary of our main results regarding the performance of the controller in (3), (4), and (5). We will assume that if the plant is stabilizable the resulting stable configuration is **"exponentially stable,"** namely that all solutions x(t) satisfy the relation $|x(t)| \leq \alpha\, |x(0)|\, e^{-\beta t}$ [Hahn, 1967].

THEOREM 1 [Hahn, 1967]: Let the right hand of the equation $\dot{x}(t) = f(x,t)$ have bounded continuous first order partial derivatives. Let the equilibrium be **exponentially stable**. Then there exists a Lyapunov function V(x,t) which satisfies estimates of the form

$$1.1)\alpha_1 |x(t)|^2 \leq V(x,t) \leq \alpha_2 |x(t)|^2;\ 1.2)\dot{V}(x,t) \leq -\alpha_3 |x(t)|^2;\ 1.3)\left|\frac{\partial V(x,t)}{\partial x_i}\right| \leq \alpha_4 |x(t)|^2,\ i=1,2,....,n$$

Because we restrict our discussion to nonlinear systems linear in control of the form (1), we assume that exponential stability of the *autonomous* system (1), with $u(t) \equiv 0$, implies:

ASSUMPTION 1: Exponential stability of the *autonomous* system (1) ,with $u(t) \equiv 0$, implies existence of Lyapunov functions of the form $V(x) = x^T(t)P(x)x(t)$ and derivative of the form $\dot{V}(x) = -x^T(t)Q(x)x(t)$, where P(x) and Q(x) are positive definite for all $x \in R^n$.

After establishing the basic definitions and facts, we can start presenting the properties of the adaptive controller. The following result shows the stabilizing properties of the main term of the adaptive controller that we propose:

RESULT 1: Let us assume that the plant

$$\dot{x}(t) = A(x)x(t) + B(x)u(t);\quad y(t) = C(x)x(t) \tag{6}$$

can be stabilized by some constant output feedback, K_y. In other words the fictitious closed-loop system is **exponentially stable** according to assumption 1. Let us define $y_a(t) \triangleq y(t) + K_y^{-1}u(t)$, and use the adaptive algorithm $u(t) = -K(t)y_a(t)$ with the integral adaptive gain K(t) = N(t) given by $\frac{d}{dt}N_{ij} = \gamma_{ij}y_{a_i}y_{a_j}$.This simple algorithm then guarantees boundedness of all values involved in the adaptation process, and asymptotically perfect regulation for the augmented system (Fig. 2)

$$\dot{x}(t) = A(x)x(t) + B(x)u(t)\quad y_a(t) = y(t) + K_y^{-1}u(t) = C(x)x(t) + K_y^{-1}u(t) \tag{7}$$

The gains reach some constant values which allow perfect regulation[Guez and Bar-Kana, 1989].

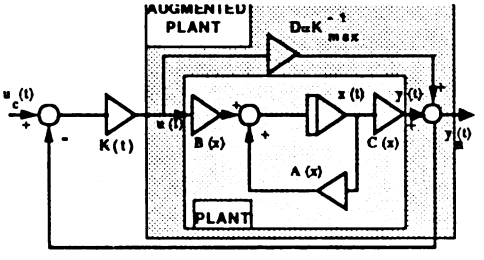


Fig. 2. The closed-loop control system.

We want to use this stability result in order to implement a stable trajectory-following adaptive algorithm. Let the teacher generate the desired trajectories that the plant must follow. Let $f(u_t, x_t, y) = [\ (y_t - y)^T,\ x_t^T,\ u_t^T\ ]^T$ where the feature vector $f(u_t, x_t, y)$ uses all values that can be measured, like the input commands, the teacher's states, and the tracking errors. We are aware that the perfect integrator used for perfect regulation in result 1 increases without bound whenever perfect regulation is not possible. Since the nonlinear system includes uncertainties that we do not assume to know or identify, and since moreover, input and output disturbances may usually be present, we do not try to prove perfect tracking, which could be obtained in some ideal situation, but concentrate on the proof of robust stability under nonideal conditions. By *"robust stability"* we mean boundness of all values involved in the adaptation process like states, adaptive gains and errors, and tracking with arbitrarily small tracking errors. Perfect tracking in idealistic conditions is a particular case of the general robust tracking under nonideal conditions. The term $-\beta_{ij}N_{ij}(t)$ in (8) and the proportional adaptive gain (7) lead to the following result:

RESULT 2 : Under the assumptions of result 1, the adaptive algorithm (3)-(5) guarantees *robust stability* of the system (7) [Guez and Bar-Kana, 1989].

Let us assume that the plant needs some general dynamic configuration to reach stability. Specifically, let H: $\{A_f, B_f,, C_f,, D_f,\}$ be some LTI dynamic feedback controller that guarantees that the closed loop system is exponentially stable according to assumption 1. Then we can state the following result:

RESULT 3 : Let $G(\cdot)$ be some nonlinear system of the form (6) and let H: $\{A_f, B_f,, C_f,, D_f,\}$ be a stabilizing controller under assumption 1. Then, the adaptive algorithm (3)-(5) guarantees *robust stability* of the augmented system $G_a(\cdot) = G(\cdot) + H^{-1}$ [Guez and Bar-Kana, 1989].

More important, when *improper* linear controllers H are needed to stabilize the nonlinear plant, we can use their *proper inverse* in parallel with the plant. This way, we only use the knowledge on the *existence* of an improper controller and actually use a proper configuration in parallel with our plant. Specifically, as in the case of nonlinear robotic manipulators, PD controllers of the form $H(s) = K(1+qs)$ can stabilize the manipulators, and if K is very large, we can get very good tracking in ideal situation. However, in practice we do not want to use differentiators or high gains in control loops. In our approach we only use the knowledge on their mere *existence* to implement simple first order poles of the form $H^{-1}(s)=\frac{D}{1+qs}$ in parallel with the plant, where $D= K^{-1}$ can be a very small gain. Notice that we do not guarantee any more that the plant is perfectly tracking, because the best we can obtain is $y_a(t) = y(t) + Du(t) \rightarrow y_t(t)$, although we actually want $y(t) \rightarrow y_t(t)$. Still, if the maximal admissible gain K_{max} is large compared with the gain of the plant, then $y_a(t) = y(t) + Du(t) \approx y(t)$.

ROBOTIC EXAMPLE

The proposed controller (5)-(8) was applied to the nonlinear robotic system[Desa and Roth,1985]: $\dot{x}_1=x_2$; $\dot{x}_2=D^{-1}[u_1+x_2x_4(B_1-B_3)\sin(2x_3)]+d_{i_1}(t)$; $\dot{x}_3=x_4$; $\dot{x}_4=[u_2-(x_2^2/2)(B_1-B_3)\sin(2x_3)]/B_2+d_{i_2}(t)$ with $D = A_1+ m_BL_1^2+B_1\cos^2x_3+B_3\sin^2x_3$, $L_1 = .1\ m^3$, $A_1 = .01\ kgm^2$, $m_B = .6\ kg$, $B_1 =.06\ kgm^2$, $B_2 =.01\ kgm^2$, $B_3 =.05\ kgm^2$. The output measurements are $y_1 = x_1 + d_{O_1}(t)$; $y_2 = x_2 + d_{O_2}(t)$, where d_i and d_O are the input and output disturbances. The teacher was given by the simple decoupled model $\dot{x}_{t_1}(t) = -25x_{t_1}(t) + u_{t_1}(t)$; $\dot{x}_{t_2}(t) = -25x_{t_2}(t) + u_{t_2}(t)$; $y_{t_1}(t) = x_{t_1}(t)$; $y_{t_2}(t) = x_{t_2}(t)$ and it was tested with demanding square-wave input commands.

In paralel with the plant (21)-(24) we employ the supplementary feedforward [Bar-Kana, 1989] $y_{s_1}(s) = 0.01u_{s_1}(s)/(1+s/10)$; $y_{s_2}(s) = 0.01u_{s_2}(s)/(1+s/10)$. Supplementary disturbances

$d_{i_1}(t)=5$; $d_{i_2}(t) = 10 \sin(100t)$; $d_{O_1}(t) = 0.1$; $d_{O_2}(t) = 0.2 \sin(120t)$ were added to check robustness, and the adaptation coefficients were $\alpha_{ij} = \gamma_{ij} = 100.$; $\beta_{ij} = 0.1$

Results of simulations are shown in Figure 3. Figure 3a compares the first plant and model output, and figure 3b shows the second output. Figure 3c shows, for ilustration, the behavior of the adaptive gain $K_{22}(t)$. It can be seen how the adaptive gain moves up-and-down in order to maintain small tracking errors. Figure 3d represents the norm of all plant states, to show that no hidden state diverges. The input disturbances were introduced from the start and their effect is hardly felt at the output. The output disturbances were then introduced at $t_1 = 0.16$sec and $t_2 = 0.23$ s. All values remained bounded while the plant tracked with small errors, although we used only position (no velocity and no acceleration) output measurements.

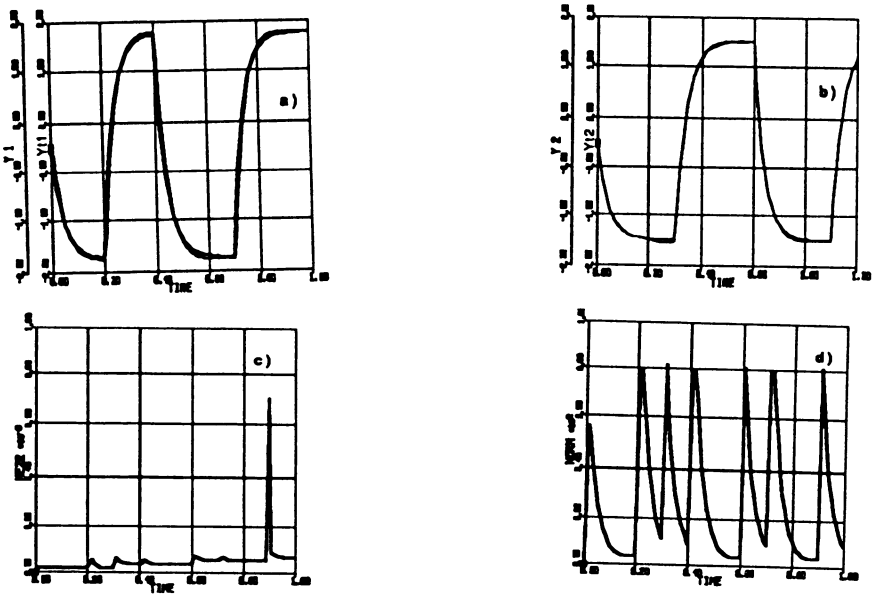


Figure 3. **Example:** (a) Output $y_{t_1}(t)$, $y_1(t)$; (b) Output $y_{t_2}(t)$, $y_2(t)$; (c) Gain $K_{22}(t)$; (d) Norm of all plant states.

CONCLUSIONS

This paper presents a neuromorphic computing architecture for adaptive control. Starting with some prior assumptions about stabilizability of the plants it results in a stable unsupervized architecture. The feasibility of the method is demonstrated on an example of robotic manipulator.

REFERENCES

Bar-Kana, I. (1989) *On Passivity of a Class of Nonlinear Systems,* Technical Report, Drexel University.

Guez, A., and Bar-Kana, I.(1989) *Unsupervised Parallel Distributed Computing Architecture For Adaptive Control,* (full report) Technical Report, Drexel University.

Desa, S., and Roth, B., (1985) "Synthesis of Control Systems for Manipulators Using Multivariable Robust Servomech. Theory," *International J. of Robotics,* Vol. 4, pp. 18-34.

Guez, A. (1988) "Neurocontrollers," *NSF Workshop on Neurorobotics,* New Hampshire, 1988.

Hahn, W. (1967) *Stability of Motion,* Springer Verlag, New York.

LaSalle, J. (1981) "Stability of Noautonomous Systems," *Nonlinear Analysis Theory, Methods and Applications,* Vol. 1, No. 1, pp. 83-91.

Morse, W., and Ossman, K. (1989) "Flight Control Reconfiguration Using Model Reference Adaptive Control," *Proceedings of 1989 American Control Conference,* Pittsburgh, Pennsylvania, pp. 159-164.

Widrow B. and Stearn S. (1985) *Adaptive Signal Processing,* Prentice Hall.

DETECTING SYMMETRY WITH A HOPFIELD NET

Eric I. Chang and David Tong

General Electric Corporate Research and Development
K1-5B28
Schenectady, NY12345
changei@crd.ge.com

ABSTRACT

Neural networks have been applied to many types of machine vision problems. This paper presents a Hopfield net, a previously little used type of neural network in machine vision, that detects vertical symmetry.* Hopfield net has been applied in many optimization problems, including The Travelling Salesman Problem [4], VLSI cell placement [2], and electric power load distribution [5]. We have implemented a Hopfield net which successfully detects mirror symmetry along a vertical axis. The issues of scaling, noise, and choice of parameters are also addressed.

INTRODUCTION

Most man-made objects exhibit symmetry, however, symmetry can be observed in several distinct levels. In other words, the definition of symmetry is often unclear. Therefore, it is often difficult to make a machine recognize symmetry. In this paper, images consisting of sets of segments are used and an image is considered to be symmetrical if all segments are symmetrical about the Y axis. A Hopfield net is utilized to recognize possible symmetry in the input segments.

The structure of the problem is as follows: given X, Y coordinates and lengths of an even number of segments, do these segments exhibit mirror symmetry about the Y axis? (Fig. 1) The difficult part of the problem is in pairing up possible pairs of segments. Since the order of the segments are random, any given segment may be symmetrical to any other segment. The Hopfield net solves this pairing problem by choosing a set of pairs of segments which are most likely to be symmetrical to each other about the Y axis.

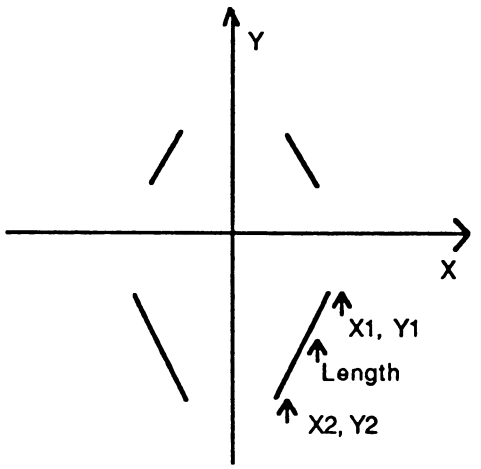


Fig. 1 A typical net input

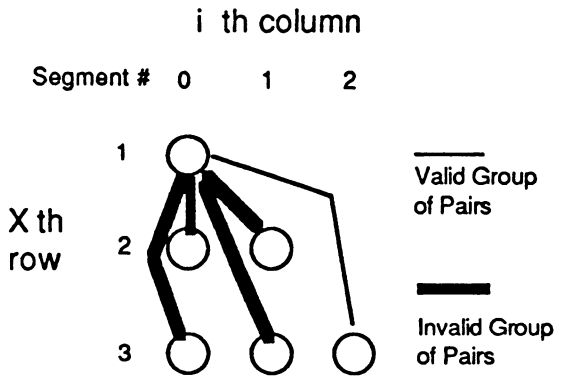


Fig. 2 Network architecture

* We wish to thank Prof. Tomaso Poggio of M.I.T. for first suggesting the symmetry problem.

NET ARCHITECTURE

The pairing of segments is performed by treating the problem as an optimization problem. One can define a function S,

$$S(seg_A, seg_B) = (X1_A+X1_B+X2_A+X2_B+Y1_A-Y1_B+Y2_A-Y2_B)^2 + (Length_A-Length_B)^2,$$

which is a measure of how symmetrical the two segments are to each other. The output of the function is inversely proportional to the degree of symmetry exhibited by the two segments. So to pair up all segments in the input vector, one would minimize the sum of the symmetry function of the pairings that the net generates.

The architecture of the net consists of a matrix of nodes, each node representing one possible pairing of segments. Each segment is identified by its position in the input vector. For example, in Fig.2, the upper left node would represent the pair of Seg. 0 and Seg. 1. In a network that pairs up N number of segments, N/2 processing elements would eventually be on, identifying the most symmetrical pairs.

Each node receives weighted output of all other nodes, the old input, a bias, and a small noise ε:

$$I_i = \sum_j T_{ij}O_j + I_{i\text{-}old} + Bias + \varepsilon \tag{1}$$

The old input is used to simulate a node with long RC constant. Wilson and Pawley have found that if one uses a RC constant of 1, oscillation between small and large values occurs.[6] We avoid this problem by using the old input, with the result of the net converging seven times more quickly. The bias value is used in conjunction with the B term in the energy equation (see below) to favor N/2 number of nodes becoming on. The noise is added to help the net escape local minima. The node sums up all inputs and generates the output through the sigmoid function.

$$O_i = \frac{1}{2}(\tanh(\frac{I_i}{I_0}) + 1.002) \tag{2}$$

1.002 is used instead of 1 used by Hopfield and Tank because if the node converged to value 0 and 1, then the node with output 0 can no longer inhibit other nodes. However, by having the node converge to 0.001 and 1.001, the nodes with low output can still inhibit other nodes. Also note that I_0 determines the slope of the sigmoid function. I_0 would be adjusted as the net converges to speed up convergence.

The energy function that is minimized by the net is presented below:

$$\begin{aligned} E = {} & \frac{A}{2}\sum_X^{N-1}\sum_i^{X-1} \sum_{\substack{X=j \text{ or } i=Y \\ \text{or } X=Y \text{or } i=j}} V_{Xi}V_{Yj} - \frac{A}{2}\sum_X^{N-1}\sum_i^{X-1} \sum_{X=Y \text{and } i=j} V_{Xi}V_{Yj} \\ & + \frac{B}{2}\left(\frac{N}{2} - \sum_{X=1}^{N-1}\sum_{i=0}^{X-1} V_{Xi}\right)^2 \\ & + \frac{C}{2}\sum_{X=1}^{N-1}\sum_{i=0}^{X-1} \sum_{\substack{X\neq j \text{ and } i\neq Y \\ \text{and } X\neq Y \text{and } i\neq j}} (S_{Xi} + S_{Yj})\, V_{Xi}V_{Yj} \end{aligned} \tag{3}$$

The first two terms of the equation represent the constraint that given a node *Xi*, all other nodes *Yj* that pairs segments that are already paired by *Xi* are inhibited by the factor A. This constraint prevents the invalid solution of having a segment paired to more than one other segment. The third term represents the constraint that N/2 nodes should be on when the net converges. This constraint prevents all the nodes being off. The fourth term constrains the net to converge to a group of nodes that exhibit the most symmetry.

The weight matrix that would minimize the above energy equation is:

$$T_{Xi,Yj} = -A(1-(1-\delta_{XY})(1-\delta_{Xj})(1-\delta_{Yi})(1-\delta_{ij})) + A\delta_{XY}\delta_{ij} - B - C(1-\delta_{XY})(1-\delta_{Xj})(1-\delta_{Yi})(1-\delta_{ij}) \quad (4)$$

where δ_{ij} denotes Kronecker delta function. Both E and T are modified versions of the original function described by Hopfield and Tank.[4]

The first two terms restrict the net from having outputs that pair one segment to more than one other segment. The third term encourages the net to have a total of N nodes on when the net converges. The last term is a function describing how closely the pairs of segments represented by the node are symmetrical to each other. By varying variable A, B, and C, different level of performance can be achieved. Much work has been done that deal with setting these parameters.[3] In this experiment, the parameters were generated through trial and error. Our experience in setting these parameters will be discussed in the next section.

Akiyama et. al. have shown that random noise and simulated annealing improve the convergence rate and reduce the number of invalid solutions.[1] In this network, a noise ε is added to the input of each node. The noise is a random value generated linearly between -D and D. The slope of the transfer function of the nodes is also gradually increased by gradually decreasing I_0. The net starts out with very low slope transfer function, thus the net can search through more surfaces by having analog outputs from the nodes. The increasing slope then forces the output of each node to become more binary and increases the rate of convergence, which occurs when all the nodes take on one of the limiting values.

The annealing of the network allows the network to be less dependent on the initial input and seek out the global minimum while still converge with relatively short number of cycles. Both D and I_0 are adjusted with the hyperbolic function:

$$I_{0t} = \frac{I_{0\,init}}{1 + t/T_o} \qquad D_t = \frac{D_{init}}{1 + t/T_o} \quad (5)$$

where t is the number of current cycle and T_0 determines the annealing schedule.

RESULTS

The Hopfield net described above was simulated on a SUN 3/60. Random segments were generated for both the four segments problem and the ten segments problem.

For the four segments problem, the net successfully paired up segments that were symmetrical. When the segments were not symmetrical, the net would still try to generate pairs that are almost symmetrical. Depending on how symmetrical the segments in the pairs are, the net may have only one or no nodes on. Thus, one can detect symmetry by counting how many of the nodes are on.

After the net successfully paired up perfectly symmetrical segments, noise was added to the inputs by first generating a perfectly symmetrical set of segments, then spinning each segment about its center by a random number of degrees within a predetermined interval. It was found that by refining the symmetry function in the energy equation, the effect of this type of noise can be eliminated. For example, when a segment is spun around its center, its endpoints' X and Y coordinates would change, but they will compensate for each other, i.e. while X1 and Y1 would increase, X2 and Y2 would decrease. Also, the length of the segment stays the same. Thus, by emphasizing the length and by summing all X Y coordinate differences together before squaring, as opposed to using the sum of the absolute value of the differences, the noise introduced by spinning the segments can be ignored.

In other words, one can tailor the energy equation to one's definition of symmetry. If equal length is more important than orientation in determining symmetry, one can emphasize the length component in the energy equation.

In the ten segments problem, a lot more possible combinations of pairings can occur. The net parameters used for the ten segments case are: A=100000, B=400000, C=5, I_0=2000000, T_0=2. The nets usually converges within 40 cycles, which takes less than 20 seconds to simulate on a SUN 3/60. The net successfully paired up all segments in symmetrical cases. For asymmetrical cases, the net pairs up all segments that it considers are closely symmetrical. Thus, sometimes the net would have two nodes in the same column on because it considers the particular segment to be symmetrical to two other segments. This behavior is due to using a large B value. Variable B constrains the net to have five nodes on at convergence, so the net picks the five best pairs it can find, even if some segments end up being paired to more than one segment. One can prevent this behavior by increasing variable A, thus emphasizing having valid solutions over having 5 nodes on at convergence.

Our experience in scaling up the problem size concurs with Hegde's[3]: the larger the problem size, the more critical the parameters become. We were able to choose a set of parameters for the four segments problem easily. However, the parameters for the ten segments problem were chosen by observing the net's output at every cycle and adjusting the parameter values to avoid oscillation and local minima. The parameters A, B, and C determine the solution that the net converges to. As mentioned previously, increasing B has the effect of favoring final states with N/2 nodes on. If we prefer nodes to be on only when pairs that they denote are absolutely symmetrical, we can increase A. Then only segment pairs that are absolutely symmetrical would be represented by a node that is on.

We did not have many invalid solutions due to the nature of our problem. When two segments are asymmetrical, the weights between the node representing this pair of segments and other nodes are large, on the order of 10^6. Thus, it is unlikely for this node to be on because it is inhibited greatly by other nodes. The large sensitivity of the weights to the degree of symmetry of the segments makes this problem less prone to invalid solutions.

The value I_0 and T_0 affect how quickly the net converges. When the problem size is increased, I_0 has to be increased as well to ensure unsaturated output from nodes because each node receives input from more nodes. If I_0 were not increased, all nodes would saturate right from the beginning, and the net will have no way of distinguishing between nodes that are less inhibited from others. The result would be that the net would converge to a false minima where all nodes are off. Too rapidly annealing the net by having a small T_0 can also lead to this problem. We usually start with large I_0 and T_0 to ensure correct solution, then try smaller values to improve the rate of convergence.

CONCLUSION AND FUTURE WORK

A Hopfield net has been implemented which can rapidly pair up symmetrical segments. The output of the net can be passed on to other nets for further processing. The parallelism exhibited by the net enables parallel implementation in the future which would speed up execution greatly. Rotation and translation invariance will be implemented in the future to allow broader applications in machine vision.

REFERENCES

[1] Y. Akiyama, A. Yamashita, M. Kajiura, and H. Aiso: "Combinatorial Optimization with Gaussian Machines", Proc. IJCNN89, vol. 1, pp.533-540, 1989

[2] D. Caviglia, G. Bisio, and F. Curatelli: "Neural Algorithms for Cell Placement in VLSI Design", Proc. IJCNN89, vol. 1, pp.573-580, 1989

[3] S. Hegde, J. Sweet, and W. Levy: "Determination of Parameters in a Hopfield/Tank Computational Network", Proc. International Conf. on Neural Networks II, pp. 291-298, 1988.

[4] J. Hopfield and D. Tank: "'Neural' Computation of Decisions in Optimization Problems", Biol. Cybern. 52, pp.141-152, 1985

[5] S. Matsuda and Y. Akimoto: "The Representation of Large Numbers in Neural Networks and Its Application to Economical Load Dispatching of Electrical Power", Proc. IJCNN89, vol. 1, pp.587-592, 1989

[6] G. Wilson and G. Pawley: "On the Stability of the Travelling Salesman Problem Algorithm of Hopfield and Tank", Biol. Cybern 58, pp 63-70, 1988.

Design of Edge Detection Templates Using A Neural Network

Scott C. Douglas and Teresa H.-Y. Meng
Information Systems Laboratory
Stanford University
Stanford, California 94305
(415) 723-9510
July 30, 1989

Abstract

We offer a method for designing matched filter edge detectors in which a neural network is trained as a local template operator to detect the location of edges in digital images. The resulting templates are shown to detect edges of given orientations and reject edges of other orientations better than other suggested template detectors. A Markov chain model for edge arrivals is presented.

Introduction

In digital image analysis, proper identification of the boundaries between homogeneous regions can ease computation in subsequent analysis. Edge detection, the identification of discontinuities in the image, represents one methodology for image segmentation which has achieved success in a wide range of applications[1]. Numerous methods for edge detection exist; most consist of a filtering step, in which a set of two-dimensional templates or masks are convolved with the image to produce edge-enhanced image data, and a classification step, in which classification of the resulting edge data produces a binary image with edges identified. Research in two-dimensional edge detection has led to solutions which include template matching[2], extensions of one-dimensional optimal filtering[3], biological analogy[4], and least-squares surface-fitting[5].

Choice of template coefficients for the initial filtering step is crucial for the success of subsequent edge labeling decisions. Since edge detection is a local operation, coefficients based upon 3x3 template sizes have been offered and include the Prewitt[6], Sobel[7], Compass[6], and Kirsch[8] operators. Templates for identifying the eight edge orientations at $\Theta = \{-135°, -90°, -45°, 0°, 45°, 90°, 135°, 180°\}$ for each of the four operators are shown in Figure 1. Each of the templates in any eight member set constitutes a filter which matches the sloping surface for edges of the particular given orientation. Their operation loosely approximates a derivative by a finite difference operation in the direction of an edge and reduces noise in the derivative estimate by averaging data values along the edge.

While the template operators given above produce peaks at edges, they also respond significantly to edges with orientations which are +/-45 degrees from the ideal edge orientation. This effect is undesirable from the standpoint of identifying edge orientation, in which we desire peak output from only one template filter for a given edge orientation.

We propose a method of two-dimensional filter design consisting of a nine-weight network followed by a three-level sigmoidal nonlinearity. We train the two-dimensional filter using the least-mean-squares (LMS) algorithm[9] to detect a particular edge orientation using a finite state Markov chain which models edge arrivals to the filter. The resulting filters are shown to reject edges at orientations other than the desired edge orientation better than the filters mentioned previously. Moreover, this scheme may be applied to real image data to produce templates which are best matched to the aberrations and edge profiles for the particular class of images being

analyzed. We now describe the filter structure and adaptation scheme.

Network Structure

Figure 2 shows the structure of the proposed filter, which is a single-layer feedforward neural network with a 3x3 array of weights **W**, which filters scanned image data **X** to produce output y. The output passes through a memoryless nonlinearity and is then subtracted from a synthesized desired input d to produce an error signal e, which is then used to adjust the weight coefficients **W** according to the LMS algorithm,

$$\mathbf{W}_{k+1} = \mathbf{W}_k + 2\mu e_k \mathbf{X}_k.$$

The nonlinearity consists of a sum of two hyperbolic tangent (sigmoid) functions which have been coordinate-shifted and amplitude-scaled to produce the family of composite curves shown in Figure 3. The resulting function resembles a three-level quantizer but is smoothly differentiable at all points and maintains the relationship f(y) = y at the desired decision outputs (y = {-1, 0, 1}) and at the midpoints between desired output values (y = {-0.5, 0.5}). This three-level form of nonlinearity is desirable in edge detection for it enables reduction in the number of template masks used from eight to four, as an edge which has been rotated 180° can be sensed as a maximum negative output from a properly-oriented template, assuming odd symmetry along the length of the edge. Equivalently, eight templates could be trained, one for each of the eight edge orientations, using a two-level sigmoid nonlinearity centered at y=0.5.

Training the Edge Detector Network

A Markov chain model for two-dimensional edge arrivals has been developed for training the network and is shown in Figure 4. The parameter λ determines rate of edge arrival for the thirty-four-state chain, which for low values of λ remains in the two no-edge states shown at left and right. Flow through the state chain proceeds clockwise around the diagram, with the probability of following any one edge path from one no-edge state to the other being an equally probable $\lambda/4$ for each of the directed paths. This Markov chain simulates the scanning of an edge detector across an image and therefore for suitably chosen values of λ can be tailored to model any rate of edge arrivals to the edge detector.

A desired response for the filter is synthesized according to the particular edge orientation to be detected, in which case (d=1) is the desired response to a centered positive-going edge of the correct orientation, (d=-1) is the desired response to a centered negative-going edge of the correct orientation, and (d=0) is the desired response for all other input. Thus, we train the edge detector to give maximum output only at the center of correctly-oriented edges and to suppress output at adjacent points in the edge scan as well as at locations of differently-oriented edges.

Results and Conclusion

The above model was used to train two edge detector neural net templates, one to recognize vertical (0°) edges, and one to recognize diagonal (-45°) edges, with λ=0.1, μ=0.01, and α=20 for 5000 steps of the Markov state chain. After approximately 500 steps, the system attained a low average error, and the resulting templates were then tested on all states in the chain to see success in producing peaked output at an edge with the desired orientation, and reduced output to either side of the peak and to edges with other orientations. Figures 5 and 6 show the absolute value output of the four fixed coefficient matched filter detectors from Figure 1, the linear output of the neural net detector y_k, and the nonlinear output of the neural net detector $\hat{y}_k$, on a set of 21 templates which represent either a positive or a negative edge in the state model. All filter output has been

normalized by the maximum output for a detected edge for comparison.

As can be seen, the neural net edge detector produces a linear filter which rejects edges of other orientations better than the aforementioned fixed template filters. For a vertical edge detector, peak output for a diagonal edge for the neural net edge detector linear output is (.57), whereas the best linear matched filters produce an output of (.67). Results for the diagonal edge detection case are similar. Moreover, the nonlinear output for the neural net shows peaked output for the desired edge and greatly reduced output for points adjacent to the edge and for edges of other orientations; this rejection aids in the setting of thresholds for binary decisioning on the edge detector output and can lead to a more accurate edge map which does not require edge thinning.

We present a methodology for the design of edge detector template filters which uses a single-layer neural net trained through the LMS algorithm on data generated from a finite state Markov chain model for edge arrivals. The resulting edge detector templates are shown to provide better classification of edge pixels through the rejection of edges of differing orientations and the suppression of filter response adjacent to an edge. This methodology can be extended to real image data to produce matched filters for a given class of images being analyzed.

Acknowledgement

This reasearch was supported by an NSF Fellowship and an NSF Presidential Young Investigator award.

References

[1] W.K. Pratt, Digital Image Processing, (New York: Wiley-Interscience, 1978).

[2] A. Rosenfeld and A. Kak, Digital Picture Processing, (Orlando, Florida: Academic Press, 1982), v. 2, pp. 84-101.

[3] F.J. Canny, "Finding edges and lines in images," M.I.T. Artificial Intell. Lab., Rep. AI-TR-720, June, 1983.

[4] D. Marr and E. Hildreth, "Theory of edge detection," Proc. R. Soc. Lond. B, v. 207, 1980, pp.187-217.

[5] R.M. Haralick, "Edge and Region Analysis for Digital Image Data," Computer Graphics and Image Processing, v. 12, 1980, pp. 60-73.

[6] J.M.S. Prewitt, "Object enhancement and extraction," Picture Processing and Psychopictorics, B.S. Lipkin and A. Rosenfeld, eds. (New York: Academic Press, 1970).

[7] R.O. Duda and P.E. Hart, Pattern Classification and Scene Analysis, (New York: Wiley, 1973), p. 271.

[8] R. Kirsch, "Computer determination of the constituent structure of biological images," Comput. Biomed. Res., v. 4, no. 3, 1971, pp. 315-328.

[9] B. Widrow and S.D. Stearns, Adaptive Signal Processing (New Jersey: Prentice-Hall, 1985), pp. 99-116.

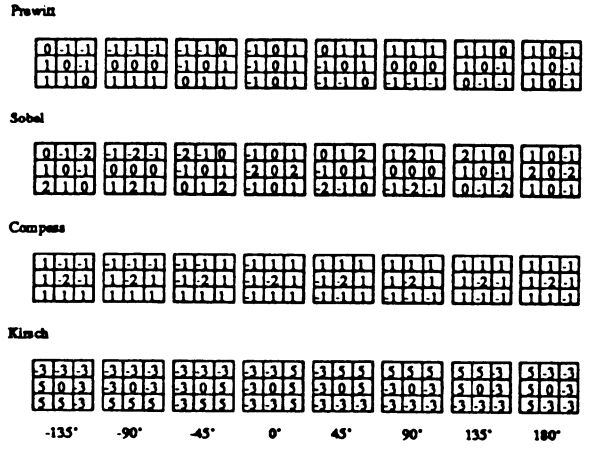


Figure 1: 3x3 Edge Detection Templates

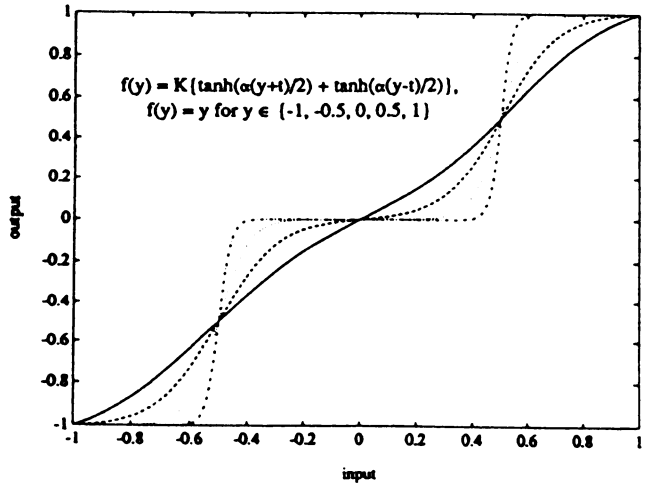


Figure 3: Three-Level Sigmoid Nonlinearity, alpha = 5, 10, 20, 50

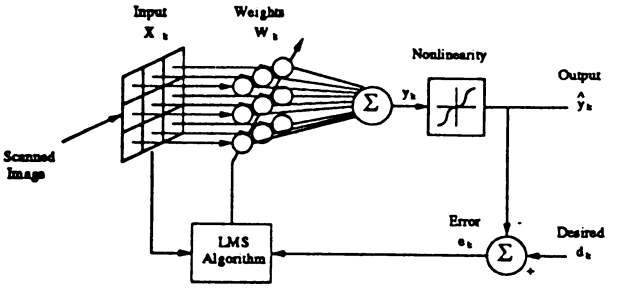


Figure 2: Neural Net Edge Detector Structure

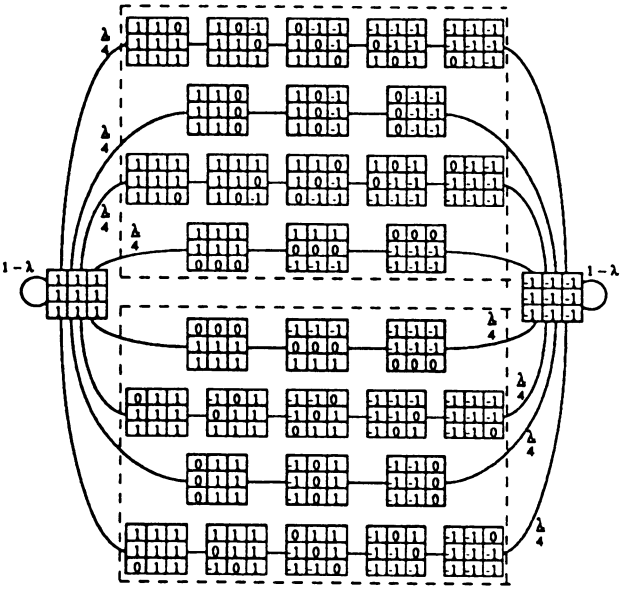


Figure 4: Markov Chain Model for Edge Arrivals

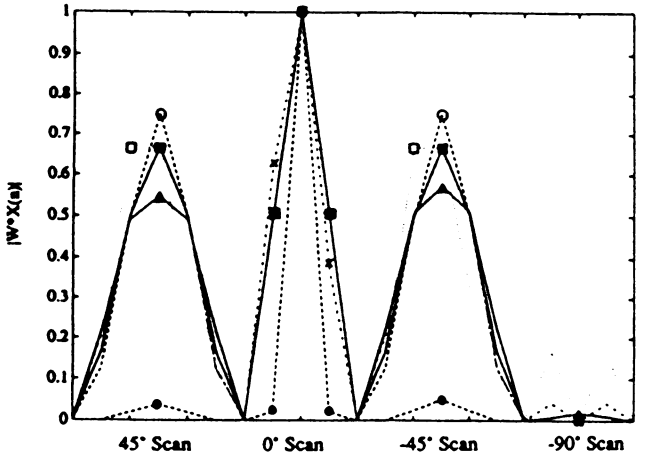


Figure 5: Vertical Template Operator Output

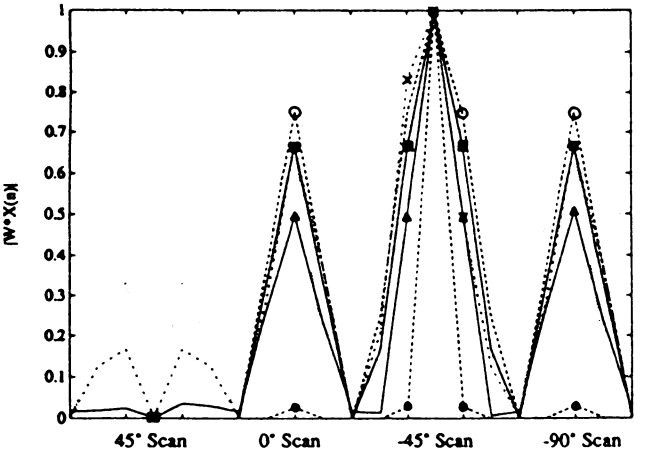


Figure 6: Diagonal Template Operator Output

Legend: Δ --Prewitt o --Sobel □ --Compass
× --Kirsch ▲ --NNet Linear ● --NNet Nonlinear

Multi-Resolutional Retina Images for Machine Vision

A.M.Fong
Neural Systems Engineering,
Department of Electrical Engineering,
Imperial College,
University of London,

Abstract:

Biological/neural processes have long been motivations for better and faster machine vision performance. Conventional machine vision normally makes use of an image that is uniformly fine in resolution throughout the scene.This contrasts with human visual perception where the retina forms images that are graded in resolution across the forvea, yielding forveal and peri/paraforveal images on saccardic movements or on instances of attentional/forveational image sampling.Instead of considering this multiresolutional mammalian-retina structure as an undersirable feature, we investigate if such a multiresolutional forvea-centred (MFC) image could have desirable properties for machine vision.We present a simple model of machine vision processing for object perception making use of some of the known anatomical/neuronal and physiological characteristics in human vision perception.This model makes use of multiresolutional forvea-centred image samples that enable the model to perform object perception using preattentive and focused-attentional phases that are characteristic of human object perception.The model also shows that the MFC image which is basically low-pass channels also provides bandpass channels on saccadic jumps, though we are not sure how this could be be put to good use. It will be seen that this model lends itself quite well to a hardware implementation using suitable optical front-end image acquisition hardware or with digital signal processing (DSP) matrix convolvers, thus affording a possibly real-time solution to some machine vision tasks.

INTRODUCTION.

Neuronal/Connectionist networks have been used for image processing tasks mainly as pattern recognition, classification, completion models with the complete image usually having been segmented (ie.object/s of interest are available to the network).These tasks can be performed using higher-level symbolic processing making use of the associative and generalisational properties of neural-like networks,for which sometimes, conventional algorithmic machine vision would findsimilar tasks difficult.[Fuku86]. However,it would be more difficult to use purely neural-like networks to process images containing multiple objects at low-levels requiring models of semantic segmentation and action- oriented processing,that is known to occur in human perception (for which algorithmic machine vision is more capable). Notwithstanding these difficulties,neural-like processing with random access memory elements have been used to engineer systems[Alek84] that can provide real-time solutions to some vision tasks,using multiple discriminators. The human visual perception must be using the range of mechanisms of neural/algorithmic processing at all levels, from the most elementary neuron level to the higher brain functions, that ultimately give human perception this supreme ability. Machine vision can be distinguished from pattern recognition,for which much connectionist systems are directed towards,by the fact that one of the aims of machine vision is scene description or discrete object labeling[Fu1982].There has been a growing awareness of the need to incorporate some of the neuronal/physiological processes to machine vision tasks (robot vision,visual inspection and object recognition) [Marr82, Bar/Ten78, Tani80, Uhr72, Han/Ris78]. This paper examines how the retinal image in its multiresolutional form can be used as the basic image acquisition model for machine vision tasks

pointing out some of the advantages and limitations of such an approach.

NEURONAL AND BIOLOGICAL MOTIVATIONS.

We are now more aware of neuronal processes in the visual pathways[Hubel/Wies62].While research continues to explain how processing is really done at neuron/synaptic level at the retina,the physical structure of the human retina is quite well known [Dowling87]. Psycho-physiological studies,[Triesman78] indicate that human object perception is characterised by two distinct phases of Global Preattentional and Local focused-attentional mechanisms occuring in most visual tasks.It is also now quite accepted that in adult vision,the spatial information is processed in parallel using a number of distinct spatial mechanisms tuned for orientations and size in spatial frequencies [Julesz81]. It is conjectured here that those mechanisms that provide humans with such characteristic manner of object perception may be related to the unique way in which humans acquire the images of the world as multiresolutional forvea-centred(MFC)samples.

RELATED MULTIRESOLUTIONAL IMAGES.

Minsky[Minsky75] advocated the need for a 'planning' process in problem solving and Kelly[Kelly71] was one of the earliest to use a mutitresolution model for 'planning' to obtain the outline of a human face.Uhr[Uhr72] used the processing cones as a model of vision processing. Tanimoto[Tani80] used pyramidal multiresolutional image structures to perform certain machine vision tasks involving searches etc.One of the characteristics of the above multiresolutional models is that the image is decomposed into distinct layers of a pyramid with each layer having a constant reduced resolution and reduced dimensions being n-logarithmic to the layer n. Most of these models do not explicitly incorporate a forveal resolution image in the centre of the resolutional images as in the MFC model although the intrinsic images of Tennembaum could conceptually incorporate a MFC images since the intrinsic images are of aimed at providing a richer description of the scene.Such pyramidal structures are then available for faster and more robust processing in hierachical coarse-fine computations.Although it is quite difficult to imagine how the internal image representation in humans could exist in the form of these distinct pyramidal image layers, such pyramidal structures do avail themselves to pyramidal/cellular machines for purposes of computation.

THE MULTIRESOLUTIONAL FORVEA-CENTRED (MFC) IMAGE REPRESENTATION.

The Human Retina:Brief background.

A simple cross-section of the vertebrate retina shows that the photoreceptors (rods and cones) are distributed in a spatially graded manner with highest concentrations at the forvea area and less of the photoreceptors further away from the forveal area.At any instance the image that is acquired is an incomplete multiresolutional image but with a small central area that is at the highest resolution.

Formulation of the MFC Image.

In this model two MFC images are used,namely the Global MFC (GMFC)and the Local MFC (LMFC) images corresponding to the images obtained in the global preattentive and the local focused-attentional phases of perception as shown in fig 1.The GMFC is obtained by generating a multiresolutional image for the whole scene (scene sizes are 256x256 grey levels of 256 intensity levels) of 7 resolutional levels. The LMFC is generated during focused-attentional image samplings and are of 3 resolutional levels of different dimensions from the GMFC images.Further the computational descriptions provided by the LMFC can be richer (as it is uses a relatively small computation window).The structure of both the GMFC and LMFC are shown in fig 1. The GMFC image is obtained from the original image Io,as a set of n annular reduced-resolution images {Ig} of resolution r_n, and area a_n.

$$GMFC = \{Ig1, Ig2 \ldots Ign\}$$

each Ig being generated by a

causal(i.e.single- line scan) of n adjacent pixels in Io,as shown in the diagram below: The LMFC image is a set of 3 resolution images representing a image area of 27 by 27 pixels but weighted according to their distances from the forvea centre.The computation results returned by the LMFC is more complete (ie.includes spectral quantities and spatial relational quantities of first and second derivatives) as in fig 3. These descriptions fall under two classes: 1.a set of values called spectral quantities (mean,variance,etc) 2.a set of values based on the spatial relationship of image pixels. (dicontinuities of edges,orientations,signal-to-noise ratio of edge elements,dominance of edge orientations,etc). These returned values are used to provide the actions required for the forveation process (e.g. where should the next forveal inspection be located).

SEMANTIC OBJECT SEGMENTATION.

Semantic object segmentation is essential in obtaining the object of interest (or area of interest) from a visually complex scene where non-semantic segmentation using global thresholding or texture segmentation may not yield desired results.It represents one of the most essential stages of machine visionor pattern recognition as it implies the mechanism of selective attention . A simplified example on the use of the MFC image representation for a machine vision task of semantic object-segmentation is described below:

a.generate the coarse-level and fine-level description of the model of the object of interest.

b.Generate the GMFC.

c.Using the coarse model and the GMFC,generate the likely candidate areas of interest.

d.Perform focus-attentional forveation on the likely area of interest.

e.Match the LMFC with the fine-level model of object of interest.

f.Iterate from b to e until fail or success.

The key to the robustness of the above procedure is that at step d,the richer descriptions of the LMFC is obtained from a weighted computation of both fine-level forveal and coaser periforveal features,giving it a better performance under noise.An example of a LMFC for edge orientations is shown in fig 3

DESIRABLE PROPERTIES OF THE MFC MODEL.

The MFC model shows the following desirable features:

a.The MFC model provides a more natural way of using global preattentive and focused-attentional object perception as is known in human perception.

b.The degraded resolutional images are more gradual in resolution -reduction than the non-causal models(window of more lines) of multiresolution reduction as in conventional pyramidal images,thus giving a more 'traceable'image.

c.The existence of the central undegraded forveal image provides a guide in the interpolational description of the MFC.

d.The LMFC being of a smaller area can be used to generate a richer,weighted description of the forveal region without high computational overheads.

LIMITATIONS OF THE MFC APPROACH.

The MFC approach for machine vision is basically based on a coarse-fine-coarse relaxational search using hierachical model descriptions of the object of interest. It depends on the robustness of the model description in performing the search and for visually complex scenes, it may be difficult to generate a good model description. For real implementation, it depends on the ready availability of the multiresolutional images that need to be obtained in real-time. Present object perception methods based on action-oriented searches are not popular as practical machine vision prefers the use of global search methods (eg.Hough Transforms) using specifically preengineered scenes.(silhouettes etc).

CONCLUSION.

The MFC model for machine vision is presented as an exploratory solution where semantic objects/areas of interest are required to be obtained from a more visu-

ally complex scene for which global methods are not amenable.It makes good use of the characteristic multiresolutional forvea-centred image that humans obtain from the retina.Such a image is seen to be more capable of handling noise in images through multiresolutional filtering,thus generating a smaller-dimensionality, but more semantically-consistent set of features for higher level processing using neural-like energy searches or algorithmic methods.The spatial dimensions of the GMFC and the LMFC have been arbitrary selected.Multiresolutional Forvea-Centred images show promise as a way to make use of the approachable part of the brain[Dowling87] for object perception in machine vision applications.

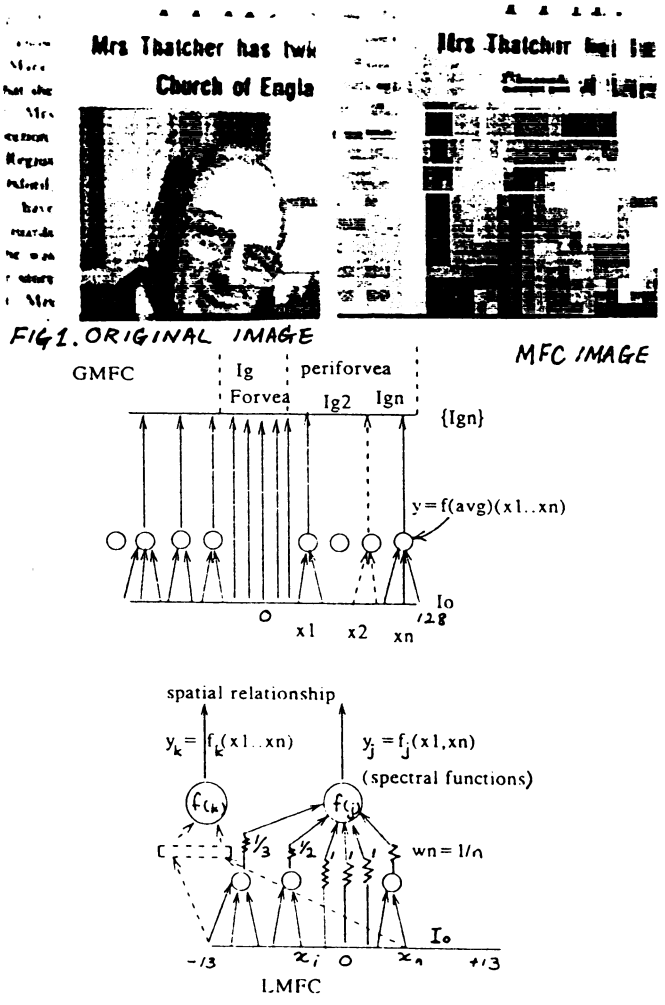


Fig 2. a. GMFC and b. LMFC formulations

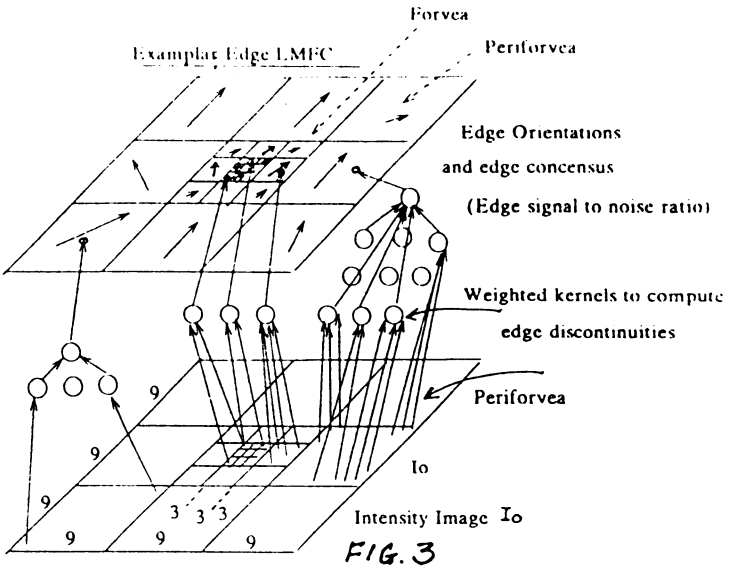


FIG. 3

REFERENCES:

J.E.Dowling,The Retina:An Approachable Part of the Brain. Cambridge:Harvard University Press,1987.

Aleksander I.,Thomas W.V.,and Bowden P.A.,"WISARD,a Radical Step Forward in Image Recognition,Sensor Review,Vol 4,no 3,pp 120-124.

K.Fukushima,"A Neural Network Model for Selective Attention in Visual Pattern Recognition",Biological Cybernetics,Vol 55, No 1,Oct 1986,pp5-15.

K.S.Fu,"Syntactic Pattern Recognition and Applications, Prentice- Hall, Englewood Cliffs, N.J.1982.

Hubel,D.H. and Wiesel,T.N."Receptive Fields:Binocular Interaction and Functional Architecture in the Cats Visual Cortex",J.Physiol vol 160,pp 106-154.

Julesz B.and Schumer R.A.(1981) "Early Vision",Annual Review of Psychology vol 32,,pp 575-627

Minsky M."A Framework for Representing Knowledge",in Psychology of Computer Vision,Winston P.H. (ed)Mc-Graw Hill,New York 1975.

S.L.Tanimoto and A.Klinger (eds), "Structured Computer Vision",Academic Press,New York 1980.

Kelly,M,D."Edge detection by Computers using Planning",in Machine Intelligence no 6,Mitchie D.(ed),1979,Edinburgh University Press pp379-409.

L.Uhr,"Layered 'Recognition Cones that Preprocess,Classify and Describe'.",IEEE Trans on Computers,1972,21,pp 758-768.

A.M.Fong,"Multiresolutional Forvea-Centred Image Representation for Robotic Object Recognition",Internal Report.

Analysis of an Inhibitive Directional Selective Unit for Vision†

David Yu-Shan Fong
Digital Image Processing Laboratory
Lockheed Missiles and Space Company
Palo Alto, CA 94304-1191

Carlos A. Pomalaza-Ráez
Department of Engineering
Purdue University
Fort Wayne, IN 46805

Abstract

Directional selective unit or velocity sensitive unit is the mechanism in a vision system which responds to motion in a specific direction. This paper analyzes a model for directional selective unit proposed by Barlow and Leveck. This model employs the inhibitory mechanism. Several parameters of the model are defined and examined. The analysis shows that the range of the response of the unit can be adjusted through the parameters, that the specific values of the parameters are not as important as the relative values of the parameters, and that a family of velocity-sensitive units with different ranges can be used to achieve velocity-specific detection.

Introduction

Directional selective unit, or sometimes called velocity sensitive unit, is the mechanism in a vision system which responds specifically to motion of a certain direction. Usually the unit has a preferred direction and a null direction. When the motion stimulus is in the preferred direction, the unit responds favorably; when the motion is in the null direction, there is no response coming out from the unit.

Barlow and Leveck [1] proposed a model for the explanation of the velocity sensitive units in rabbit's retina. The model employs the inhibitory mechanism as opposed to the excitatory mechanism. The main reason for this is because the observed behavior can be explained better by the inhibitory mechanism than the excitatory mechanism.

In this paper we explore further the Barlow-Leveck model using the approach of parameter analysis. We define a number of parameters for the model in an attempt to quantify the model. These parameters are examined with the intention of using the directional selective unit as a building block for computer vision. The main findings of the study is the conditions of forming the preferred-null directional selectivity. It also demonstrates that the range of the response can be adjusted through the parameters and that the specific values of the parameters are not as important as the relative values of the parameters.. This knowledge is needed to better understand the behavior of the inhibitive directional selective unit and its potential usage in the computer vision system.

The Directional Selective Unit

Two models of directional selective units can be devised. One is the excitatory mechanism, such as the Reichardt model [2]; the other one is the inhibitive model, to which the Barlow-Leveck model belongs. Figure 1 shows a schematic for the excitatory model, which works by the principle of excitation coincidence. When the stimulus is moving in the *preferred* direction, the response of sensor n on which the stimulus falls at the moment goes through a delay ε_h to reach neuron $n+1$ in the preferred direction. During the propagation of the response, the stimulus itself is moving to sensor $n+1$ in the preferred direction. When the

†Part of the work reported in this paper was carried out in the ECE Department, Clarkson University, Potsdam, New York.

response from sensor n reaches neuron $n+1$, the response from sensor $n+1$ may have traveled down through delay ε_s to reach neuron $n+1$ at the same time. When this happens, the total stimulus to neuron $n+1$ exceeds the threshold, and an output response is generated, indicating the movement is in the preferred direction. Since the two sensor responses will coincide only when the stimulus is traveling under a certain speed in the preferred direction, the output response is velocity-range specific.

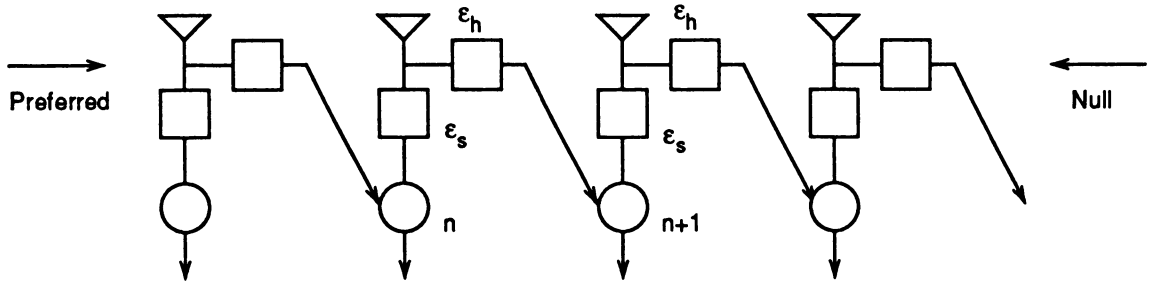


Figure 1. Excitatory mechanism

On the other hand, the inhibitory model works by the principle of inhibition avoidance. Notice that the input to the neuron from the neighboring sensor is inhibitory. This inhibitive signal creates a 'window of inhibition' during which the neuron in consideration is suppressed of any output. When the stimulus travels in the preferred direction, the response going through the delay ε_s to the neuron is not inhibited since the window of inhibition is lagging behind. This response is therefore output by the neuron. But when the stimulus is traveling in the *null* direction, the window is also traveling in the null direction. If the stimulus falls on the neuron within this window, the output of the neuron will be suppressed.

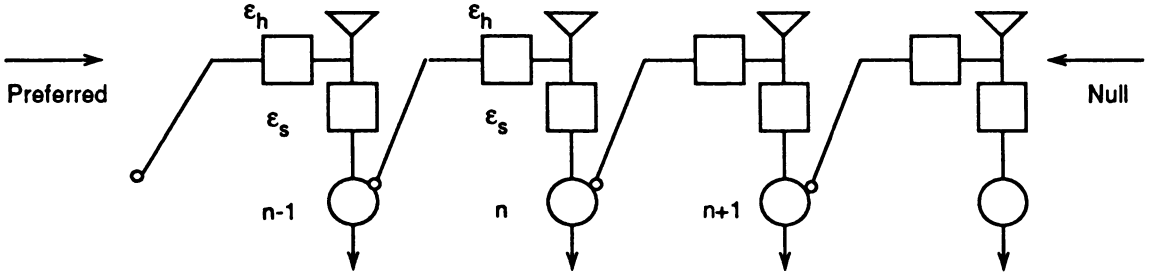


Figure 2. Inhibitory mechanism

Parametric Analysis of the Inhibitory Model

In this section we analyze in detail the behavior of the inhibitory directional selective unit through parameter analysis. First we define the parameters of the model:

d: distance between sensors

v: velocity (positive values for preferred direction, and negative values for null direction)

ε_h: time delay of inhibitory signal from sensor to the previous neuron

ε_s: time delay of response signal from sensor to its corresponding neuron

c_h: inhibition recover time, i.e. the time needed for a neuron to go back to the excitable state from inhibitory state

One dimensional space x is used for the ease of analysis. We assume a stimulus is

moving at velocity v across the space. When $v > 0$, the stimulus is moving in the preferred direction; and when $v < 0$, the stimulus is moving in the null direction. Without loss of generality, assume sensor S_n is located at $x = nd$, where n is an integer, positive or negative. The moving stimulus is at $x = 0$ when $t = 0$.

If $v > 0$, the stimulus crosses sensor S_n at $t_s = (nd)/v$. The response of the sensor reaches neuron N_n at $t_n = (nd)/v + \varepsilon_s$.

When the stimulus moves to the next sensor, the inhibition signal reaches N_n at $t_h = (n+1)d/v + \varepsilon_h$.

Now if $t_h > t_n$, the inhibition signal arrives too late, and there is no inhibition at N_n. Since the motion is in the preferred direction, this is what we want.

Rewriting the condition with model parameters, we have

$$\frac{(n+1)d}{v} + \varepsilon_h > \frac{nd}{v} + \varepsilon_s$$

or

$$\frac{d}{v} > \varepsilon_s - \varepsilon_h.$$

Here if $\varepsilon_s \leq \varepsilon_h$, then $\varepsilon_s - \varepsilon_h \leq 0$, and for any $v > 0$, the inequality is always satisfied. This means we will have the response in the preferred direction as long as the sensor delay is less than the inhibition delay.

If on the other hand $\varepsilon_s > \varepsilon_h$, then $v < d/(\varepsilon_s - \varepsilon_h)$ is the condition to elicit a response at neuron n in the preferred direction.

For $v < 0$, the stimulus traverses in the null direction. Note that with the assumption we have, $n < 0$.

In this case we have $t_s = (nd)/v$, $t_n = (nd)/v + \varepsilon_s$, and $t_h = (n+1)d/v + \varepsilon_h$. Since this is in the null direction, inhibition is needed. To inhibit output, we need

$$t_h < t_n < t_h + c_h$$

Similar analyses can be carried out for conditions like $\varepsilon_s - \varepsilon_h > 0$, $\varepsilon_s - \varepsilon_h = 0$, $\varepsilon_s - \varepsilon_h < 0$, and $\varepsilon_s - \varepsilon_h - c_h > 0$, $\varepsilon_s - \varepsilon_h - c_h = 0$, $\varepsilon_s - \varepsilon_h - c_h < 0$.

The main results

The main results of the analysis are the following:

1) When $\varepsilon_s > \varepsilon_h$, v in the preferred direction will elicit output only when $v < d/(\varepsilon_s - \varepsilon_h)$.

2) In the null direction, in addition to the condition of $\varepsilon_s > \varepsilon_h$, if $\varepsilon_s - \varepsilon_h - c_h < 0$, then

$$|v| > \frac{-d}{\varepsilon_s - \varepsilon_h - c_h}$$

will inhibit output at N_n. In other words,

$$\frac{d}{\varepsilon_h - \varepsilon_s - c_h} < v < \frac{d}{\varepsilon_s - \varepsilon_h}$$

is the range a response will be seen at neuron N_n.

Examples

Some numerical examples are shown here:

1) If $\varepsilon_s = 10$, $\varepsilon_h = 9$, $c_h = 100$, $d = 1$, then $-0.01 < v < 1$

2) If $\varepsilon_s = 1.0$, $\varepsilon_h = 0.9$, with the same c_h and d, then $-0.01 < v < 10$

3) If $\varepsilon_s = 0.1$, $\varepsilon_h = 0.09$, then $-0.01 < v < 100$

4) Now if $\varepsilon_s = 1.5$, and $\varepsilon_h = 1.4$, since the difference is still the same as case 2 above, the response is the same: $-0.01 < v < 10$.

Grouping of the Directional Selective Units

Since each directional selective unit is sensitive to a range of velocities, a family of these units can be grouped together to give information about specific velocities.

A possible scheme is to use the lateral inhibition to single out a velocity. As shown in Figure 3, a number of directional selective units are connected through lateral inhibition. The velocity range of unit 0 is the lowest (-0.01 - v_0), that of unit 1 (-0.01 - v_1) is higher than unit 0 ($v_0 < v_1$). The ranges gradually go up to unit 6 (-0.01 - v_6), which is the highest in this example. If the stimulus is traveling with a velocity between v_2 and v_3, then units 0, 1 and 2 do not respond while units 3, 4, 5 and 6 responds. But the responses of units 4, 5 and 6 are laterally inhibited (treat unit 6 as a special case), only the response of unit 3 goes through. Thus the family of units have combined to indicate that the stimulus is traveling in a velocity between v_2 and v_3.

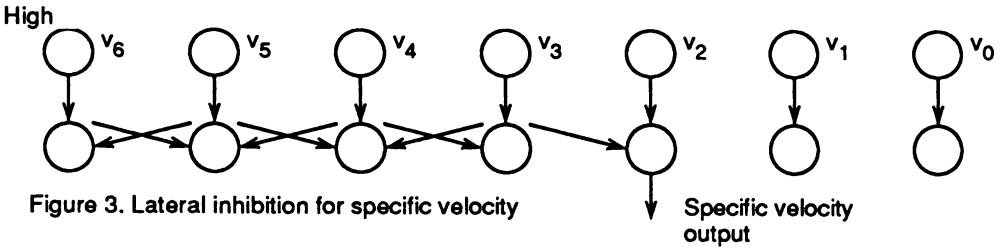


Figure 3. Lateral inhibition for specific velocity

Summary

In this paper we briefly analyzed the inhibitory directional selective units with the intention of exploring the possible usage of these units as building blocks for computer vision. We demonstrated that the range of response of the velocity selective unit can be adjusted through the parameters ε_s, ε_h, and c_h.

Parameter c_h can be used to control the null range. We want the starting point of the null range to be as close to the origin as possible. We also found that the specific values of the parameter is not as important as the relative values of the parameters. Finally we showed that a family of these directional selective units can be grouped together with lateral inhibition to generate velocity-specific responses.

References

[1] H.B. Barlow and W.R. Levick: 'The mechanism of directionally selective units in rabbit's retina,' *Journal of Physiology London*, pp. 477-504, 1965.

[2] J.P.H. van Santen and G. Sperling: 'Temporal covariance model of human motion perception, *J. Opt. Soc. Am.*, pp. 1024-1028, September/October 1983.

Competitive Activation Methods for Dynamic Control Problems[†]

Sharon M. Goodall James A. Reggia
Department of Computer Science, University of Maryland, College Park, MD 20742

Connectionist models often use competition between nodes to achieve a coherent overall behavior. Usually, these competitive interactions between nodes are implemented structurally with inhibitory (negatively–weighted) connections. In constrast, with competitive activation mechanisms, competitive interactions are implemented functionally through the competitive allocation of node output [5]. With this technique there is often a substantial reduction in the number of links required for implementation and node fanin/fanout. Unfortunately, developing a specific competitive activation mechanism for a given application can be quite challenging. Competitive activation mechanisms have previously only been developed for "static" pattern classification and non–linear optimization tasks [1, 4, 6, 7]. In such "static" tasks, an input pattern is given, the network reaches equilibrium with an output pattern, and information processing is complete.

We have recently begun to explore whether competitive activation mechanisms can be developed for "dynamic" control problems where input is an ongoing, varying function of time. In this report, we describe the first competitive activation mechanism for time–varying control problems and the results of its experimental testing. The reasonably good performance of this model suggests that competitive activation mechanisms may have substantial applicability to control problems.

The Camera Control Problem

The connectionist model developed in this research is that of a simplified "camera tracking" controller. Figure 1 gives a pictorial representation of the problem considered here. In this version, there are three tracking cameras that can move horizontally along the bottom edge of a 15 x 20 array of terrain locations, each of which can potentially contain a photographic target. The targets propagate as shown from the top edge of the array (row 1) towards the bottom edge (row 20), within one of the columns. The targets move downward at the rate of one location per time step of the simulation clock, modelling entities whose photograph is desired as they approach and pass under a spacecraft. Each camera photograph covers a one row by one column field of view in row 20 of the target grid and can move horizontally one column per time step. The objective of the control system is to maximize the number of targets photographed. The global constraints can be summarized as:

C1: Targets propagate one row per time step.
C2: Cameras move at most one column location per time step.
C3: Cameras take a picture when a target is in their current field of view.
C4: The number of pictures taken of targets is maximized.
C5: Only one camera takes a picture of a particular target.

In the implementation, three node sets are used to represent the problem: Terrain nodes, Camera nodes and Position nodes. The Terrain nodes represent the terrain location grid. There is a node for each grid location. Terrain nodes are active when there is a target at their location in the array. At each time step in the simulation, activated Terrain nodes pass their activation along to their neighbor in the next row of the same column. In this way, the Terrain nodes implement the first global constraint of the problem (C1).

The Position nodes form three rows, each row being connected to a separate camera. The fifteen Position nodes of a row represent all the possible positions that their camera could be occupying. The activation values of these nodes represent the level of demand for camera coverage in that particular column position. The Position nodes in all three rows compete for Terrain node output (described below), using a competitive activation mechanism.

[†]Supported by NASA Award NAG1–885 and by NSF Award IRI–8451430. Dr. Reggia is also with Dept. of Neurology (UMAB) and University of Maryland Institute for Advanced Computer Studies.

There are three Camera nodes, one for each camera in the control problem. At each time step in the simulation, each camera "moves" towards the most active of its Position nodes, thereby moving towards the column location with the greatest demand for camera coverage. On each time step, the cameras also check (before they move) for a target to photograph in their old position in row 20. If there is an active target in their field of view, the Camera node turns on (takes a picture), and cancels the target. Global constraints C2 and C3 are thus handled directly by the Camera node processing.

Competitive Activation in the Network

The last two global constraints C4 and C5 are addressed by the competitive activation mechanism (summarized here; details in [3]). The Position node competition for the incoming targets is intended to maximize the camera coverage of the targets, and to minimize the camera coverage overlap.

Terrain nodes use two different activation output functions. They output their entire activation value to their Terrain node neighbor as described above; the weight on this connection is always 1.0. In contrast, Terrain node output to a Position node is proportional to the Terrain node's activation level, the weight on the link to that Position node *and* the activation of the destination Position node. This activation method is *competition-based* since the Position nodes actively compete for a Terrain node's activity. A Position node's competitive strength is determined by its activation–link weight product, relative to its competitors. The three Position nodes of a particular column compete for activation output by each Terrain node in the column.

The exact output at time t from Terrain node i to Position node j is formulated as follows:

$$out_{ji}(t) = \frac{(a_j(t) + \delta)\, wt_{ji}}{\sum_k (a_k(t) + \delta)\, wt_{ki}} \frac{a_i(t)}{d^2}$$

where $a_j(t)$ is the activation level of Position node j at time t, wt_{ji} is the weight on the link from Terrain node i to Position node j, and d is the Terrain node's distance from the photographic edge (row 20) of the target array. Output from activated Terrain nodes near the photographic region is thus larger than from those farther away (due to the factor d^{-2}), since imminent targets require immediate attention. The constant δ is a very small number (10^{-15}); it is used to avoid a zero divide situation.

Implementation and Testing Scenario

The camera tracking connectionist network was implemented using the general–purpose network simulation system MIRRORS/II [2]. Each test run had a duration of 1000 time steps of the simulation clock. During a simulation, photographic targets were randomly generated at various constant rates (densities) according to a uniform distribution. Performance was measured in terms of the fraction of the total targets passing by the photographic region of the target array that were photographed (row 20).

We analyzed the performance of our implementation by comparing the system performance at various target densities to three other measures. The first measure B (blind cameras) is the small fraction of targets that would be photographed by chance by three "blind" cameras (either stationary or randomly moving). Specifically, $B(t,c) = t*c$ where $t \epsilon [0,1]$ is target density and $c \epsilon [0,1]$ is camera density, i.e., $c = \frac{n_c}{n}$ for n_c cameras and n target field columns. The other two measures represent ideals which the model could not possibly achieve, particularly at high target densities. The measure U (unrestrained cameras) estimates the expected maximum number of targets the system could photograph without any constraint on the camera movement (C2). The function U is $U(t,c) = \sum_{i=0}^{n} \frac{min(n_c,i)}{i} P(i)$. Value i is a discrete random variable with probability $P(i)$ which follows a binomial distribution $B(n,t)$. The measure F (fortuitous cameras) estimates the expected maximum number of targets the system could photograph, with constraint C2 in place but with a fortuitous distribution of random targets (e.g., targets in each row widely spaced). The function F (derived in [3]) is

$$F(t,c) = \sum_{i=0}^{n} \frac{min(n_c, f_{n_c,n} * i)}{i} P(i)$$

where i is the number of targets in a row and $f_{n_c,n} = \sum_{i=0}^{n_c-1} P(i) + \frac{3n_c}{n} \sum_{i=n_c}^{n} P(i)$.

Results

Performance results at various target densities t are shown in Figure 2 for the intact system and for two "damaged" versions (one camera removed, two cameras removed). Also shown are the B, U and F values described above ($n_c = 3, n = 15, c = 0.2$). For very low target densities—which would presumably be the operating situation—the model performed close to the F measure. As target density increased, the fraction of targets photographed decreased, but so did the F values. Even with two cameras disabled, the remaining system, having one camera, operated well above the level of performance of the three "blind" cameras (B values).

Conclusions

When fully operational, the connectionist model performed well at low target densities, suggesting that competition–based connectionist models may have potential for control applications in this range. Even at higher target densities, the system's performance can be viewed as quite reasonable. As noted above, the measures U and F substantially *exceed* the true upper bound on performance possible with any optimal system operating under constraints C1–C5, particularly at higher target densities. The U measure assumed unconstrained camera movements, while the F measure was derived based on overoptimistic assumptions (e.g., the targets being photographed at each instance are far apart) [3]. In fact, as target density approaches 0.2, more than three targets per row become frequent, while at most three targets per row could be photographed.

The robustness of the model is demonstrated by the fault tolerant performance of the system under various degraded conditions (excessive target densities; damage to one or two cameras). This property of fault tolerance is essential in the design of systems which are intended for remote operation.

Future work could investigate a number of possible improvements to the performance level. A better activation rule, or permitting network weights to be altered during learning, might produce better performance. An important extension of this work would be to observe the system performance with other target generation functions, such as interdependent targets and targets which move diagonally.

(1) Bourret, P., S. Goodall, and M. Samuelides, "Optimal Scheduling by Competitive Activation: Application to the Satellite–Antenna Scheduling Problem", *Proc. IJCNN*, I:565–572, [1989].

(2) D'Autrechy C. L., J. Reggia, G. Sutton and S. Goodall, "A General–Purpose Simulation Environment for Developing Connectionist Models", *Simulation* **51**(1):5–19 [1988].

(3) Goodall S., and J. Reggia, "Competitive Activation for Dynamic Control", Technical Report, Dept. of Comp. Science, Univ. of Md. College Park, MD, [1988].

(4) Peng, Y., "A Connectionist Solution for Vertex Cover Problems", Institute for Software, Academia Sinica, Beijing, The People's Republic of China [1988].

(5) Reggia, J. "Properties of a Competition–Based Activation Mechanism in Neuromimetic Network Models", *Proc. Intl. Conf. on Neural Networks*, II:131–138 [1987].

(6) Reggia, J., P. Marsland, and R. Berndt, "Competitive Dynamics in a Dual–Route Connectionist Model of Print–to–Sound Transformation", *Complex Systems*, (in press) [1988].

(7) Wald, J., M. Farach, M. Tagamets and J. Reggia, "Generating Plausible Diagnostic Hypotheses with Self–Processing Causal Networks", *Jrnl. of Experimental and Theoretical AI*, (in press) [1989].

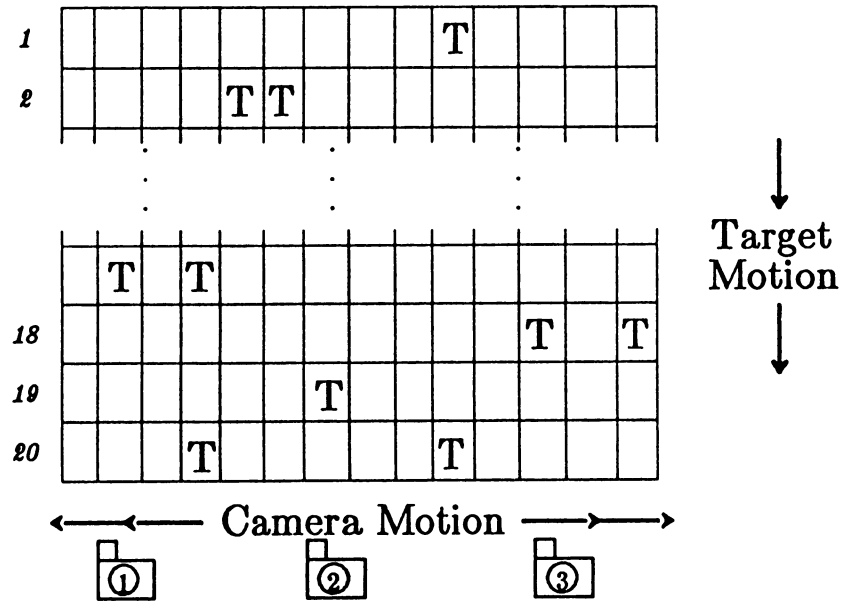


Figure 1. Camera Controller Problem (above): Photographic targets propagate from row 1 of 15 by 20 array of target locations towards row 20, staying within a column. The three horizontally–moving cameras have a one column by one row (row 20) field of view.

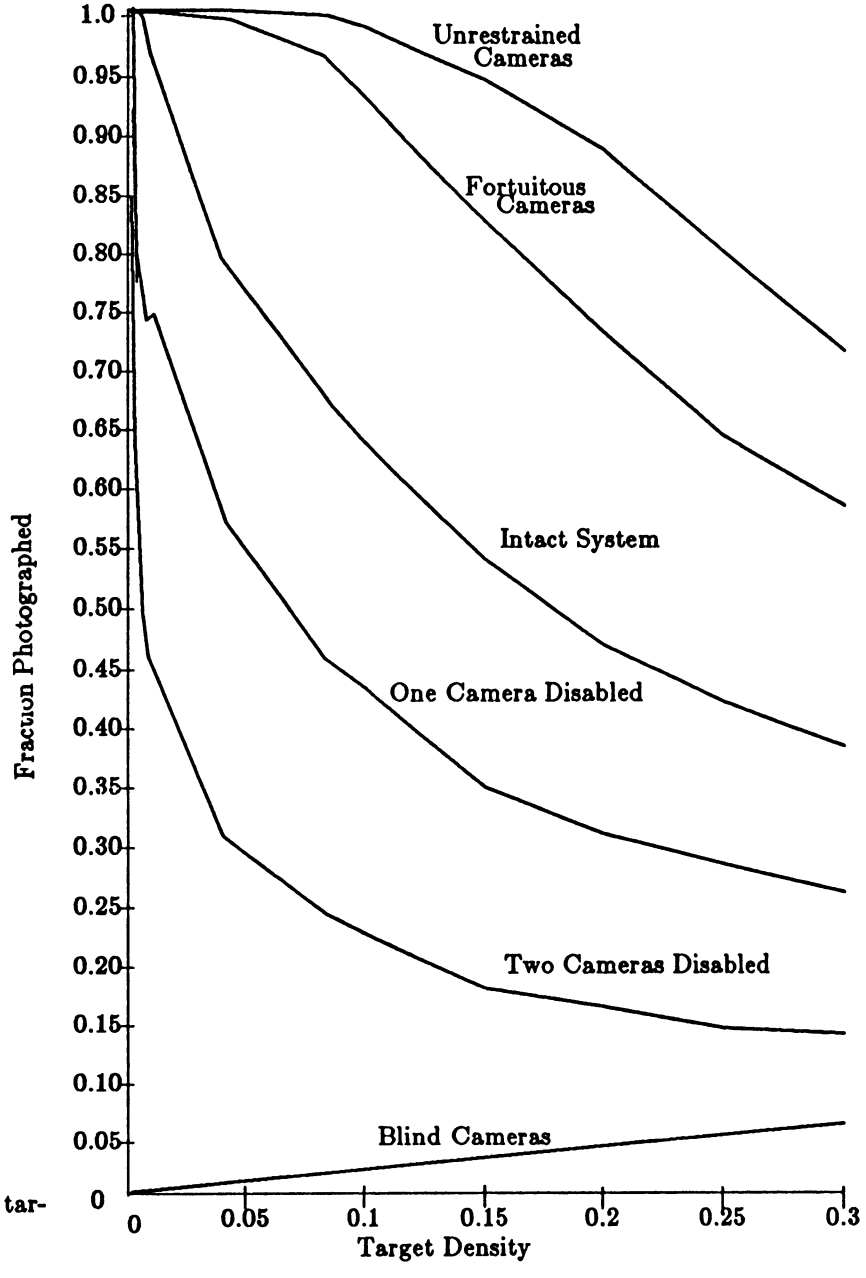


Figure 2. (at right): Fraction of targets photographed to total targets generated as a function of the target density.

A Neurocontroller with Guaranteed Performance for Rigid Robots

Allon Guez J.W. Selinsky

Drexel University, ECE Department
32nd and Chestnut St., Philadelphia, PA 19104

Summary

In this article, we propose and evaluate a neural network based adaptive controller for rigid robots. The proposed neurocontroller incorporates a priori knowledge of the robot's dynamic structure so as to provide proven trajectory tracking and parameter identification. A theorem regarding constructive sufficient conditions for asymptotically stable closed loop learning is stated and used in the design of Exploratory Schedules (ES). ES are reference trajectories which are specifically designed to provide efficient learning. In neurocomputing, ES are the training samples/examples that are used to modify the network architecture during learning. We specify the ES as a desired trajectory that is to be followed to do learning while the manipulator is not doing other useful tasks.

Simulation results of a 2 degrees of freedom (DOF) manipulator are given.

Rigid Robot Dynamics

The Lagrange-Euler formulation of rigid robot dynamics [1], has the form

$$\tau = Y[q,\dot{q},\dot{q},\ddot{q}]\emptyset \qquad (1)$$

where $Y[q,\dot{q},\dot{q},\ddot{q}]$ is an n x p time varying matrix of known and generally nonlinear functions, $\emptyset$ is a p x 1 vector of constant parameters, τ is an n x 1 vector of joint torques, q is an n x 1 vector of joint coordinates, and n is the number of degrees of freedom.

Neurocontroller Design

When (1) is written in terms of the individual torques at each joint, it can be viewed as a single layer linear network, where the inputs to the network are the $Y_{ij}[.]$ and the weights are the $\emptyset_j$ [2],[3]. The $Y_{ij}[.]$ are transcendental algebraic functions of the manipulators states and may be realized via feedforward neural networks that are trained with a suitable learning algorithm.

Let q_d, $\dot{q}_d$, $\ddot{q}_d$ designate the desired trajectory. Following [4] define the virtual reference trajectory $\dot{q}_r$, $\ddot{q}_r$, and the virtual trajectory velocity error $\dot{e}_r$

$$\dot{q}_r = \dot{q}_d - A\,e, \quad \ddot{q}_r = \ddot{q}_d - A\,\dot{e}, \quad \dot{e}_r = \dot{q} - \dot{q}_r = \dot{e} + Ae \qquad (2)$$

where $e[t] = q[t]-q_d[t]$ is the joint coordinate error, and A is a positive definite matrix with constant coefficients.

The output of the neurocontroller implements the control law

$$\tau_i = \sum_{j=1}^{p} Y_{ij}[\,q, \dot{q}, \dot{q}_r, \ddot{q}_r\,]\,\hat{\emptyset}_j + K_{d_{ii}}\,\dot{e}_{r_i}\,, \quad i = 1,2,\ldots,n\,, \qquad (3)$$

where $\hat{\emptyset}$ denotes the estimate of $\emptyset$. Equation (3) has been shown to be asymptotically stable when the learning rule

$$\hat{\emptyset}_j = -\sum_{i=1}^{n} \frac{1}{K_{a_{ii}}} Y_{ij}[\, q, \dot{q}, \dot{q}_r, \ddot{q}_r] \, \dot{e}_{r_i} \,, \qquad j = 1,2, \ldots, p \,. \tag{4}$$

is used [4].

Notice in equation (4) the similarity to the LMS learning rule (see [5]) where the weight change is proportional to the error ε and the input features X. In equation (4) the input features are the $Y_{ij}[.]$ functions and the error is $\dot{e}_{r_i}$.

Figure 1 shows the internal structure of the proposed neurocontroller for joint i. Each feedforward network module is trained to provide one of the $Y_{ij}[q,\dot{q},\dot{q}_r,\ddot{q}_r]$ functions. **This training can be done offline since the $Y_{ij}[q,\dot{q},\dot{q}_r,\ddot{q}_r]$ functions are known a priori and are the same for all rigid robots of the same kinematics and number of degrees of freedom.** The inputs to the neurocontroller are the components of the trajectories as required. The outputs of the neurocontroller are the control torques to be applied at each joint of the manipulator.

Closed Loop Learning and Tracking Via Exploratory Schedules

Theorem 1 [6]: If $p \leq n$, and if

$$\text{Rank} \left\{ \begin{array}{c} Y[\, q_d, \dot{q}_d, \dot{q}_d, \ddot{q}_d] \\ \lim \; t \to \infty \end{array} \right\} = p \,, \tag{5}$$

then the controller specified by equations (3) and (4) guarantees global asymptotic tracking and parameter identification i.e. $q(t) \to q_d(t)$, and $\hat{\emptyset}(t) \to \emptyset(t)$.

Lemma 1 [7]: $\tilde{\emptyset} \to 0 \;\; \forall \; \emptyset \in R^k$, $k \leq n$, such that $\emptyset$ is not contained in

$$N\left(\begin{array}{c} Y[\, q_d, \dot{q}_d, \dot{q}_d, \ddot{q}_d] \\ \lim \; t \to \infty \end{array} \right), \tag{6}$$

where N(.) denotes the null space of the matrix (.), and denotes the parameter estimation error vector. The implication of lemma 1 is that if, $k \leq n$ parameters are not in the null space of equation (6) then the components of the parameter estimation error vector corresponding to the k parameters are zero.

Design of Exploratory Schedules

Lemma 1 implies desired trajectories (which specify $q_d, \dot{q}_d, \ddot{q}_d$) can be designed such that $k \leq n$ specific columns of $Y[\, q_d, \dot{q}_d, \dot{q}_d, \ddot{q}_d]$ will be linearly independent in the limit. The exploratory schedule then consists of a sequence of desired trajectories which are designed to learn different components of the parameter estimation vector $\hat{\emptyset}$, where the number of desired trajectories is such that all p components of $\hat{\emptyset}$ are identified.

Simulation Results: 2 DOF Manipulator

The simulation results reported were obtained using exactly computed values of the $Y_{ij}[.]$ functions, rather than the three-layer neural network form. This simulation provides experimental verification of lemma 1. The dimension of $Y[q, \dot{q}, \dot{q}_r, \ddot{q}_r]$ in the 2 DOF case is 2 x 5 so that p>n,

and at most 2 parameters can be guaranteed to be learned simultaneously.

The Exploratory Schedule for the 2 DOF manipulator (see figure 2) consists of a sequence of three desired trajectories so chosen as to guarantee learning of different parameters. In trajectory 1, $\emptyset_4$ and $\emptyset_5$ are learned. The time period corresponding to trajectory 1 is $0 \leq t < 4.2$. In trajectory 2, $\emptyset_3$ is learned. The time period corresponding to trajectory 2 is $4.2 \leq t < 7.3$. In trajectory 3, $\emptyset_1$ and $\emptyset_2$ are learned. Trajectory 3 corresponds to the time period $7.3 < t \leq 10.3$.

The time period corresponding to identification of $\emptyset_4$ and $\emptyset_5$ is $0 \leq t < 4.2$. As can be seen in figures 3 to 6, the parameter estimation error for $\emptyset_4$ and $\emptyset_5$ approaches zero as all joint errors approach zero at t= 4.2. The time period corresponding to identification of $\emptyset_3$ is $4.2 \leq t < 7.3$. The parameter estimation error for $\emptyset_3$ approaches zero as all joint errors approach zero at t= 7.3. Parameters $\emptyset_1$ and $\emptyset_2$ are identified during the time period $7.3 < t \leq 10.3$.

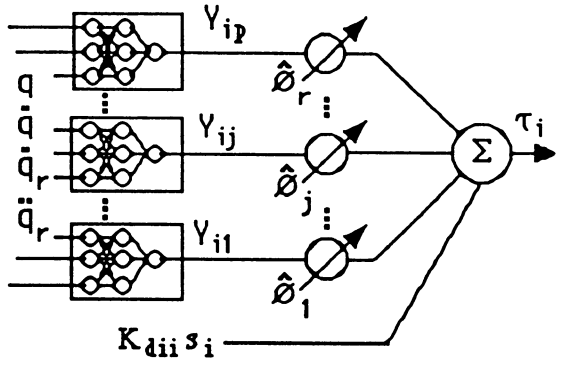


figure 1: Neurocontroller structure for link i.

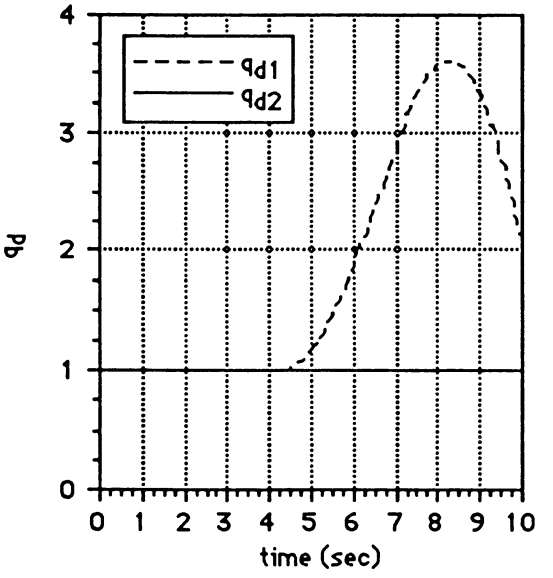


figure 2: Desired joint positions during ES.

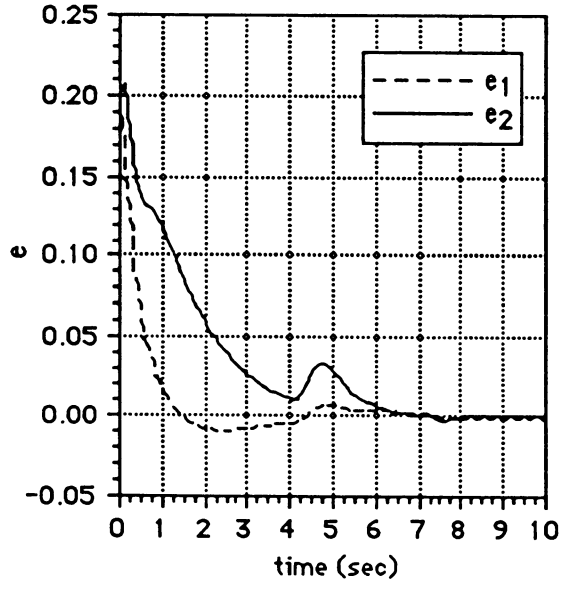


figure 3: Joint position error during ES.

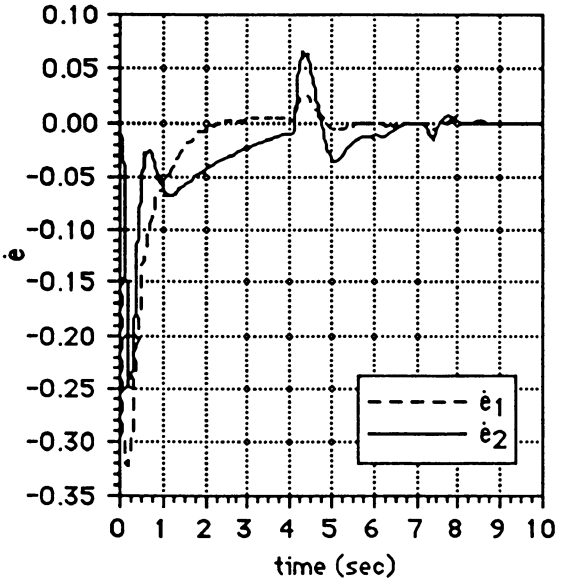


figure 4: Joint velocity error during ES.

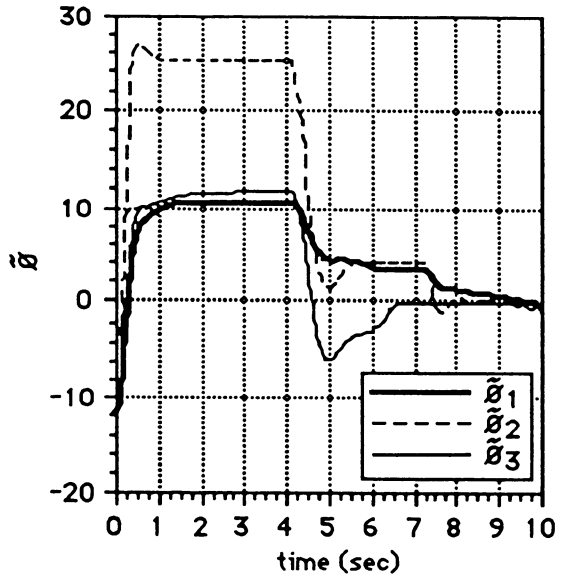


figure 5: Estimation error for $\varnothing_1$, $\varnothing_2$ and $\varnothing_3$.

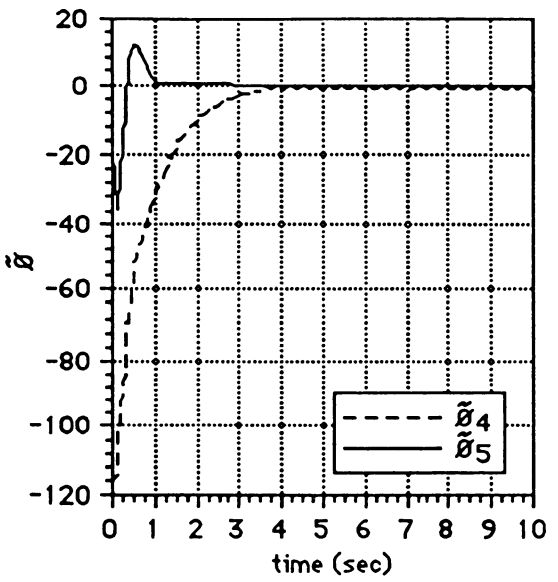


figure 6: Estimation error for $\varnothing_4$ and $\varnothing_5$.

References

[1] Paul, R.P., *Robot Manipulators: Mathematics, Programming, and Control.* Massachusetts: The MIT Press, 1981.

[2] Kawato, M., Furukawa, K. and Suzuki, R., "A hierarchical neural-network model for control and learning of voluntary movement," *Biological Cybernetics*,56, 1987.

[3] Guez, A., and Selinsky J., "Neurocontroller design via supervised and unsupervised learning," to appear in the *Journal of Intelligent and Robotics Systems,* 1989.

[4] Slotine, J.J.E., and Li, W., "Adaptive manipulator control: a case study," *IEEE Trans. Automatic Control*, Vol. 33, no.11, p. 995, 1988.

[5] Widrow, B., and Stearns, S. D., *Adaptive Signal Processing.* New Jersey: Prentice-Hall, 1985.

[6] Guez, A., and Selinsky J., "Design of Exploratory Schedules in Learning and Adaptive Robot Control," accepted for publication in the *International Journal of Adaptive Control and Signal Processing*, 1989.

[7] Guez, A., and Selinsky J., "A Neurocontroller with Guaranteed Tracking and Model Identification for Rigid Robots," submitted for publication, 1989.

Multiple-Order HMM Based Speech Recognition Using Neural Network

Isao Hayakawa and Seiichi Nakagawa
Department of Information & Computer Sciences,
Toyohashi University of Technology
Tenpaku-cyo, Toyohashi, 440, Japan

ABSTRACT

In this paper, a multiple-order hidden Markov model which is approximated by a neural network is proposed. It is the extension of a sequential network and is called by the name of "Neural Markov Model"(NMM).

The NMM with two states achieved a comparable recognition rate with feed-forward type neural network. The recognition rates are 65.4% and 65.0% respectively on the experiment of speaker-independent speech recognition of Japanese syllables [ba], [be], [bo], [da], [de], [do], [ga], [ge] and [go].

1 Introduction

As the methods of speech recognition, DP matching and HMM have been established from the past. DP matching allows warps of time sequential patterns on time domain. It matches a reference pattern which is standard for recognition unit (for example, word or syllable) with an input pattern by expanding and contracting the input pattern non-linearly. This method is powerful for fluctuations of time domain structure of time sequential patterns, but has weakness in the point of fluctuations of spectral patterns which occur by individuality of the speakers.

HMM captures and absorbs the fluctuations of the spectrum and time domain structure. It describes the recognition units by a probabilistic state machine. The parameters of HMM are estimated to absorb the fluctuations of spectrum and time domain structure by training samples.

For the reasons mentioned above, HMM has advantage than DP matching principally. But if we would like to describe patterns by HMM minutely, because of increase of parameters, the accuracy of estimated parameters becomes worse. Thus, actually patterns should be described by coarse models.

Recently, applications of neural networks to speech recognition are being attempted by using its feature extraction ability and learning ability[1]. But when we would like to apply neural networks to speech recognition, arise a problem how to deal the time sequential patterns and its flexibility of time domain structure. From this difficulty, no continuous speech recognition systems using neural network is developed. Most successful systems have used the neural networks only means of feature extraction and then been processed based on the other methods(like HMM or DP matching)[1]

A sequential network, an architecture of neural network intended to deal time sequential patterns by its recurrent structure, has been proposed by Jordan and applied to speech recognition[2]. We note that a sequential network can be considered as a similar model with one-state HMM[3] In this paper, we propose an architecture which is called by the name of Neural Markov Model. It approximates a multiple order HMM by extending a sequential network. This model consists of only neural network without use of other traditional method or logical component. We describe as follows the results of NMM applied to speech recognition of Japanese syllables.

2 Similarity between Sequential Network and HMM[3]

Figure 1 shows a sequential network with three layers. This network has a recurrent structure that feed-backs one to one from the output layer to a context layer which belongs to the input layer, and each unit in the context layer has a self-loop and its coefficient(momentum). The context layer is used to preserve the contents of previous states of the network. Suppose that the activation value of the unit in the output layer was trained to 1.0 when the input pattern associates to the unit, and others 0.0. The recognition result is the unit which corresponds to the maximum activation value.

HMM has several states, output probabilities and transition probabilities. Transition probability is the probability of state to state transition, and output probability is the probability that generates a label when the transition is occurred. The recognition result is the corresponding category to the model presenting the maximum likelihood.

To consider these similarity, each output unit corresponds to each HMM, weights of connection among units of the network absorb the parameters(i.e. output probability and transition probability) of HMM, and the activation of each output units corresponds to the forward probability of HMM.

Further, since the states of the network at proper time is affected by past output of the network, the sequential network approximates a multiple-order HMM.

3 Neural Markov Model

The NMM, described in this paper, has several states, and each state corresponds to one sequential network. This NMM is intended to built the model consists of the concatenation of these sequential networks trained independently. Fundamental structure of the NMM is shown in Figure 2. This network approximates a multiple-order HMM with two states, and consists of two sequential networks through "connect unit". The connect unit has a self-loop like a unit in the context layer, and accepts the copy of activation level of the output unit in the preceding network.

Each of these two sequential networks is trained independently. The connect unit is trained to control the network behavior of "active" or "non-active".

For example, suppose that the previous network was trained to classify "A" and "B", the following network was trained to classify "C" and "D". In case that the time sequential pattern "AD" is given to the network, first, the activation of the units corresponding to "A" in the preceding network becomes a high value while the input pattern belongs to "A", and the state of the following network becomes "active". Next, the input pattern comes to the part of "D", the activation value of the unit corresponding to "A" begins to down, and the activation value of the unit corresponding to "D" in the following network rises.

4 Experiment

4.1 Speech Data

The NMM was applied to speech recognition of Japanese syllables [ba], [be], [bo], [da], [de], [do], [ga], [ge] and [go]. These data was extracted from 216 words and 80 foreign words spoken by 6 male speakers HU, MA, KO, SE, SN and TI. These speech signals were sampled at 10kHz, and transformed to 10 coefficients of LPC mel-cepstrum every 5ms by the 14-order LPC analysis. Training speakers of the networks were HU, MA, KO and SE. Two other speakers SN and TI were used to test the networks. The amount of syllables for training is 216 and for test is 108, respectively.

4.2 Network Architecture and Training

The NMM for this task is shown in Figure 3. Since one basic sequential network can't describe large unit patterns by its limitation of number of connections. Therefore, we choose the phoneme for recognition unit of a sequential network. We trained two sequential networks independently. One classifies the voiced consonants /b/, /d/ and /g/(BDGnet), the other the vowels /a/, /e/ and /o/(AEOnet).

Both the BDGnet and the AEOnet have 20 input units, 3 context units(which coefficient of self-loop is 0.4), 10 hidden units and 3 output units, respectively. The input pattern was given the input layer, coupled of 2 frames and shifted every 1 frame. Each output unit corresponds to each phoneme. The activation value of the output units was set to 1.0 when an input pattern corresponds to the unit and the other units to set 0.0. Further, the AEOnet has a "connect unit" which is trained to control the behavior of the network, "active" or "non-active". When the network is "active", the network performs recognition about the input patterns and when "non-active", the output value of the network is all zero.

4.3 Recognition Results

The recognition rate of the BDGnet applied to test data of /b/, /d/ and /g/ was 70.1%, and the recognition rate of the AEOnet was 87.7% on the test data of /a/, /e/ and /o/.

Then these networks were combined with the connect unit and applied to the speech recognition on the test data of the syllables. The recognition rate of the NMM became 65.4% for test data (better than 70.1%x87.7%=61.5%). These results and the results of the other methods on the same task are shown in Table 1. The performance of the NMM is comparable with a feed-forward neural network which recognizes syllables as static patterns. The NMM has some advantages as being able to treat time-sequential patterns with variable length and to construct a word pattern (network) from basic-unit patterns. The transition of activation level of output units is shown in Figure 4.

The problems of the current NMM is as follows;

- *The effect of connect unit was not sufficient for the targeted function. When the activation value of the BDGnet felt, the activation of AEOnet also felt. As the counterplan for this problem, the self-loop and its coefficient was added to the connect unit. Thus, the AEOnet retained its activation value.
- *The activation values of the BDGnet did not decrease for the section of the following vowel. In most cases of this problem, the activation values of BDGnet was retained its values on the vowel section(see Figure 4, ideally, the activation value falls on the vowel section). This was caused by the fact that vowel patterns were not used for the BDGnet training. If the new training of BDGnet with counter categories like vowels is done, the performance of the NMM will be improved by combining with AEOnet added self-loop.

5 Conclusion

We have proposed the NMM and showed the prospectiveness of the NMM for time sequential patterns by the experiment of speech recognition of Japanese syllables.

We are going to attempt the word recognition by an improved NMM in near future.

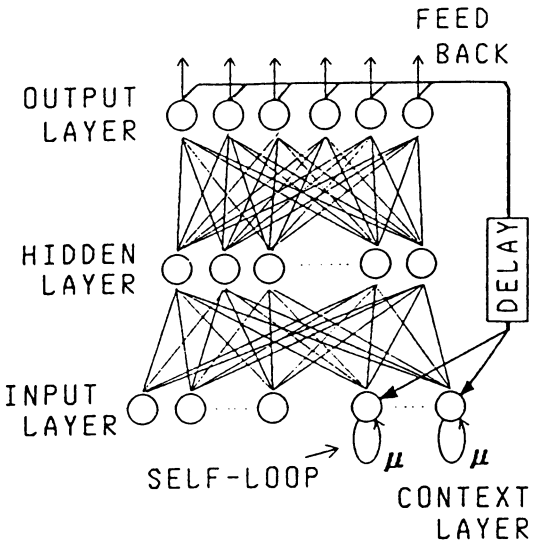


Figure 1: Sequential Network

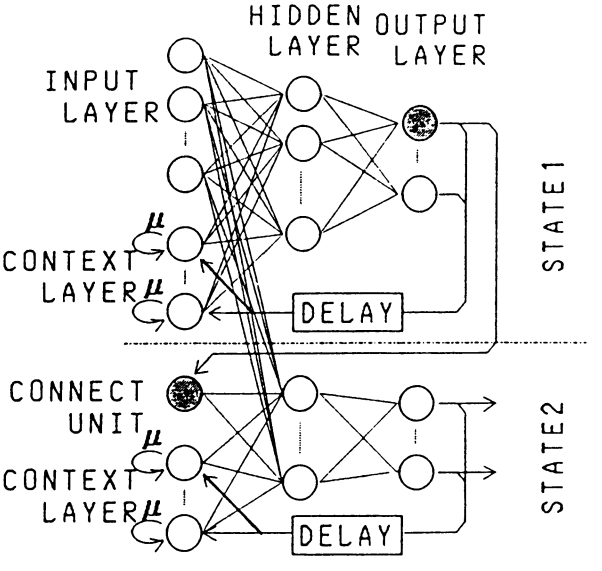


Figure 2: Basic NMM with 2-States

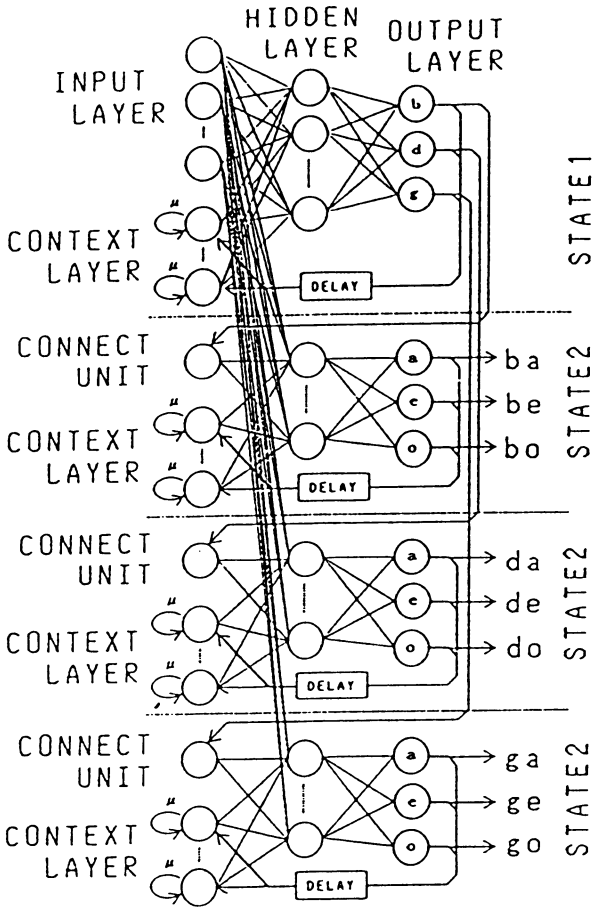


Figure 3: NMM with 2-states for syllables [ba], [be], [bo], [da], [de] and [do]

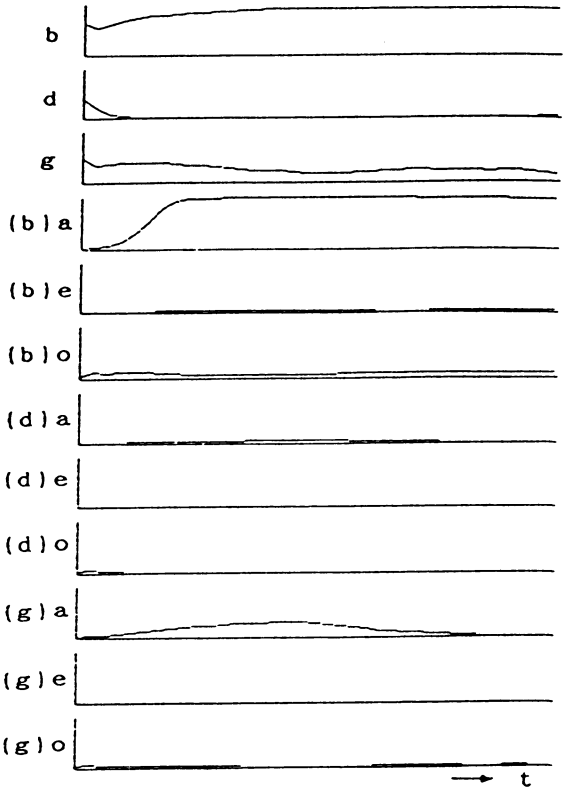


Figure 4: The transition of activation level of NMM's output units

Table 1: Recognition Rates of syllables

Methods	Recognition Rate[%]	Contents
NMM	65.4	See Figure 3
Sequential Network	51.4	Input: 20, Context: 9, Hidden: 10, Output: 9
Feed-Forward	65.0	Input: 150(for 15 frames), Hidden: 10, Output: 9
HMM	52.8	5-state left-to-right type

References

1)R.P.Lippmann; "Review of Neural Networks for Speech Recognition, Neural Computation", Vol.1, No.1, pp.1-38 (1989)
2)S.Anderson et al.; "Dynamic Speech Recognition with Recurrent Networks", Technical Report, Indiana University (1988)
3)H.Bourlard and C.J.Willekens; "Links between Markov models and multilayer perceptrons", Presented at INNS, Philips Research Lab. (1988)

The Use of Modular Neural Networks in Tactile Sensing

Dean Hering, Pradeep Khosla, B.V.K. Vijaya Kumar

Department of Electrical and Computer Engineering
Laboratory for Automated Systems and Information Processing
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

In this paper, we propose modular backpropagation neural networks to implement a task by separating the task into related subtasks, training a smaller network module to learn each subtask, and combining the modules to form a network that accomplishes the entire task. We show the advantage of using modular neural networks by using tactile edge detection as an example.

Introduction

As a neural network grows large, current backpropagation based learning tends to become more difficult due to an increased number of local minima, speed of learning, and the CPU time associated with implementing large neural networks. Furthermore, the internal representation of the learned task tends to be more distributed in large networks and therefore more difficult to understand--especially when the network is unable to learn a task. Smaller networks learning the subtasks are faster in learning, easier to understand internally, and can be used as building blocks for different applications. This approach works especially well when subtasks are related; when the subnetworks are combined the related areas aid in the combining process.

In order to empirically study the use of modular neural networks, we selected an application in tactile sensing. Specifically, we are interested in determining the position and orientation of an edge (with respect to a frame fixed at the center of the tactile sensor) in a thresholded tactile image. This information is important in dynamic object exploration where the goal is to fit these pieces of information together in order to determine the shape of an object.

Approach

Figure 1 shows a 4 by 4 grid that was chosen as the input to a neural network. The task is to learn all 2 unit horizontal (H), vertical (V), +45 degree diagonal (DR), and -45 degree diagonal (DL) lines; based on that training, the network must identify if the line on the input grid (which may be a 2, 3, or 4 unit line) is a H, V, DR or DL line.

Using standard backpropagation techniques as described in [1] and implemented in [2], we first trained a backpropagation network (Network 1) to identify 2 unit horizontal (H), vertical (V), and diagonal (D) lines--in this case +45 degree and -45 degree diagonals were grouped into the diagonal class. The network was able to learn the task and correctly identify the 3 and 4 unit test lines. Training took 1042 epochs to achieve a total error of 0.004 and 8600 epochs to achieve a total error of 0.0001. The network's hidden layer contains seven hidden units. Figure 2a shows Network 1 combined with Network 2 (described below) combined in parallel. The input units, hidden units 16-22, and output units 29-31 of the final network compose Network 1; the unit numbers of the original network are identified in Figure 2a.

To understand the internal representation of the input data, we plotted the activation levels for the different input classes and examined the connection weights and biases. From this data we observed that the network formed distinct representations for each class of data; furthermore, we found that other information, such as an indication of the active row for an H input, was present in the representation. We observed that for diagonal inputs the network represented right diagonals on the main diagonal differently than those off the main diagonal; however, the representation for a right on-main diagonal was identical to that for a left off-main diagonal and vice versa. We observed other interesting tendencies of the network such as weight combinations; space considerations prevent us from describing the other characteristics further.

We then modified Network 1 by adding an additional output. Now the outputs were to identify if an input line was H, V, DR or DL, again based on learning the 2 unit input lines and generalizing to 3 and 4 unit input lines. We performed eight trials in an attempt to train the network; in each trial we used a different set of initial weights to allow the gradient descent process to begin at a different initial point. In each trial the network was only able to learn part of the task. For example, in one trial the network correctly identified all H, V, and DR inputs, but incorrectly identified all DL inputs; in another trial the network correctly identified all H and DL inputs but

incorrectly identified the V and DR inputs, and so forth. In each case, the network correctly identified all elements of one or more classes and incorrectly identified all elements of the other classes.

Next, we added more hidden units to the network. We attempted four more trials with up to thirteen hidden units; the network behaved as above. In addition to those hidden units, we added a second layer of hidden units containing two units. We performed several additional trials with the second hidden layer connected to both the inputs and the first hidden layer and with the second hidden layer connected only to the first hidden layer. Again the network correctly identified all members of one or more classes, but incorrectly identified members of the other classes.

The above results do not conclusively mandate that the task is an impossible one. Certainly by changing the learning rate and momentum parameters in conjunction with the number of hidden units the task should be learnable; however, the results of the above experiments do indicate that this might be a reasonable task to break into subtasks so that the overall task might be accomplished by using a combination of smaller network modules.

Network 1 can identify lines as H, V, and D. Since Network 1 cannot distinguish between diagonal slopes (which may explain the difficulty in training the network to identify DR and DL inputs), a reasonable separation of subtasks would be to use that network and another (Network 2) which can identify the slope of an input diagonal line as +45 degrees (DR) and -45 degrees (DL) to form the overall network. Note that these two subtasks overlap--the D output of Network 1 will be on when the DR/DL outputs of Network 2 are valid. This overlap is useful in combining the two networks.

Network 2 consists of the sixteen unit input layer, a six unit hidden layer, and a two unit (DR, DL) output layer. We trained the network to identify 2 unit diagonals as being +45 degree (DR) or - 45 degree (DL). No H or V inputs were shown to the network. The network took 1484 epochs to learn the task with a total error of 0.001 and 7157 epochs to achieve a total error of 0.0001. We tested the network with 2, 3, and 4 unit diagonal inputs and it correctly classified all inputs. Once again, due to the small size of the network, the internal representations of the classes of patterns were fairly easy to analyze and understand. Figure 2a shows Network 2 (input units, hidden units 23-28, and output units 32 and 33 of the final network) with the unit numbers of the original and final network identified.

To sum: Network 1 learned to identify whether an input was horizontal (H), vertical (V), or diagonal (D)--independent of slope; the Network 2 learned to identify the slope of an input diagonal line as +45 degrees (diagonal right, DR) or -45 degrees (diagonal left, DL) . The networks were trained on 2 input units only--both networks correctly learned to identify 3 and 4 unit inputs constructed from the 2 unit building blocks. Network 1 was trained on 2 unit horizontal, vertical, and diagonal lines and correctly identifies 3 and 4 unit lines constructed from the 2 unit lines. Network 2 was trained on 2 unit diagonal lines and correctly identifies the slope of 3 and 4 unit diagonal lines constructed from the 2 unit lines. These two network modules were then combined to form a larger network which accomplishes the entire task of identifying 2, 3, and 4 unit H, V, DR, and DL inputs.

Combining Modules

We combined the networks using two configurations. The first is a parallel configuration in which the hidden layers were combined and the output layers were combined. The resulting network has five outputs (H, V, D, DR, DL). Since Network 2 has undefined outputs for horizontal and vertical inputs, the outputs of the combined network must be decoded so that invalid states (DR and DL active during H or V inputs) can be ignored.

The second combination network again combines hidden units but makes the D output of Network 1 a second hidden layer unit. This unit is used to inhibit (via two added connections) the DR and DL outputs of the Network 2 whenever the input is horizontal or vertical. Thus, no output decoding is necessary in this configuration--the outputs of the network are the four desired outputs. Using a custom program, we combined the weights and biases of the two network modules in the two configurations above. Figure 2 shows the two combinations with the module unit numbers and the final unit numbers are identified.

There is an interesting note on the second combination: the D output of Network 1 is active whenever the input is a diagonal line. Since that unit must inhibit the DR and DL outputs of Network 2, the sense of the output must be reversed; i.e., D should be off whenever the input is a diagonal and on whenever the input is H or V so that DR and DL can be inhibited. To accomplish this, we sign reversed all connection weights that are input to D (unit 29 in the final network) and we reversed the unit's bias. In this manner, inputs that tended to turn on D now turn it off, and inputs that tended to turn off D now turn it on when they are present. Thus D has been "complemented" in a

sense. We gave the two weights that were added from the D unit to the DR and DL units values that caused the product of the D unit activation and the connection weight to exceed in magnitude the maximum of all other sums of products to the DR and DL units and to be of the proper sign to inhibit the DR and DL outputs when D is active.

We tested the two combined networks--each performed identically to the component module networks. Additionally, we trained the second combination network to provide all zero outputs for an all zero input and the sixteen possible 1 unit inputs. This took required an additional 174 epochs of training to accomplish.

Training an Equivalent Larger Network

The two modular networks combine to form a network that contains thirteen hidden units arranged in two layers of twelve units and one unit, respectively. Since this configuration is known to work, it seemed plausible that one network should be trainable to accomplish the entire task--as was conjectured in the beginning experiments. In keeping with the known working network, we constructed a neural network containing two layers to learn the task. The first hidden layer contained fourteen units, each connected to all the second layer units and all the outputs; the second layer contained two units, each connected to all the outputs. The same training patterns and learning parameters that were used in the module training were used in training this network. We performed three trials with different initial starting weights ; in each case the network failed to learn the task.

We then eliminated a number of the connections from the network so that the structure of the network was nearly identical to the second combination network. The only difference was that each hidden layer contained one more unit than the combination network. Again, we performed several trials using identical learning parameters as the component modules; again the network failed to learn the task.

Next, we used the combination network structure shown in Figure 2b. We performed several trials on it with random initial weights and the same initial weights that were used in training the modules. The network failed to learn the task using the same learning parameters. However, by setting the learning rate to a lower level (0.05) and using the initial weights used in the module training, the network began to learn the task. By tweeking the learning rate as the processing continued, the network was able to learn the 2 unit H, V, DR, DL classes after 1800 epochs with a total error of 0.005. We then tested the network on the 3 and 4 unit inputs; the network had relatively large errors on seven patterns (fully activating the correct output but partially activating others as well) and incorrectly identified two others. Thus its performance was not nearly as good as the combined modular networks. However, after 80 additional epochs of training on *all* patterns, the network learned the overall 2, 3, and 4 unit task.

Thus, by tweeking the learning parameters, a network with the same structure as the combination modular network was able to learn the 2 unit task; however, it did not generalize to the 3 and 4 unit inputs as well as the modular network.

Conclusion

In this paper we have given an example of using modular neural network techniques. A task (to classify input lines as H, V, DR, or DL) to be implemented using a backpropagation neural network was separated into related subtasks (classifying H, V, and D lines and classifying the slope of a diagonal line); these subtasks were each learned on small neural networks, which were combined to form a network to implement the overall task. Each module's internal representation was analyzed and understood, due to the small number of hidden units. Additionally, the training of the modules themselves was modular--the networks trained on 2 unit input lines and then were tested on 3 and 4 unit lines constructed from the 2 unit lines. By understanding these modules, their use as components in other networks, some of which may take advantage of some of the internal representations not exploited in the present problem, is made possible. Furthermore, knowing the internal functioning of the modules allows them to be adjusted (as in the complementing of the D unit in the example) to fit a given need. Once combined, the resulting network is able to learn additional tasks.

This work is not an effort to prove that modular networks learn better than larger networks, or that larger networks cannot learn the same tasks; rather, its purpose is to show through an example how modular neural networks provide a working network with components that are easily understood in terms of their internal representation and function. To use modular neural networks effectively, one must be able to separate the task into subtasks that are related or independent so that the combination of the modules is straightforward and advantageous over training an overall network. Modular training requires similar needs.

By using modular neural networks, basic building blocks can be trained for use in constructing larger networks which use similar functions. As a library of functions is assimilated, training time for larger networks may

be reduced through the use of the modules. Current applications being explored at Carnegie Mellon University include the ones mentioned above and control of CMU's Direct Drive Arm II.

References

1. Rumelhart, Hinton, and Williams, "Learning the Internal Representations by Error Propagation", in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume2: Psychological and Biological Models*, MIT Press, Cambridge, Massachusetts, 1986, ch. 8.

2. McClelland and Rumelhart, *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*, Bradford Books, Cambridge, Massachusetts, 1988.

Figure 1
Typical Input Patterns for Input Grid

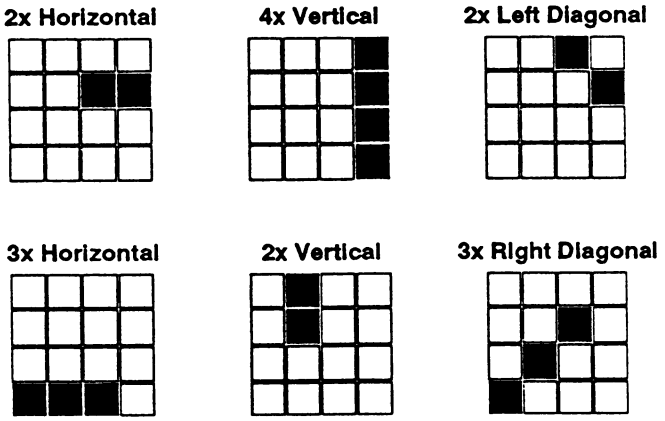
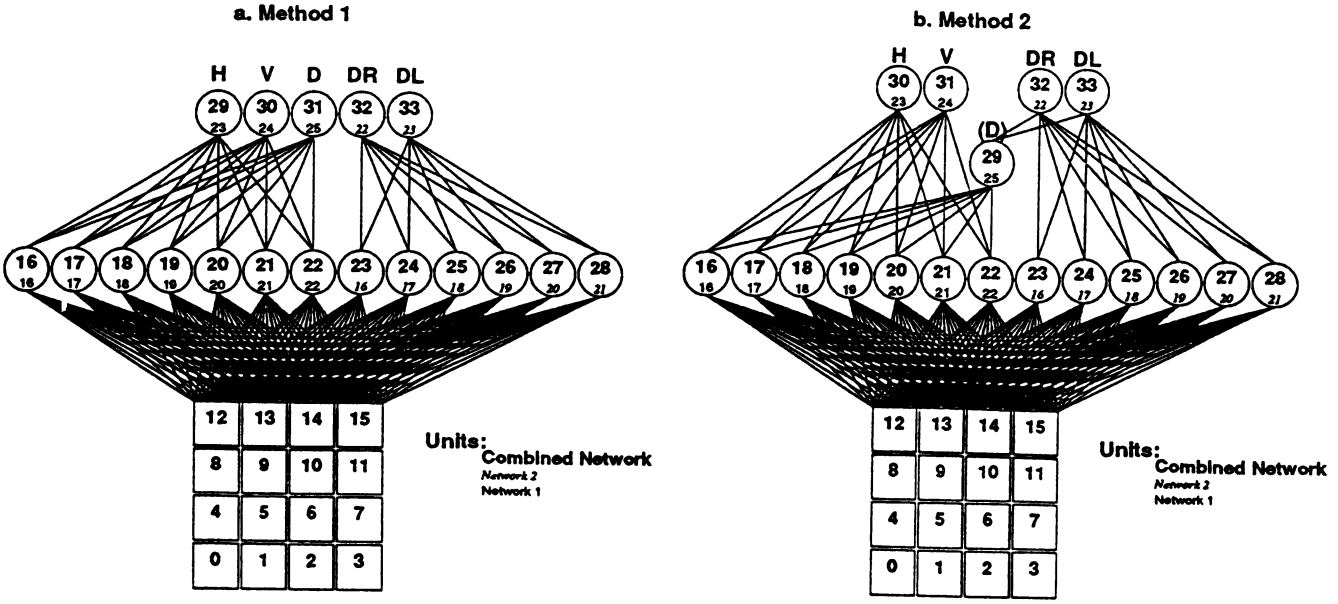


Figure 2: Combining Network Modules



Classification of Unaveraged Evoked Cortical Magnetic Fields

Lasse Holmström *
Rolf Nevanlinna Institute

Petri Koistinen
Rolf Nevanlinna Institute

Risto J. Ilmoniemi
Low Temperature Laboratory, Helsinki University of Technology

1 Introduction

Signal transmission from one neuron to another in the brain involves electric currents. When thousands of neighboring neurons act together, the currents produce a weak magnetic field that can be detected outside the skull. The recording of these fields is called magnetoencephalography (MEG). The only sensor sensitive enough for MEG is the Superconducting QUantum Interference Device or the SQUID. The MEG data used in this paper were measured in the Low Temperature Laboratory of the Helsinki University of Technology using the 7-channel SQUID magnetometer constructed in the laboratory [4].

In our experiments, the subject was presented either a low or a high tone pip and was instructed to respond by pressing a button with the left or the right thumb, respectively. The purpose of this work was to study the feasibility of classifying individual MEG responses into the two categories with pattern recognition and neural network methods. Such analysis may prove useful for diagnostic purposes (c.f. [2]).

The conventional approach is to use averages computed from a large number of individual responses. As shown in Fig. 1, this method can result in effective noise cancellation. Note that "noise" here includes all ongoing brain activity not associated with the response itself. Our approach, however, is to classify *single* measurements. While a successful system based on this approach would have many advantages over the use of averages (e.g., in diagnostic work or in monitoring patients), the high level of noise in a single measurement data vector makes signal classification very difficult.

2 Data analysis

2.1 The experiment

Our experimental data were gathered during several measurement sessions, each of which consisted of about 100 low-frequency (500 Hz) and about 100 high-frequency (1000 Hz) tones presented in random order. The magnetometer was placed on top of the subject's head giving equal weight to the motor cortex responses originating in the two cortical hemispheres. The data were gathered from 7 MEG channels at a sampling frequency of 400 Hz for an interval of 1.5 s. We have analyzed responses from two subjects, two measurement sessions per subject. To study the feasibility of signal classification, an interval of 150 ms (60 points) following the pressing of the button was extracted from each measurement. We also experimented with the time interval following the stimulus presentation but the resulting error rate turned out to be quite high. Probably the location of the magnetometer was too far from the auditory cortex, which is the primary source of the responses during this time interval [3].

2.2 Preprocessing and spatial filtering

During preprocessing the data from each channel of each trial (600 points) were normalized to unit standard deviation and a linear trend was removed from the data by subtracting the best least-squares straight line. Next, the appropriate 60 data points were extracted from the trial.

*Author's address: Rolf Nevanlinna Institute, Teollisuuskatu 23, SF-00510 Helsinki, Finland.

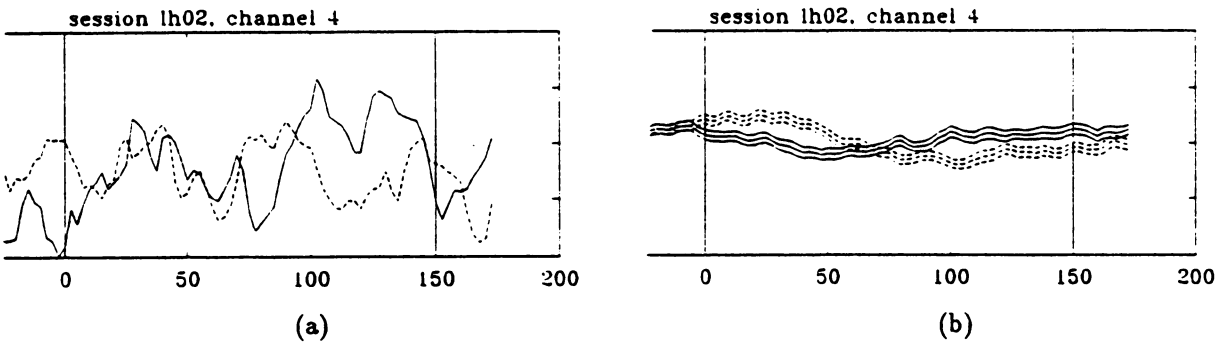


Figure 1: Single vs. averaged measurements. (a) Single normalized measurements: one trial with a low stimulus (solid line) and one trial with a high stimulus (dashed line). (b) Averaged measurements: the middle solid line is the average of 92 trials with low-tone stimuli, while the middle dashed line is the average of 94 trials with high-tone stimuli. The data have been shifted so that all responses are aligned at 0 ms. In addition to the two averages, $a_i(t)$, also the quantites $a_i(t) - b_i(t)$ and $a_i(t) + b_i(t)$ are shown, where $b_i(t)$ is the standard error of the mean of $a_i(t)$. Vertical scale is the same in (a) and (b).

The data were filtered using a noise-canceling filter [7, ch. 12]. Channel 4 (the sensor placed in the middle of the subject's head) was chosen as the primary input which is thought to consist of signal and uncorrelated noise: $d(t) = s(t) + n(t)$. The remaining channels were used as reference inputs $x(t) \in \mathbf{R}^6$ to derive an estimate $\hat{n}(t)$ of $n(t)$:

$$\begin{aligned} w_{opt} &= \arg\min_{w} \int_{0\text{ ms}}^{150\text{ ms}} E\{(d(t) - f(w; x(t)))^2\} dt, \\ \hat{n}(t) &= f(w_{opt}; x(t)). \end{aligned}$$

The filtered signal is then $d(t) - \hat{n}(t)$. Here $f(w; x)$ is either $w^T x$ (a linear spatial filter) or the output of a feedforward net with weights w (a backpropagation spatial filter).

Next the filtered data were decimated to 20 components by segmenting the data into 20 separate parts and by calculating the averages of the data inside each segment. The results constitute our pattern vectors which were classified with several different feature extraction and classification methods.

2.3 Feature extraction and classification

The dimension of feature vectors was chosen to be 5. The simplest feature vectors were obtained by direct decimation from 20-dimensional vectors to 5-dimensional vectors.

The second feature extractor was designed with the SFS (Sequential Forward Selection) algorithm [1, ch. 5.6] based on maximizing the estimated Euclidean inter-class distance.

The third feature extractor was based on projections on the eigenvectors of the average of the class conditional covariance matrices, $S_W = P_1 C_1 + P_2 C_2$. Here P_1, P_2 and C_1, C_2 are the *a priori* probabilities and covariance matrices of the two classes, respectively. If U is the 20 x 20 matrix whose rows are the eigenvectors of S_W, i.e., the Karhunen-Loève basis vectors, then one can transform the pattern vectors $x \in \mathbf{R}^{20}$ to $y = Ux$ and apply the SFS algorithm to select the best 5 indices (in the sense of inter-class distance) from the y vectors. This leads to a feature extractor $y = Vx$, where V is a 5 x 20 submatrix of U. The feature extractor obtained in this way was somewhat better than the one obtained by arranging the eigenvectors with a standard criterion designed to maximize the ratio of the inter-class distance and the intra-class variance [1, ch. 9.6].

The 5-dimensional feature vectors were classified using several conventional classifiers (see, e.g., [8,1]) and a nearest-neighbor classifier trained with the LVQ algorithm of Kohonen [6,5]. The linear classifier is obtained by solving for $w_i, i = 0, \ldots, 5$, the (overdetermined) linear least-squares problem

$$w_0 + w^T x = \begin{cases} 1 & \text{for each } x\text{: } x \text{ a design vector from class 1} \\ -1 & \text{for each } x\text{: } x \text{ a design vector from class 2} \end{cases}$$

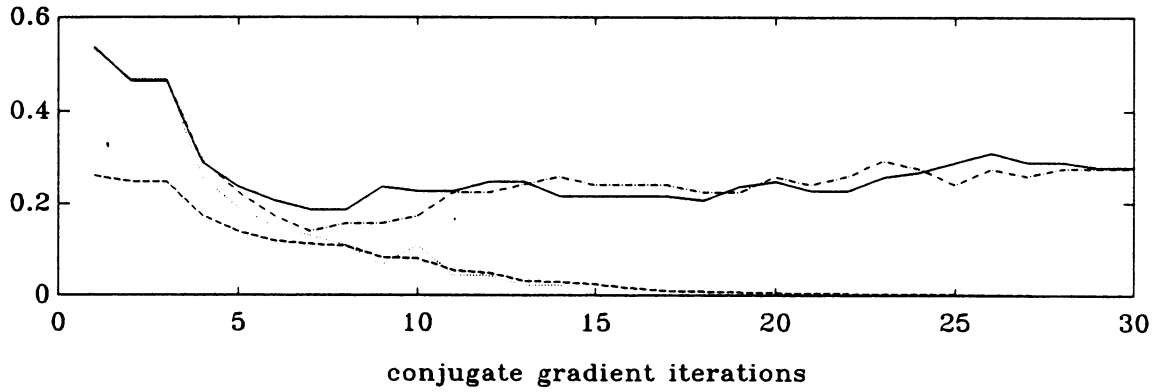


Figure 2: Stopping rule for a backpropagation classifier. Shown are the object function (dashed line), and classification error rates for the set D_1 (dotted line), set D_2 (dash-dotted line) and an independent test set (solid line). The aim is to find a minimum of the solid curve when only the three other curves are known.

Feature extr.	*Classifier*					
	lin.	min. dist.		par. Bayes	LVQ	BP
Decim.	23	18	22	22	20	
SFS	21	21	19	24	20	
KL-SFS	17	18	18	21	19	
No extr.						20

(a)

Feature extr.	*Classifier*					
	lin.	min. dist.		par. Bayes	LVQ	BP
Decim.	22	17	22	24	20	
SFS	17	17	18	18	21	
KL-SFS	23	18	22	25	21	
No extr.						20

(b)

Feature extr.	*Classifier*					
	lin.	min. dist.		par. Bayes	LVQ	BP
Decim.	17	17	17	17	17	
SFS	15	13	16	17	17	
KL-SFS	17	19	17	14	15	
No extr.						17

(c)

Figure 3: Averages of classification error percentages. (a) Linear spatial filter, average over the four sessions. (b) Backpropagation spatial filter, average over the four sessions. (c) Session "lh02" with a linear spatial filter.

The classification rule is to assign an unknown vector x to class 1, if $w_0 + w^T x > 0$ and to class 2 otherwise. The minimum-distance classifier refers to a classifier that assigns an unknown vector x to that class, whose mean vector is nearest to x. This classifier was constructed both using Euclidean distance $d(x,y)^2 = (x-y)^T(x-y)$ and using weighted Euclidean distance $d(x,y)^2 = (x-y)^T {S_W}^{-1}(x-y)$. The parametric Bayes classifier is the Bayes classifier minimizing the probability of misclassification assuming the data comes from normal distributions with the class conditional mean vectors and covariance matrices estimated from training data. The LVQ algorithm was initialized by first running the Kohonen feature map algorithm with zero neighborhood radius on a randomly selected set of reference vectors from the two classes.

The 20-dimensional pattern vectors were classified with a backpropagation classifier—a feedforward net with 20 inputs, 5 units in the middle layer, and 1 output. Such a net contains 111 adaptable weights but we had only about 100 vectors to design the classifier. In this case the backpropagation approach is akin to solving an underdetermined nonlinear least-squares problem. The error can then be brought down to zero but then there is no guarantee as for the quality of the resulting classifier. We solved this problem by stopping the optimization at an early stage. To this end we divided the design vectors into two sets, D_1 and D_2, trained the network with vectors D_1, and monitored the classifying performance of the network with vectors D_2. We selected those weights that gave the best classifying performance during the minimization for the set D_2—see Fig. 2 for an illustration.

3 Results

The data vectors were divided into two sets: A and B. The filters, feature extractors and classifiers were designed on the basis of set A. The performances of the classifiers were assessed with set B. Finally the roles of A and B were reversed. This gave us two classification error percentages per session for the conventional classifiers, and by using several random initializations, 10 for LVQ and 12 for backpropagation classifiers. Fig. 3 (a) and (b) summarize the results and Fig. 3 (c) shows the results for our best session.

4 Conclusions

The particular classification task considered in this paper is difficult because of the high level of noise and the relatively small number of training vectors available for feature extractors and pattern classifiers. Nevertheless, the results obtained do appear to suggest that pattern classification is feasible. To be of practical use the error rates must, however, be further reduced. It is our belief that this cannot be achieved by experimenting with new feature extractors and classifiers. Instead, more measurements per session are needed and a better use of the available multichannel information is called for. The considerable variation of error rates between different sessions suggests that the experimental setup should be better controlled. Neural network methods appear to be competitive with more traditional approaches. The multi-layer feedforward net is easy to implement, it can be used in spatial filtering and it combines feature extraction and classification. However, especially with a small number of training vectors, one has to monitor the training of the net in order to prevent over-learning.

References

[1] P.A. Devijver and J. Kittler. *Pattern Recognition: A Statistical Approach.* Prentice-Hall International, 1982.

[2] A.S. Gevins and A. Rémond, editors. *Handbook of Electroencephalography and Clinical Neurophysiology. Revised Series.* Volume 1: Methods of Analysis of Brain Electrical and Magnetic Signals and Volume 2: Clinical Applications of Computer Analysis of EEG and other Neurophysiological Signals, Elsevier, 1987.

[3] R. Hari. The neuromagnetic method in the study of the human auditory cortex. In M. Hoke, F. Grandori, and G.L. Romani, editors, *Auditory Evoked Magnetic Fields and Potentials. Adv. Audiol.*, Karger, 1989. vol 6.

[4] J. Knuutila, S. Ahlfors, A. Ahonen, J. Hällström, M. Kajola, O.V. Lounasmaa, V. Vilkman, and C. Tesche. Large-area low-noise seven-channel dc SQUID magnetometer for brain research. *Rev. Sci. Instrum.*, 58(11):2145–2156, Nov. 1987.

[5] T. Kohonen. *Learning vector quantization for pattern recognition.* Technical Report TKK-F-A601, Helsinki University of Technology, Finland, 1986.

[6] T. Kohonen, G. Barna, and R. Chrisley. Statistical pattern recognition with neural networks: benchmarking studies. In *IEEE International Conference on Neural Networks*, pages I:61–68, 1988.

[7] B. Widrow and S.D. Stearns. *Adaptive Signal Processing.* Prentice-Hall, 1985.

[8] T. Y. Young and T. W. Calvert. *Classification, Estimation and Pattern Recognition.* American Elsevier Publishing Co., Inc., New York, 1974.

A Two-Layer Hopfield-Tank Network for Motion Estimation

R.M. Iñigo† and C. Narathong‡

†EE. Dept., Univ. of Virginia, Charlottesville, VA 22903
‡EE. Dept., Univ. of Wisconsin, Platteville, WI 53818

Abstract

Real-time visual tracking is a difficult problem requiring high speed processing. We have previously reported a fast tracking algorithm (the *Line Correlator Tracker* (LCT)) [1,2] capable of estimating displacement for a sequence of images using a conventional rectangular sensor. When used with a logarithmic-spiral sensor [3], changes of scale can also be estimated. Although the algorithm can be implemented using sequential or parallel digital processing, a Hopfield-Tank (HT) network implementation is potentially simpler and faster.

1. Introduction

Velocity computation using neural networks has recently been reported by several researchers [4,5,6,7]. In this paper a two-layer HT network to estimate 2D motion based on the LCT is described. The HT network is well suited for small optimization problems. The LCT is used to compute individual row displacements, implemented in the first layer of the HT network. The second layer is used to combine the individual displacements and then produces the overall horizontal and vertical displacements. The use of two layers increases the probability of convergence and reduces the sensitivity to initial conditions and to constant parameters. To support the claims, we present the simulation results on both synthetic and natural images. Due to lack of space, computer simulation on natural images will be presented at the conference.

2. Line Correlator Tracker

The LCT was developed for real time tracking applications as part of a system with biological vision features. The system has two sensors, a conventional uniform rectangular grid and a grid of exponentially increasing rings, conformally mapped to a *computation plane* by the natural log function. The LCT uses the rectangular grid for translation in image plane and the log-spiral grid for scaling and rotation about the optical axis. Details of the architecture and results of simulations are given in [2,3]. For the rectangular grid, displacements in the x direction are given by

$$\Delta x_{lk} = \frac{\sum_j R_{lj}^2 - \sum_j R_{lj} T_{(l+k)j}}{\sum_j \frac{\partial I}{\partial x_{(l+k)j}} R_{lj}} \tag{1}$$

where l and k are indices within a tracking window of size $n\times(2m+1)$. R and T are two successive image frames. The overall horizontal and vertical displacements are given by

$$\begin{aligned} \Delta X &= \Delta \bar{x}_k \\ \Delta Y &= k \end{aligned} \tag{2}$$

respectively. The $\Delta \bar{x}_k$ is the average row displacement in the kth column. The index k is calculated by

$$\sigma_{n_k} = min(\sigma_{n_{-k_{max}}}, ..., \sigma_{n_{+k_{max}}}) \tag{3}$$

$$f^{-1}(\sigma_{n_k}) = f^{-1}(min(\sigma_{n_{-k_{max}}}, ..., \sigma_{n_{+k_{max}}})) = k \tag{4}$$

where σ_{n_k} is the standard deviation of n estimates for the kth column and f is a one to one function which maps the σ_{n_k} to its index k.

The above algorithm utilizes only point-to-point correlation, not full correlation. It was shown in [1] that computation time of 40%, or more, can be saved over the conventional correlation algorithm. Furthermore, simulation results with real images have shown that the algorithm possesses good noise immunity and it can be applied directly to a tracking window without segmentation.

3. The Network Architecture

The network is a two-layer HT network. The first layer consists of $(2m + 1)$ planes. Each plane contains n HT networks. There are $(2D + 1)$ neurons in each network, where D is the maximum row displacement. The row displacements are computed by each of these networks. The output of these networks are fed directly to the second layer. This layer computes the overall displacement based on the outputs of the first layer. There is a single plane containing $(2m + 1)\times(2D + 1)$ neurons in the second layer. The outputs from the second layer are the horizontal and vertical displacements. Figure 1 illustrates the overall network. The following subsections describe the energy functions of the two layers.

3.1. The First Layer

The model in the first layer contains binary neurons representing the row displacement between the two images. We use $n\times(2m + 1)\times(2D + 1)$ neurons. For implementation, we discretize the row displacement by letting $-D \le j \le +D$. We also let $[V_j]_{lk}$ represent the state of the jth neuron of lth row and kth plane. When the neuron $[V_j]_{lk}$ is 1, this means that the row displacement for the lth row in the kth plane is j. If subpixel accuracy is desired, one can simply increase the number of neurons within the tracking window. For this model to function, an energy function must be developed such that only one neuron within each row is turned on when the network reaches stable state. The energy function for the lth row of kth plane is given below.

$$E_{lk} = \frac{A}{2}\left[\sum_i\sum_{j\neq i}\left[V_i\right]_{lk}\left[V_j\right]_{lk} - \sum_i\left[V_i\right]_{lk}\right] + \frac{B}{2}\left[\sum_j\left[V_j\right]_{lk} - 1\right]^2 + C'\sum_j\left[S_{lk} - Q_{lk}i\right]^2\left[V_j\right]_{lk} \tag{5}$$

where

$$S_{lk} = \sum_j R_{lj}^2 - \sum_j R_{lj}T_{(l+k)j}; \quad Q_{lk} = \sum_j \frac{\partial I}{\partial x_{(l+k)j}}R_{lj}; \quad C' = \frac{C}{(i+\varepsilon)^2}; \quad \Delta x_{lk} \to i; \quad 0 < \varepsilon \le 1$$

The first and the second terms in (5) provide row inhibition and global inhibition, respectively. These two terms assure that there is one and only one neuron "on" for the lth row of the kth plane when the network reaches the stable state. These two terms are also known as the constraint terms. The last term is the data term or the objective term. Without this term, a neuron will be on randomly. Notice that this term is taken directly from (1). By manipulating (5) and then equating it to the general energy function given in [8], the weight connection matrix and the bias

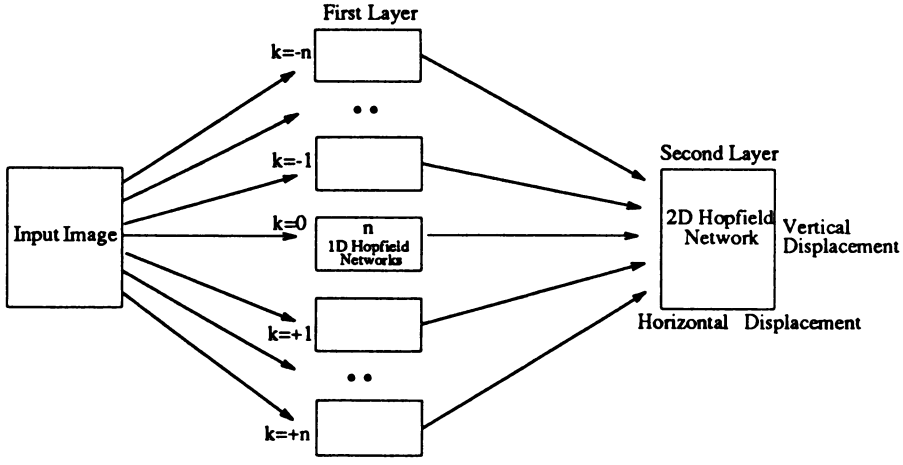


Figure 1 The Network Structure

input (excitation term) are derived as

$$\left[T_{ij}\right]_{lk} = -A(1-\delta_{ij}) - B$$
$$\left[I_i\right]_{lk} = \frac{A}{2} + B - C'\left[S_{lk} - Q_{lk}\, i\right]^2 \tag{6}$$

Notice that the quadratic terms in the energy function define a connection matrix and the linear terms define input bias current.

3.2. The Second Layer

The second layer contains binary neurons representing the overall row and column displacements between the two images and consists of $(2m+1)\times(2D+1)$ neurons. The horizontal displacement is discretized by letting $-D \le j \le +D$. The vertical displacement is represented by index k. V_{kj} represents the state of the kjth neuron. When V_{kj} is 1, the horizontal displacement is j and the vertical displacement is k. When the network reaches stable state, there should only one neuron on for the entire layer. Since this is a 2D problem, a 4-dimensional energy function is required. Using the x and y subscripts as the row indices and the i and j as the column indices, such an energy function is given by

$$E = \frac{A}{2}\sum_x\sum_i\sum_y\sum_j V_{xi}V_{yj}\left[1-\delta_{xi,yj}\right] + \frac{B}{2}\left[\sum_x\sum_i V_{xi} - 1\right]^2 - C\sum_x\sum_i\left[\sum_l V'_{li}\right]_x V_{xi} \tag{7}$$

The first term provides row and column inhibitions, and the second term global inhibition, thus assuring that only one neuron is turned on for the entire layer. The last term forces the neuron corresponding to the object translation to turn on. The V'_l is the neuron output at the lth row of the first layer. By summing up the neurons from the different rows at the same column for a given kth plane (corresponding to the xth row in the second layer), we can provide proper excitation for each neuron in the particular location. For example, if l neurons in the jth column of the kth plane in the first layer are all 1's or nearly all 1's (few 0's), then the output of the neurons in this column should have a minimal standard deviation. Thus, it is appropriate to add the sum of all neurons in this column to the excitation term at xi position, where x indicates the corresponding kth plane and i is the index to the column which corresponds to the most consistent estimates (the column whose standard deviation is minimal). Through similar analysis as in the first layer, the weight connection matrix and the bias input are derived from (7) as

$$T_{xi,yj} = -A\left[1-\delta_{xi,yj}\right] - B$$
$$I_{xi} = B + C\left[\sum_l V'_{li}\right]_x \tag{8}$$

As can be seen, the weight connection matrices for both layers are fixed and independent of image sequence frames. Thus, once the weight matrices are set, they can be used for the entire tracking task. In addition, the constant A, B, and C for both layers are also insensitive to the image sequence frames. Motion estimation is carried out by neuron evaluation. Each neuron asynchronously evaluates its state and readjust itself according to the sigmoid function in [8]. The network proposed here calculates the motion based on each individual row of the image. Thus, the size of the network representing the row is relatively small. This increases the convergent probability of the network. In addition, the output of neurons is not sensitive to their initial states. The simulation results shown in the next section support the claims.

4. Simulation Results and Conclusion

To simulate the network, the differential equation shown below is used.

$$\tau\frac{dU_i}{dt} + U_i = \sum_j T_{ij}V_j + I_i \tag{9}$$

Equation (9) was numerically integrated using the simple Euler method. The constant values are A=500, B=200, C=50, and ε=0.03 for both layers. A binary image used contains 9 different objects whose motion directions and actual displacements are given in Table 1. Figure 2 illustrates the two successive frames of the image. Notice that small portions of object (8) and (9) are occluded. The occlusion is allowed in order to test the

robustness of the network. A tracking window of size 10×9 was used. That is, there are 10 HT networks per plane and 9 planes in the first layer. The maximum row and column displacements were limited to 4 pixels. Consequently, there are 81 neurons in the second layer.

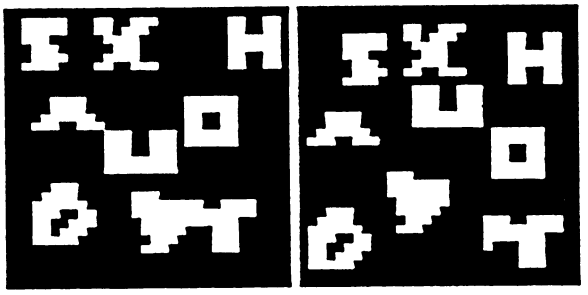


Figure 2 The First and The Second Image Frames

Object Number	Moving Direction	Actual Displacement	Estimated Displacement
(1)	SE	(3,2)	(3,2)
(2)	SE	(2,1)	(2,1)
(3)	SW	(-1,2)	(-1,2)
(4)	SW	(-2,2)	(-2,2)
(5)	NE	(2,-4)	(2,-4)
(6)	SE	(2,4)	(2,4)
(7)	SW	(-2,3)	(-2,3)
(8)	NW	(-2,-2)	(-2,-2)
(9)	SE	(2,2)	(2,2)

Table 1 Simulation Results

As shown in Table 1, the estimated displacements for every object are very accurate despite of occlusion for object (8) and (9). During the simulation, we also found that the network seems to converge rapidly (less than 50 iterations) and also that it converges to the correct displacement every time regardless of initial conditions. In addition, the convergence seems to be less sensitive to the network constants. However, for a large displacement or a large percentage of occlusion, the network either converges to an incorrect state or does not converge at all. For a larger value of constant C (100 or larger), the network seems to converge faster, but sometimes it converges into an incorrect state. We believe that, for a large value of C, the objective term overwhelms the energy function, thereby not assuring a valid result.

Acknowledgements This research was sponsored by the Strategic Defense Initiative Organization through the Air Force Armament Laboratory and the US Army Strategic Defense Command.

REFERENCES

[1] Narathong, C., Iñigo, R.M., McVey, E.S., and Doner, J.F., "A Target Tracking Algorithm Using A Sensor with Biological Vision Features," invited paper, Int'l Journal of Mach. tools and Manuf. Special Supplement on Robotics and AI, Vol.28, No.3, 1987.

[2] Narathong, C., Iñigo, R.M., McVey, E.S., and Doner, J.F., "Real-Time Algorithm and Architecture for Motion Prediction," Proc. IEEE Conf. on Comp. Vision and Pattern Recognition, Ann Arbor, MI, June 1988.

[3] Iñigo, R.M. et al., "A Biological-Structure Visual Sensor for Robotics Applications," Proc. of the 16th Int'l Symp. on Industrial Robotics, Brussels, Belgium, September 1986.

[4] Grzywacz, N.M., and Yuille, A.L., "Massively Parallel Implementations of Theories for Apparent Motion," Tech rep AI Memo No. 888, CBIP Memo No.016, MIT Lab. and Cen.for Biol. Info. Proc., June 1987.

[5] Koch, C., "Analog Neuronal Networks for Real-Time Vision Systems," in Proc. Workshop on Neural Network Devices and Applications, Los Angeles, CA, February 1987.

[6] Hutchinson, J., Koch, C., Luo, J., and Mead, C., "Computing Motion Using Analog and Binary Resistive Networks," IEEE Computer Magazine, 52-63, March 1988.

[7] Zhou, Y.T., and Chellappa, R., "Computation of Optical Flow Using A Neural Net," Proc. of IEEE Int'l Conf. on Neural Network, Vol.II, San Diego, CA, July 24-27, 1988.

[8] Hopfield, J.J., and Tank, D.W., "Neural Computation of Decisions in Optimization Problems," Biological Cybernetics, Vol.52, pp.141-152, 1985.

An Artificial Neural Network Approach for Solving Autonomous Navagation Control Problems

Oleg Jakubowicz and Robert Spina
Department of Electrical and Computer Engineering
238 Bell Hall
Buffalo, New York 14260

ABSTRACT

An artificial neural network is used to model the characteristics of a system containing multiple input-output mappings in a dynamically changing environment. Biological properties for learning and data representation have been used wherever possible. These include topology preserving mappings, logarithmic input representation, and localized sensory data. The network learns to achieve a goal based on the current state of the system. The network is capable of producing the desired control signals even though it must deal with delayed performance evaluation, incomplete input specification and non-linear functions.

INTRODUCTION

One of the problems with traditional approaches for modeling a system is an incomplete input-output specification. For a system of any reasonable size, it is impractical or impossible to provide complete mappings of the operating environment. By using an extension of Kohonen's self- organizing feature maps [1] and proper input representation, a more robust system can be developed. The system is capable of achieving its goal even with random trajectory starting points in previously untrained environments. The network outlined in this project uses a centralized view of its environment. The environmental information is obtained as if the sensors were mounted in the robot. This is important since precise fixed frame of reference information could be difficult to obtain during run time processing as is the case in real animals.

THE AUTONOMOUS AGENT PROBLEM

The goal selected for this project was the control of a "space-lander" or isomorphically an autonomous undersea vehicle. A one dimensional system of this type was outlined in [2]. The goal of the network is to safely land a space craft in a two dimensional space. Safely in this case means with a minimum vertical velocity and within a minimum distance of a randomly selected base location. Mountains having random location and height are placed on the surface of the planet to complicate the landing. One possible starting environment is shown in Figure 1. The network is to select the proper value for four thrusters which determine movement in two directions.

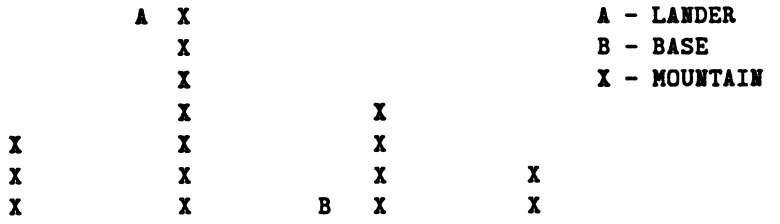


Figure 1 - Initial Environment

One pair of thrusters control the vertical path of the craft and the other control the horizontal path. These thrusters can range in value from 0 to 9. The gravity of the planet and the variation in the mass of the craft due to fuel consumption are implicitly accounted for in the simulation.

INPUT REPRESENTATION

There are eight input variables to the network. They are : Altitude, Distance to Base, Angle to Base, Distance to Mountain between craft and base, Angle to Mountain Top between craft and base, Magnitude of Velocity, Angle of Velocity, Distance to Closest Mountain in current path. A distributed value-type representation of the input was constructed. Figure 2 shows an example with altitude equal to 135 (max was 250).

Example : ALTITUDE = 135 TOTAL RANGE [0..250]

0	0	0	.3	.7	1	.7	.3	0	0
1	2	3	4	5	6	7	8	9	10

Figure 2 - Smeared Input Representation

In order to have better resolution on the critical variables, such as Distance to Mountain, a logarithmic distributed representation is used. In the current implementation Altitude, Distance to Base, Distance to Mountain between craft and base and Distance to Mountain in current path are all represented logarithmically. Each of the distributed input variable vectors are concatenated together to form one large vector that is 80 cells long.

OUTPUT MAPPING

Two self-organizing 15x15 Kohonen networks are used in a hidden layer of the model. One for coarse control and one for fine control. Only one of the networks operates for any given input. The coarse control is used if the craft is greater than a threshold distance from the base, and the fine network is used if the craft is less than the threshold distance from the base. The learning of the network and the modification of the thruster maps is similar to the rule used by Ritter [3] to learn Visuomotor-Coordination. The error-correction rules are

$$W_r^{new} = W_r^{old} + ETA(X - W_r^{old}) \quad (1)$$

$$O_r^{new} = O_r^{old} + ETA(T - O_r^{old}) \quad (2)$$

where W = weights, X = input vector, O = output value, and T = desired output. The r subscript denotes the winning neuron. Both the network and the thruster maps have their neighborhoods adjusted to preserve topology. The simulation program is run by a human teacher (i.e. pilot) and the events of the flight are recorded including pilot selected thrusts.

One objective of the system is to provide for robust operation with minimum training. Although the exact number of training vectors needed can not be determined, an incremental approach can lead to an acceptable solution with a minimum number of vectors. Training may start with five successful landings and then testing performed. If the behavior of the system is inadequate retraining with 10 landings can take place and so on until the desired result is achieved. Twenty successful landings generate approximately 500 unique input vectors for this project.

An interpolation process is used for inputs that do not match a winner well. Since the sample thrust map is topologically order even with only a few training trajectories linear interpolation is advantageous. A sample thrust map showing topographical ordering can be seen in Figure 3.

2	0	2	2	2	5	4	2	3	-1	-1	0	3	3	2
0	2	2	2	1	3	3	6	3	-1	-9	1	4	3	2
0	1	3	3	3	2	0	2	0	-3	-4	-1	4	3	3
0	0	3	3	3	2	2	3	4	0	-5	-1	1	1	1
3	4	3	3	3	3	3	3	4	5	0	-3	2	-1	0
7	5	2	4	4	3	3	4	4	3	2	-1	-1	-1	0
5	4	3	8	4	3	9	6	2	2	2	1	2	1	1
5	5	3	3	2	2	6	3	3	1	2	2	2	2	2
3	4	3	2	2	2	3	2	0	0	3	2	2	2	2
3	3	5	3	2	3	2	1	1	0	3	2	2	2	2
2	3	2	3	2	2	3	2	2	2	2	1	3	2	4
3	2	3	2	2	9	2	1	0	2	2	3	2	9	3
2	3	2	2	3	2	2	1	2	4	3	2	3	3	3
3	3	2	1	2	2	2	0	1	2	2	3	2	2	0
3	3	3	5	2	2	3	1	0	2	2	3	1	2	1

Figure 3 - Sample Thrust map.

RESULTS

Currently a 100 percent success rate has not been achieved. The network's average error on training exemplars is less than 0.5 of a thruster value. Some trainings produce as little as 0.1 average error. These values are dependant on the number of unique input vectors and the artificial decay rate ETA. The lander with coarse and fine networks and logarithmic input mapping has been able to achieve a 50 percent success rate. This is with totally random initial conditions including altitude, velocity, mountains and base location. A successful landing is one where the craft impacts the surface of the planet with less than 15 units of velocity and within 20 units of the base. There were no successful landings without the coarse and fine networks and modification of thrust based on input distance from the exemplar. It should be noted that even in the unsuccessful landings the network is never behaving eratically. The thrusts selected appear to be appropriate, but simply not adequate. It is believed that this can be improved with greater input resolution and a larger network. A slower decay in ETA may also improve performance.

CONCLUSIONS

There seems to be promise in the use of artificial neural networks for black box control problems. Any system with arbitrary input-output mappings can be learned with the architecture outlined. There is a direct improvement in the topology of the output map with an increase in the magnitude and radius of the input magnitude of distributedness. The distributed representation outlined in Figure 2 produces an average distance of adjacent neurons of approximately 0.2 (maximum distance = 1.0). The average distance without any smearing is about 0.5. The method outlined holds promise for systems with multiple outputs based on the same inputs. Each output need only have its own output variable map in parallel with the networks output map. This reduces computation time to a simple look up for each desired output.

The system outlined here is complex with multiple dependacies. Its solution should provide insight into the development of systems for real applications.

REFERENCES

[1] T. Kohonen, "Self-Organized Formation of Topologically Correct Feature Maps", Biological Cybernetics, 43:59-69.

[2] S. Suddarth, S. Sutton, and A. Holden, "A Symbolic-Neural Method for Solving Control Problems".

[3] H. Ritter, T. Martinetz, and K. Schulten, "Topology- conserving Maps for Learning Visuomotor- Coordination".

Visual Discrimination of Multi-Spectral Signals

Oleg G. Jakubowicz*
State University of New York at Buffalo
Department of Electrical and Computer Engineering
238 Bell Hall
Buffalo, New York 14260

August 1, 1989

Abstract

A multi-layer neural network model is presented which is able to distinguish distorted multi-spectral signals in noisey environments. Discrimination is by signal shape recognition and not by Fourier transforms. Therefore in contrast to most other signal recognition models recognition is by visual means rather than auditory. The model is a simulation of the visual retino-cortex system and employs localized receptive fields, lateral inhibition utilization in both learning and recognition and multiple resolution. Localized inputs and lateral inhibition incurs special problems in learning and these are described.

Introduction

Many times signals are distinguished by visual means as opposed to auditory means. Often times isolated features are sought for and are more distinguishable visually because of the segmentation and localization capabilities which are highly developed in the vision system. Multi-layered localized receptive field neural models of which Fukushima's Neocognitron [1] is most notorious offer a high potential for excellent visual recognition. For this reason we have developed and utilized such a model for multi-spectral signal recognition. The network has previously been used in the recognition of patterns of spatially related objects and has been described in Jakubowicz [2]. Since that version we had increased our recognition capabilites by modifying the learning rule. This learning rule

*Sponsored by the Air Force Office of Scientific Research/AFSC, United States Air Force, under contract F49620-87-R-0004.

requires a system of localized lateral inhibition and incremental activation via differential equations. In simulation the model is capable of recognizing 18 distinct multiple-spectrum signals. Since the model has been described in the earlier paper [2] we will present only a summary of the network architecture here and then procede to describe the lateral inhibition.

The Multi-Level Multi-Feature Localized Receptive Field Architecture

The network architecture consists of 5 levels of neurons which have localized 5x5 receptive fields. Each level consists of 100 two dimensional planes, each of which contains neurons sensitive to only one feature. The input plane consists of three feature planes, one for each sensor. In the simulation to be described the input has 289 pixels arranged in a 17x17 grid. One dimension represents time and the other signal amplitude. There is an approximately two to one increase in receptive field size as the signals propagate up the levels which means the number of neurons at any level is approximately cut in half at the next higher level.

Each level contains two layers of planes. A US plane integrates inputs from all planes in the preceding level while a UC plane merely performs a spatial-temporal integration (spatial-spatial in vision). In some instantiations of the model the third level only contains 25 planes or features. The top level contains only 18 nodes which are taught with a teacher to identify complete input patterns: this is the recognition level. Learning of all levels other than the top one procedes via the usual Kohonen self-organizing feature map methods. A diagram of the architecture is presented in figure 1.

Lateral Inhibition in Localized Receptive Fields

A major distinction of natural vision systems is the profusion of localized receptive fields and localized lateral inhibition. The ranges of each neuron's local receptive field and independently their local lateral inhibition always overlap with those of many other eighboring neurons. At each retinotopic location there is a column of many neurons each of which is sensitive to a different input feature. Biologically synapse growth is commonly believed to occur by either chosing a winner within the columns or by using a non linear Hebb related law where the stronger I/O correlations produce a much greater change in the weights and the weaker produce virtually no change. However since neurons at an adjacent spatial locations (same hypercolumn according to the Hubel and Weisel conventions) have close to the same receptive fields and there are thousands of neurons in a hypercolumn many neurons will be feature detectors for identical features which are merely displaced by one or two cones (when projected backwards to

the retina). This results in an extreme inefficiency of natural resources which is contrary to nature in systems with limited resources. This has been the reason that ANS modelers choose only one *winner* in a hypercolumn instead allowing one in each column (see Fukushima's Neocognitron or Jakubowicz's Visually related Situational Analysis System). Each of these two systems has rule-based approximate methods for choosing a *winner*. However the Neocognitron method for selecting *winners* doesn't work for analog inputs or images with regions of varying luminance and the Situation Analysis selection process still has a finite degree of redundancy.

Therefore we've considered a closer look at biological systems and propose a more biologically accurate solution. The network's performance is not limited by analog signals or images with non constant regions of luminance as in the Fukushima hypercolumn model, nor by partial redundancy of feature detectors as in the symmetry selection rule. Instead we have good capability in analog images and in regions with lower output or luminance. This obliterates the problem of redundancy in the form of features which are merely translated in the window.

The resulting system consists of localized inhibitory fields which tend to produce only one winner in a spatial neighborhood and feature column.

Simulation Example

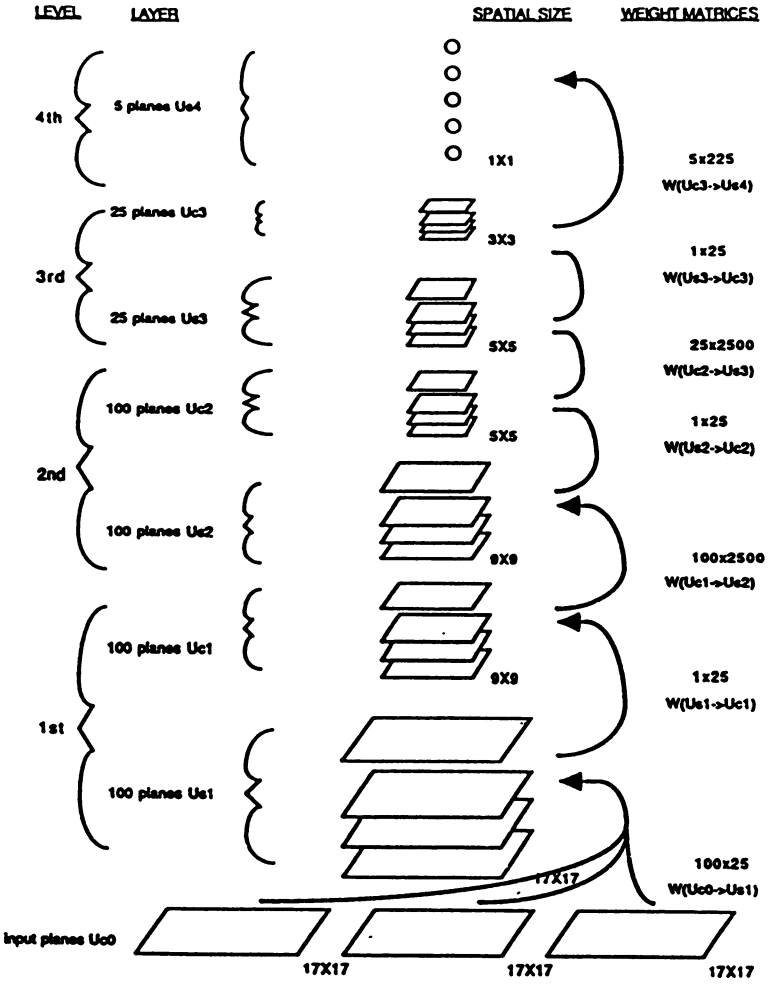
We have simulated the architecture described above with incremental activation differential equations, lateral inhibition and the learning rule referred above on a SUN 4 computer. Eighteen signals were taught to the system, each of which had inputs from three different types of sensors (e.g. microwave, infrared and ultraviolet). All eighteen were recognized correctly, even when slightly distorted.

In the top line of figure 2 we illustrate the infrared components of four of the signals. The microwave and ultraviolet components were of comparable complexity. In the next four lines of figure 1 we present the outputs of four of the eighteen output nodes in the form of histograms. Proper identification of each of the signals is evident from the histograms. The outputs of the other fourteen nodes have small or zero activation values and were not illustrated here.

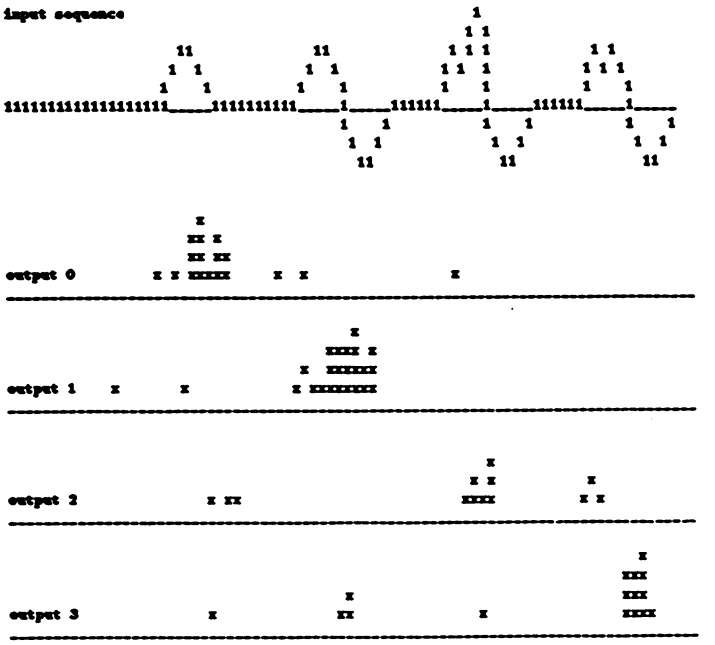
References

[1] Fukushima, K. *A Neural Architecture for Selective Attention in Visual Pattern Recognition*,Biological Cybernetics, 55, 5-15, 1986.

[2] Jakubowicz, O., *Multi-Layer Multi-Feature Map Architecture for Situational Analysis*, Proceedings of IEEE International Joint Conference on Neural Networks, June 1989.



Architecture of the multi-layer multi-feature network of neurons with localized lateral inhibition and receptive fields.
Figure 1.



Histogram of activation values for four of the 16 output units in response to the signal presented in the top line. The time dimension is along the abcissa and activity values are along the ordinant. Each 'x' represents 25% of maximum possible activation.
Figure 2.

Superresolving Neural Network for Deconvolution

Peter A. Jansson

Engineering Physics Laboratory
E. I. Du Pont de Nemours and Company (Inc.)
P.O. Box 80357
Wilmington, Delaware 19880-0357

Results of spreading and blurring phenomena encountered in science, engineering and analysis may be described by the convolution of two functions. One of these functions characterizes an original object or time series as it would be observed in the absence of spreading. The other describes the spreading process. Frequently, one wishes to recover the original object, given only the spread function plus the image or smeared data . We show how this may be accomplished by processing the image with a multilayer, feed-forward higher-order neural network containing sigmoid nonlinearities. Because of physical realizability constraints imposed through nonlinear processing, this network is capable of restoring frequencies in the image that have been completely obliterated by a band limiting spread function. Although the principle of this network[1] was shown in 1968, the relationship to the connectionist viewpoint has not been previously demonstrated.

Convolution and Deconvolution

For illustration, let the object of interest be a function of one independent variable, a hypothetical spectrum o(x) as it would be observed by a perfectly resolving spectrometer, and let s(x) be the spread function, where x is the independent variable of measurement such as wavelength or wave number. The image or observed spectrum is then described by the convolution integral

$$i(x) = \int s(x-x')\, o(x')\, dx'. \qquad (1)$$

A large literature[2] exists on the extraction of o(x) given i(x) and s(x). The classic solution is given by inverse filtering. In one expression of inverse filtering, the data i(x) are convolved with a function y(x) to obtain an estimate o(x). The optimal filter is the Wiener inverse filter.[2] No linear restoration method can yield a better result than this when judged by the minimum mean-square-error criterion. The Wiener inverse technique can be applied to time-series data by employing a simple tapped delay line or shift register, and to images with coherent optics.

Deconvolution employing such linear methods has enjoyed only limited success. Its difficulties are legendary. One of them is extreme sensitivity to noise. Small contaminants in i(x) give rise to great uncertainty in o(x). Spurious solution components appear that are not possible in the physical context represented by the data. Furthermore, the blurring process is often band limiting: frequencies beyond a certain cutoff are removed from the data. Good results require restoration of the lost frequencies. Although at first glance this seems to be impossible, it has now been clearly demonstrated with a number of nonlinear techniques[3] that employ additional *a priori* knowledge. It was the effective use of such *a priori* knowledge, beginning with the principle employed in the present research, that has widely extended the practical application of deconvolution.[4]

A Nonlinear Solution

Equation (1) may be sampled and given the discrete representation

$$i_n = \sum_m s_{nm}\, o_m. \qquad (2)$$

It has been shown[1,2] that superior solutions for o(x) may be obtained by making use of the knowledge that solutions must be physically realizable. For absorption spectra, o(x) may only assume values between absorptance bounds of zero and one. Such physically-bounded results are achieved by modifying a traditional linear iterative method used to solve the above set of equations. In the traditional method (successive overrelaxation), estimates of the object o(x) are given by

$$\hat{o}_n^{(k+1)} = \hat{o}_n^{(k)} + \frac{\kappa}{s_{nn}} \left\{ i_n - \sum_{m<n} s_{nm}\, \hat{o}_m^{(k+1)} - \sum_{m \geq n} s_{nm}\, \hat{o}_m^{(k)} \right\}. \quad (3)$$

Here, κ is a constant coefficient that weights the correction terms applied to the $\hat{o}_n^{(k)}$ with each iteration. The first estimate $\hat{o}_n^{(1)}$ is usually taken to be i_n for n over the range of the data. In the original research, it was observed that samples of the solution in the physical region near the bound needed relatively little correction, and that samples in the middle of the physical region would benefit from more adjustment. Samples in the unphysical region would require correction with sign opposite to that given by the linear method, in which unphysical components grow with successive iterations. Superior restorations with no unphysical components were obtained by replacing the constant weight κ by a function of $\hat{o}_n^{(k+1)}$ in order to meet these requirements. The nonlinear function chosen was $\kappa[\hat{o}_n^{(k)}] = \kappa_0 (1 - 2 \mid \hat{o}_n^{(k)} - 1/2 \mid)$ and is illustrated in Fig. (1).

The Neural Network

The deconvolution process may be implemented in either analog or digital hardware.[5] Input time-series data are clocked into an image-data shift register from the right side as illustrated by the top row of triangular symbols in Fig.(2). Convolution (symbolically "$\circledast$") by spread function s(x) is implemented by applying weights represented by connections shown below the shift register. For purposes of illustration, the trivially simple case of a three-sample spread function is illustrated. Also note that a true discrete convolution is illustrated for simplicity (*simultaneous* overrelaxation). The summations in Eq.(3) employ contributions from both the $(k)^{th}$ iteration samples and the $(k+1)^{th}$ iteration samples (*successive* overrelaxation). With either simultaneous or successive methods, a linear PE (processing element), shown where the connections from the shift register converge, forms the sum and passes it to a sigma-pi unit.[6] At the sigma-pi unit, the sum is multiplied by a signal from the shift register that has passed through a PE having the nonlinear transfer characteristic $\kappa[\hat{o}_n^k]$. The product is summed in the sigma-pi PE with another signal from the input data shift register. Resultant values transfer to the left in a new shift register with each clock pulse. In this new shift register, the values are available for the convolution required by the stage below, which represents the next iteration of Eq.(3). The design of the rest of the neural network follows this example, one shift-register layer corresponding to each iteration required.

In the implementation described, there is one sigma-pi unit and one shift register per layer. Alternatively, the top row of Fig.(2) could represent a row of linear buffers operating in parallel and receiving parallel data. Beneath them would be a layer of linear summers, a layer of nonlinear PE's, a layer of sigma-pi units, and so forth. Although we have thus far considered one-dimensional data, the analysis is readily extended to higher dimensionality. In two dimensions, for example, we draw the analogy with biological retina. The first layer of linear summing PE's yielding $i - s \circledast \hat{o}(x)$ is indeed like the visual opponent-response receptive field. Thus, even in this first stage some sharpening of $\hat{o}(x)$ occurs. It is interesting to speculate on the degree to which the biological analogy could be carried to higher stages of vision. Humans *do* have the capacity to infer the underlying sharp content of blurred images.

It is also noteworthy that the nonlinearity employed may be expressed as the first derivative of a sigmoid-shaped function of spliced parabolas (Fig.(3)). Specifically, we may write

$\kappa(\hat{o}) = d\sigma(\hat{o})/d\hat{o}$, where $\sigma(\hat{o}) = \kappa_0[\hat{o}^2]$ for $\hat{o} < 1/2$, and $\sigma(\hat{o}) = -\kappa_0[\hat{o}^2 - 2\hat{o} + 1/2]$ for $\hat{o} > 1/2$.

Over part of its range, $\sigma(\delta)$ is qualitatively similar to exponential-based sigmoids[6] used in back propagation and other network paradigms. However, it loses its monotonicity in the region representing unphysical solutions. This characteristic is responsible for its effectiveness in enforcing the bounds and may be useful in other neural-net paradigms.

Simulation

The network was simulated on an Apple MacIntosh II computer. An absorption spectrum object containing two pairs of closely spaced lines (Fig.(4)) was synthesized. The absorption coefficient profile was chosen to be Gaussian for each of the lines. The large peak absorption coefficients of the leftmost two lines gave rise to considerable saturation and some peak flatness when the absorption coefficient was converted to the illustrated absorptance via exponentiation. The spectrum was then blurred by band limiting its Hartley[7] transform with a triangular transfer function. Frequencies beyond the cutoff were set to zero.

Because simulations of even inferior methods with noise-free "data" can give unrealistically good deconvolution solutions, uncorrelated gaussian-distributed random noise of 0.25% of full absorptance range rms was added to the object. Results of a seventeen-layer simulation are illustrated in Fig.(4). Polynomial smoothing[2] convolution layers were employed between the first fifteen layers described above. They were omitted from Fig.(2) for purposes of clarity.

Results illustrated in Fig.(4) show clear resolution of both pairs of lines. The steepness of the rising Gaussian profile near the baseline is well represented. Also shown in this figure are the Hartley transforms of the simulated data, the restoration, and the original synthesized spectrum. These transforms show restoration of some frequencies lost in band limiting. Naturally, the high frequencies farther from the band limit exhibit increasing error. Even restoration of lost frequencies close to the band limit is notable, however, in view of severe attenuation by the triangular transfer function in this region.

Conclusions and Future Research

We conclude that it is possible to perform deconvolution by use of a higher-order neural net architecture, and that physical realizability constraints implicitly contain sufficient information, when used in conjunction with information present in the data, to restore some of the frequencies absent from the data. We also conclude that control of the shape of sigmoids and other nonlinear transfer characteristics employed in the network can be used to enforce constraints on the outputs. While it has been known for some time that bandwidth extrapolation is possible when using constraints in conjunction with other methods, it has not been widely appreciated that the first of the successful constrained methods owes its success to the restoration of lost frequencies.

All the weights in the network were specified in advance. Whereas no learning method was employed in this research, it is clear that the prespecified weights may be regarded as initial values of weights in a network to be trained via back propagation or by other means.

References

[1]Jansson, P.A. (1968). Ph.D. Dissertation, Florida State Univ., Tallahassee.

[2]Jansson, P.A. (1984). "Deconvolution: With Applications in Spectroscopy." Academic Press, New York.

[3]See papers by Y.Biraud, J.R.Fienup, B.R.Frieden, S.J.Howard, J.L.Harris, R.W.Gerchberg, A. Papoulis, H. Stark, D.Youla, and others cited in Ref.1 above, pp.132-134.

[4]Blass, W.E., and Halsey,G.W. (1981). "Deconvolution of Absorption Spectra." Academic Press, New York.

[5]See above Ref.1, p.110.

[6]Rummelhart, D.E., and McClelland, J.L. (1986). "Parallel Distributed Processing," Volume 1, Foundations. MIT Press, Cambridge, Massachusetts.

[7]Bracewell, R.N. (1986). "The Hartley Transform." Oxford University Press, New York.

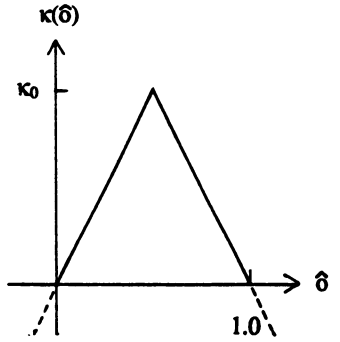


Fig. 1. First derivative of sigmoid.

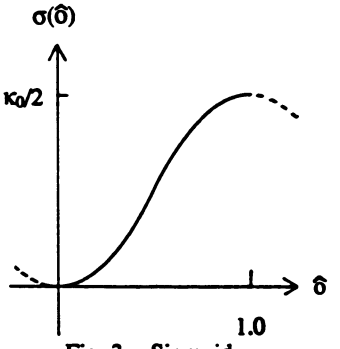


Fig. 3. Sigmoid.

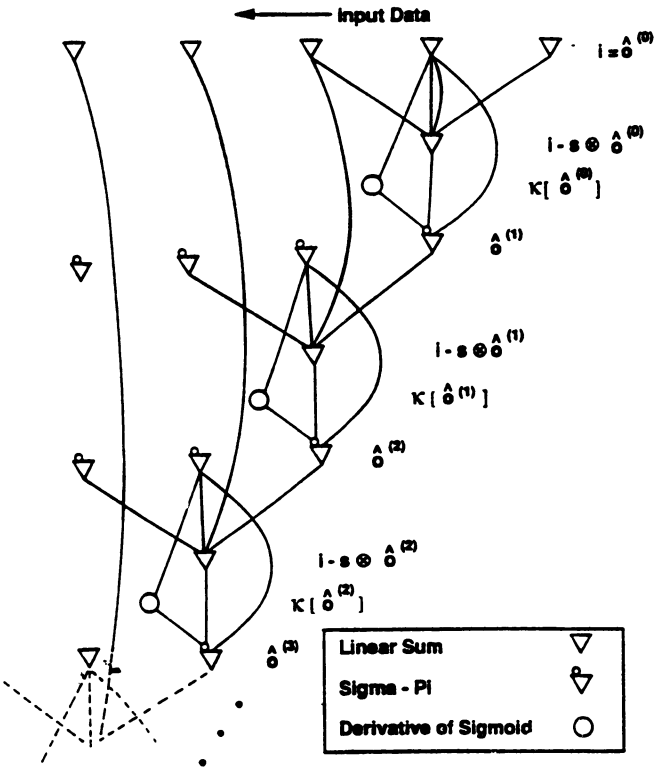


Fig. 2. Supperresolving neural network -- simplified representation.

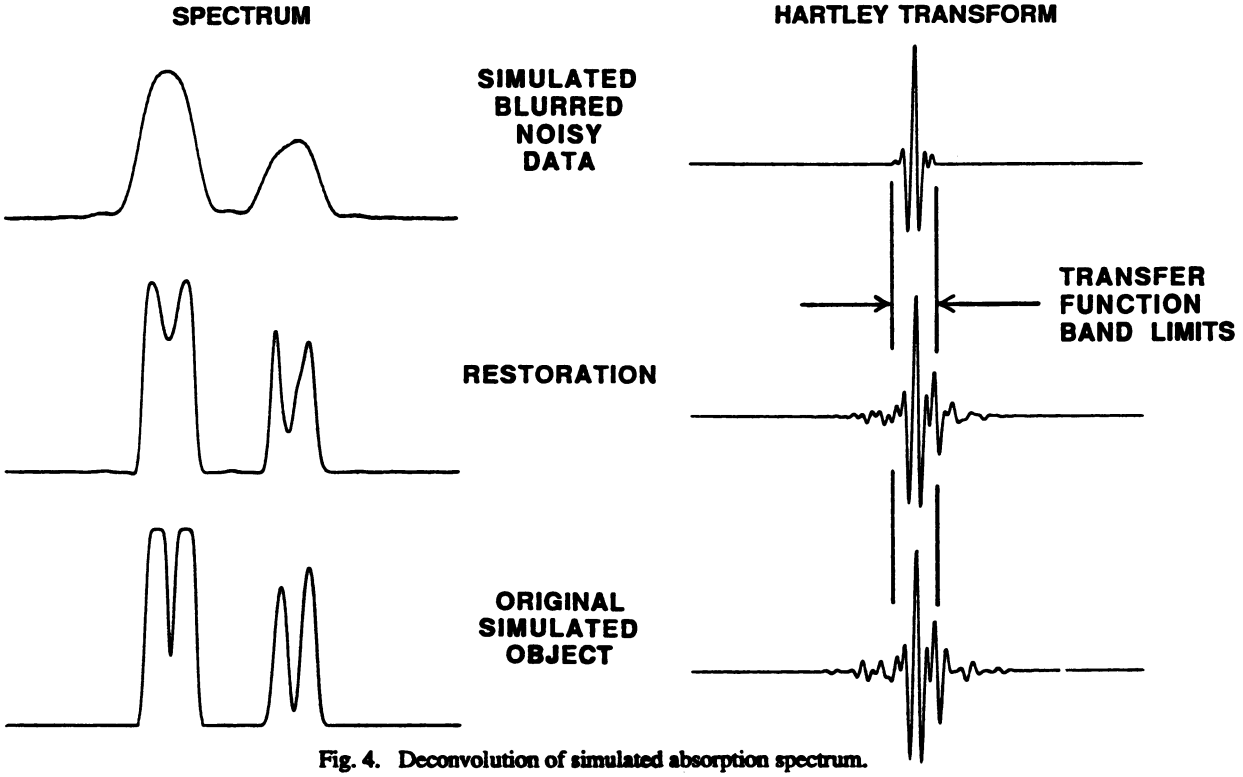


Fig. 4. Deconvolution of simulated absorption spectrum.

Design of a saccadic motion generator that learns

J.D. Johnson and T.A. Grogan Department of Electrical and Computer Engineering
M.L. 30 University of Cincinnati Cincinnati, Ohio 45221-0030

1.0 Abstract

This paper will report on current work on the design of a neural network based artificial vision system. The system, called VisSac, utilizes neurons capable of real-time learning mechanisms and classical conditioning phenomena to learn sequences of human-like eye saccades necessary for the recognition of objects.

2.0 The VisSac System

2.1 Architecture. The VisSac system[1, 2] (Figure 1), simulates the human early vision system which utilizes eye saccades that foveate those image features important for recognition. The VisSac system moves a small (in comparison to the input image size) window about the input image. These saccade-like movements of the window (fovea) are to positions the system has learned are sufficient to distinguish the image from all other images it has learned. The Low Level Feature Extractor (LLFE) is responsible for classification of the input set based on low resolution features. The Innate Saccade Motion Generator (ISMG) is responsible for generating saccade sequences in untrained VisSac systems. The Learned Saccade Motion Generator (LSMG) is responsible for learning the correct saccade sequences necessary to differentiate all input images. The Recognition Net is responsible for correlating the saccade information and input image identities. The Critic rewards or punishes the system based on whether recognition is successful or unsuccessful.

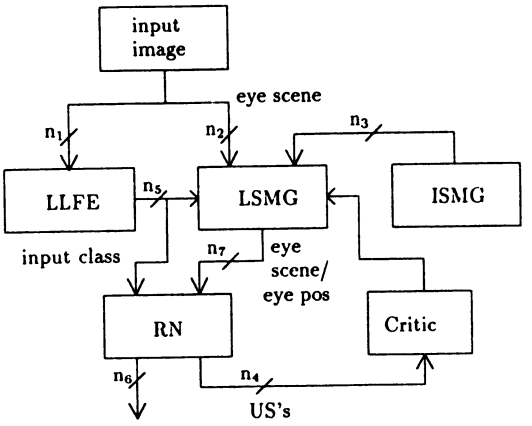


Figure 1. The VisSac System

2.2 The Low Level Feature Extractor. The LLFE is responsible for the clustering of the input image set into subsets of images based on common features. The output of the LLFE, the Class Indicator Signal, will be used by both the Learned Saccadic Motion Generator

(LSMG) and the Recognition Net (RN) in the development of a unique saccade sequence for each image subset capable of distinguishing all the images of that subset.

2.3 The Innate Saccadic Motion Generator. The ISMG generates saccade sequences that are the basis for learning by the LSMG. These saccade sequences are random in nature and the neurons governing these saccades have fixed weights, thus eliminating the likelihood that a specific saccade pattern will begin to dominate the ISMG output. It is important that all saccade positions have the same likelihood of generation by the ISMG, otherwise, those saccade positions that are not generated will never be candidates for the final saccade sequence learned by the system.

2.4 The Learned Saccadic Motion Generator. The LSMG is responsible for learning the correct saccade sequence necessary to identify the individual elements of the input set. The input to the LSMG is the Class Indicator Signal from the LLFE and the PUNISH and REWARD signals from the Critic. The output of the LSMG drives the motor neurons controlling the saccades. The LSMG is composed of several banks of neurons. The Class Indicator Signal selects a unique bank for each class the LLFE has created, and that bank is responsible for learning the saccade sequence necessary for distinguishing all members of that class. Initially the weights from the LSMG to the motor neurons are smaller than the fixed weights from the ISMG to the motor neurons. This results in the ISMG winning the competition between the two generators for control of the saccade sequence. However since the weights of the LSMG can grow, the ISMG looses control of the saccade generation after sufficient training. During training, the values of some of the weights from the LSMG to the motor neurons will be increased through positive reinforcement while others will be decreased through negative reinforcement. Positive reinforcement (REWARD) occurs if recognition is successful and strengthens the dominance of that saccade sequence. Negative reinforcement (PUNISH) occurs if recognition is not successful and weakens the weights of the winning saccade sequence allowing a new saccade sequence to gain control of the motor neurons.

2.5 The Recognition Net. The RN is responsible for recognition of the input image based on the information received from the LLFE and LSMG. By the nature of the VisSac system the RN must be capable of recognizing a temporal sequence of inputs.

2.6 The Critic. The last component of the VisSac system is the Critic. The Critic judges the response of the RN and determines whether or not it has successfully identified the input. The Critic rewards the LSMG strengthening the saccade sequence responsible for the successful recognition and increasing the likelihood of that saccade sequence gaining dominance over other sequences. Punishment of the VisSac system occurs if two input images generate an identical RN output. In such a case, the positions of the saccades are not sufficient to extract the distinguishing features of the input images. Punishment of VisSac alters the LSMG, suppressing that saccade sequence and allowing another sequence to emerge and be tested.

3.0 Three-from-Six Saccade LSMG

A Learned Saccadic Motion Generator was designed capable of learning a three sequence saccade (Figure 2). The LSMG uses a slight modification of the Drive-Reinforcement Neuronal model developed by Dr. A. Harry Klopf[3]. The LSMG design (Figure 2) reserves

one neuron for each possible saccade position. The inputs to all neurons are the Punish and Reward USs, as well as one CS for each possible saccade position. Each neuron has an inhibitory US described below, as well as an input from the ISMG. A MAX operator is used at the output of the neurons. The MAX operator passes, unchanged, only the strongest neural output, all other outputs are zero. Equation 1, which governs the learning of the Klopf neuron, is modified to include the MAX operator. For the i^{th} weight of the k^{th} neuron:

$$\Delta w_{ki}(t) = \Delta y_k(t)\Delta z_k(t)\sum_{j=1}^{\tau} c_j \mid w_{ki}(t-j) \mid \Delta x_{ki}(t-j) \qquad (1)$$

With the inclusion of the $\Delta z_k(t)$ term, weights will only be modified if the positive change in the input resulted in that neuron having the strongest output $y_k(t)$, and therefore, a nonzero $z_k(t)$. (The $\Delta z_k(t)$ helps alleviate the credit assignment problem.) To prevent multiple saccades to the same location during any one sequence, an inhibitory US has been supplied to each neuron. This US is driven by an R/S flip flop, the input of which is that neuron's z(t) output of the MAX operator.

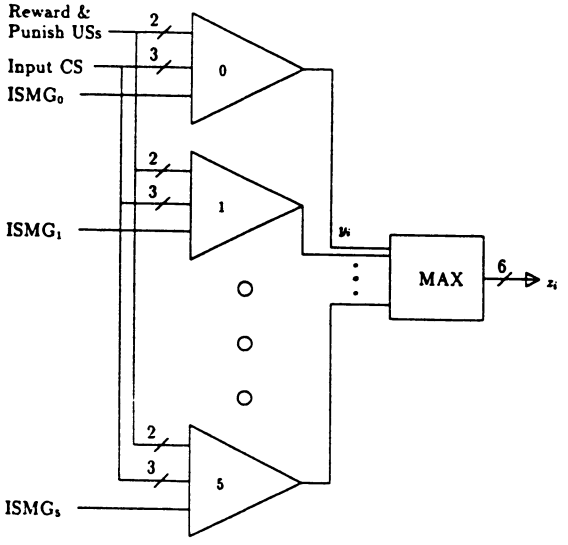


Figure 2. The Three-from-Six Saccade LSMG

The goal of the experiment was to determine if this design could learn a saccade sequence consisting of specific saccade positions. Three (3) saccade positions were arbitrarily chosen from six (6) possible positions as necessary for distinguishing all input images in a given (ISMG determined) class. These positions are 1,3, and 4, though order is not important. The inputs to the LSMG were artificially created, but designed to correspond to the inputs the LSMG would receive from the other components of the VisSac system. After every three saccade sequence the LSMG was either rewarded if the saccade positions it generated matched the desired positions or punished if there were any discrepancies. The CS input to the neurons is the output of a 1-of-6 encoder, where each saccade in the sequence has a unique code. Since saccade position, as well as scene information, is encoded into the

CS, this uniqueness is guaranteed as long as a saccade position is not repeated in the same sequence.

In Figure 3 the results of the experiment are shown. Each tick mark along the time axis corresponds to a Punish or Reward signal and signifies the end of a saccade sequence. The LSMG eventually learns and remembers the saccade sequence 3-1-4. The time taken to learn the sequence can be altered by strengthening or weakening the influence of the Punish and Reward USs.

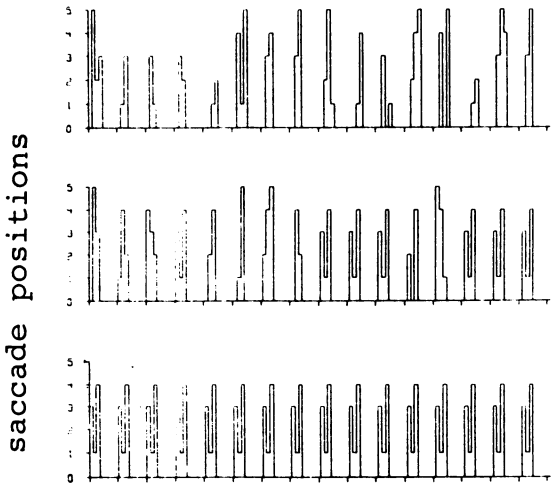


Figure 3. Saccade Positions Selected by LSMG

4.0 Further Research

Further modification of the Klopf neuronal model is necessary to prevent weight value runaway, due to continuous rewarding. Research into the other components of the VisSac system is also underway.

5.0 REFERENCES

1. J. D. Johnson and T. A. Grogan, "Neural network controlled visual saccades," in *Image Understanding and the Man-Machine Interface II* (E. B. Barrett and J. J. Pearson, eds.), (Los Angeles, California), pp. 44–49, SPIE/SPSE, January 1989.

2. J. D. Johnson and T. A. Grogan, "An artificial neural system for controlling visual saccades," Whirlpool Fellowship Interim Report, Department of Electrical and Computer Engineering, University of Cincinnati, February 1989.

3. A. H. Klopf, "A neuronal model of classical conditioning," Interim Report AFWAL-TR-87-1139, Air Force Wright Avionics Laboratory, Wright-Patterson Air Force Base, October 1987.

A MULTILEVEL NEURAL ARCHITECTURE FOR ROBOT DYNAMIC CONTROL

A.KHOUKHI

Télécom Paris ENST Dépt Réseaux 46 rue Barrault 75634 Paris cedex 13.

and

ESIEE Dépt Automatique Cité Descartes 2 Bd Blaise Pascal 92634. Noisy-le-Grand. France.

Summary:

In this paper we consider the problem of the design, trajectory planning and on-line control of robotic systems. A new approach based on the neural networks architecture is developed. This approach operates at three levels:

First, the optimal design of the mechanical structure. The geometric and kinematic parameters such as the length of the links and the maximum and minimum of the joints are optimised. The optimisation process is accomplished according to a new function giving the maximum mobility of the robot. To do so, a six layers network is simulated, corresponding to the desired manipulator (six degrees of freedom).

Second at the learning level where the trajectory planning problem is solved by determining the motor commands from a goal of the movement and the optimisation of a time-energy creteria.This is represented by a weighted multiobjective nonlinear function. A modified backpropagation gradient algorithm is developed and simulated.

The third level is that of the on-line and feedback control of the robot .

Special architectures are described in the literature (1), and it is shown that these controllers are more efficient than the traditional controllers such as the linear and nonlinear control schemes based on the decoupling or adaptive control theory.(2).

The main contribution of the feedback approach proposed in this paper is however, in taking into account of the environment evolution and associated constraints.The neural network developed for the feedback control and the on-line tracking is composed of ten layers, six,6 correspond to each degree of freedom and the last fourth are designed such that each direction of the end effector is associated with a layer. The results of experiments repported consider only the kinematic caracteristics of the environment. An extention to dynamic evolution of the environment is now being pereformed and specially is applied to a multi-robot control system.

The neural networks developped are simulted on a Vax workstation and first results are promissing in perfomances and time computation and show the efficiency of this control scheme compared with the traditional control schemes.

References:
1) J.S.Albus: A new approch to manipulator control: The Cerebellar model articulation controllers(CMAC). J.Dyn.Sys.Meas.,Contr. 97:1975.
2) W.T.Miller, F.H.Glanz, L.G.Kraft: Application of a general learning algorithm to the control of robotic manipulators. Int.Jou.Rob.Res.Vol 6 N°2 Summer 1987.

MSK Signal Noise Estimation Using a Hopfield Neural Network

G. J. Klein
The Johns Hopkins University Applied Physics Laboratory
Laurel, MD 20707

Neural networks are attracting much attention because of their ability to perform complex pattern recognition tasks. This paper discusses a Hopfield neural network used to estimate the probability amplitude distribution of a noise source which is interfering with a communications signal. Accurate estimation of a noise source is useful when one wishes to cancel the effects of the noise upon a desired signal. The estimation problem is formulated such that the interference amplitude distribution is viewed as a pattern distorted by a known Minimum Shift Keying (MSK) communications signal. Available to an observer is a distorted pattern, the received signal amplitude distribution. The network estimates the original pattern (i.e., the interference distribution) based upon the distorted received pattern (i.e.,the received signal) and knowledge of the distorting function (i.e.,the MSK signal).

General Formulation

The following signal model is used in the development of the Hopfield network. Assume that the observable environment consists of two narrowband signals; one a known communications signal, and the other an unknown interference. These signals can be represented in vector notation on an in-phase and quadrature plane, denoting the communications signal by a magnitude, z, and a phase angle, Θ_Z. For the specific MSK example, the amplitude of z is a constant and the information is carried by the phase of the vector. The interference signal similarly is described by a magnitude, x, and an angle, Θ_X. These two signal vectors combine to produce a received signal, with magnitude, y, and an angle, Θ_Y. It is assumed that the interference and the communications signal are not phase locked so that the difference between the phases is uniformly distributed between 0 and 2π. Furthermore, the distribution of the interference signal is arbitrary. Using this signal description, the goal for the neural network is to estimate the amplitude distribution, f(x) of x, given only an estimate of f(y), the amplitude distribution of the received signal, y.

It is convenient to use a discrete-valued amplitude distribution in the form of a histogram. We define the value of the i^{th} histogram bin as

$$f(y_i) = \int_{y_i-\frac{\Delta y}{2}}^{y_i+\frac{\Delta y}{2}} f(y)\,dy \tag{1}$$

where Δy is the width of a histogram bin. For any given y_i, we can use Bayes' law and the law of total probability to obtain $f(y_i) = \sum f(y_i|y_j)\, f(x_i)$. To simplify notation, we define $y_i = f(y_i)$, $x_j = f(x_j)$, and $h_{ij} = f(y_i|x_j)$ so that in vector notation, $\mathbf{Y} = \mathbf{HX}$. Note that with this notation, x is a pdf value and the subscript, i, is representative of the signal magnitude. For the problem at hand, $\mathbf{H}$ is known and $\mathbf{Y}$ can be estimated from an observed histogram of the received signal. When the communications signal is an MSK signal, $\mathbf{H}$ can be expressed as

$$h_{ij} = \frac{1}{N}\sum_{j=0}^{N-1} f\left(y \in y_i | x_j\right); \quad x_j = \left(x_i - \frac{\Delta x}{2}\right) + \frac{(\Delta x)\,j}{2N} \tag{2}$$

where

$$f(y \in y_i|x) = \int_{y_{i_1}=y_i-\frac{\Delta y}{2}}^{y_{i_2}=y_i+\frac{\Delta y}{2}} f(y|x)\,dy =$$

$$\begin{cases} \frac{1}{\pi}\left[\cos^{-1}\left(\frac{y_{i_1}^2 - (x^2+z^2)}{2xz}\right) - \cos^{-1}\left(\frac{y_{i_2}^2 - (x^2+z^2)}{2xz}\right)\right] & : y_{i_1}, y_{i_2} \in \kappa \\ \frac{1}{\pi}\left[\cos^{-1}\left(\frac{y_{i_1}^2 - (x^2+z^2)}{2xz}\right)\right] & : y_{i_1} \in \kappa,\ y_{i_2} \notin \kappa \\ \frac{1}{\pi}\left[1 - \cos^{-1}\left(\frac{y_{i_2}^2 - (x^2+y^2)}{2xz}\right)\right] & : y_{i_1} \notin \kappa\ y_{i_2} \in \kappa \\ 0 & : y_{i_1}, y_{i_2} \notin \kappa \end{cases} \tag{3}$$

In this form, **H** is invertible. Thus, it would appear that the the technique for estimating **X** is trivial: simply multiply the received vector by $\mathbf{H}^{-1}$. However, **Y** is only an *estimate* of the received signal amplitude distribution. Therefore, the estimation errors which exist make this proposed solution fail. When simulated, it has been found that the estimated **X** has negative values or values greater than 1, and is usually not at all like the actual vector.

Zhou, et al. [1] have developed a model for image reconstruction using a Hopfield network which can be adapted to solve the noise estimation problem. Using a similar formulation, we define the estimation as an energy minimization problem, where the energy function is defined as

$$E = -.5 \| \mathbf{Y} - \mathbf{HX} \|^2 - .5 \beta \| 1 - \mathbf{HX1} \|^2 \tag{4}$$

and where 1 is the column vector of all ones. Here, the first term represents the error between the estimated and received signal distribution and the second term represents the error in which the estimated received signal has fulfilled a constraint that the sum of its terms equal one. By iteratively changing our estimate, **X** until both terms are minimized, assuming that the estimation error in **Y** is not too great, we arrive at a good estimate of the interference amplitude distribution.

In order to use a Hopfield network for this problem, the energy function of (4) must be related to that defined in Hopfield's original formulation [2]

$$E = -0.5 \sum_i \sum_j T_{ij} v_i v_j - \sum_j I_j v_j \tag{5}$$

Writing out (4) term by term gives

$$\begin{aligned} E = {} & 0.5 \sum_p y_p^2 - \sum_p \sum_i h_{pi} y_p x_i + 0.5 \sum_p \sum_i \sum_j h_{pi} h_{pj} x_i x_j \\ & + 0.5 \beta \left(1 - 2 \sum_p \sum_i h_{pi} x_i + \sum_p \sum_q \sum_i \sum_j h_{pi} h_{qj} x_i x_j \right) \end{aligned} \tag{6}$$

In (6), the first line represents the first term in (4) which is a function of the difference in the received and estimated received signal. The second line represents the second term which constrains the sum of histogram values to 1. Comparing the terms in (6) and (5), and letting the output of the i^{th} neuron, v_i, represent the value of the i^{th} histogram bin, x_i, the weights of the Hopfield network are

$$T_{ik;jl} = - \sum_p h_{pi} h_{pj} - 2 \beta \sum_p \sum_q h_{pi} h_{qj} \tag{7}$$

and the biases are given by

$$I_{ik} = \sum_p h_{pi} y_p + 2 M \beta \sum_p h_{pi} \tag{8}$$

At this point we stop and make a few observations. First, it is noted that in the original Hopfield network as described in [2], inter-neuron weights are symmetric and self feedback weights to neurons are equal to zero. Second, the update rule used in [2] is a binary threshold rule which outputs a 1 if the weighted sum at a neuron is greater than 0, and 0 if the weighted sum is less than 0. By using such weights and update rules, the network is guaranteed to decrease in energy until it converges to a locally minimal energy state. For the noise estimation application, it is seen that the weights satisfy the Hopfield requirement of symmetry; however, they do require non-zero self-feedback. Additionally, the outputs for this application must be able to take on all values between zero and one, so a simple binary threshold will not suffice. Therefore, it is desired to formulate an update rule where the outputs are constrained to lie between 0 and 1, and which will allow the network energy to decrease to a minimum as the network iterates.

Hopfield Network Implementation

In order to use a continuous valued network which decreases in energy upon iteration, we use the following formulation. As in Hopfield's original network, the output rule is a function of weighted sum of inputs. Denoted at the i^{th} neuron as net_i, this sum is given by

$$net_i = \sum_j T_{ij} v_j + I_i \tag{9}$$

To define an update rule which will allow the network to decrease in energy, we perform gradient descent upon (5); that is,

$$\frac{\partial E}{\partial v_i} = \sum_j T_{ij} v_j + I_j = net_i \tag{10}$$

From (10), it is seen that in order to use an energy function in the form of (5), the current value of v_i must be incremented by an amount proportional to net_i. At first look, one might guess that a non-linear output function

is not needed to form a network which satisfies the aforementioned requirement. However, it has already been demonstrated [3] that networks consisting of linear elements can carry out only a very limited set of functions. Furthermore, it is desired that in the continuous formulation that the output range of a neuron lie between 0 and 1. This would enforce the constraint that any histogram bin must take values between 0 and 1. Keeping these ideas in mind, we define the following output update rule

$$v_i^t = f(u_i^t) = \begin{cases} \dfrac{2}{1+\exp(u_i^t)} - 1 ; & u_i^t > 0 \\ 0 ; & u_i^t \le 0 \end{cases} \tag{11}$$

where u_i^t is given by

$$u_i^t = u_i^{t-1} + \alpha\ net_i^t \tag{12}$$

Here f( ) is a half sigmoid function whose argument is proportional to the previous value of the neuron, $u_i^{t-1} = f^{-1}(v_i^{t-1})$, and of the current net_i^t. Using the update rule of (11), the continuous valued network is guaranteed to decrease in energy as defined in (5) and the output of each neuron is constrained to values between 0 and 1. Just like in the original Hopfield formulation, the neuron outputs are bounded, and in turn, the energy function is bounded. Therefore, for symmetric weights where $T_{ii} = 0$, the network is guaranteed to converge to an energy minimum. For the case where $T_{ii} \neq 0$, it can be shown that energy will not increase as long as $net_i \Delta v > 0.5 T_{ii}(\Delta v)^2$. Thus, as long as α is made small enough to make v change slowly, the network will generally converge.

The neural network described for this application is implemented as follows. Assuming a signal amplitude distribution is to be divided into L bins and each bin is represented by the output of one neuron, L neurons are required. Inter-neuron weights and biases are determined using (7) and (8). Initially, the output of each neuron in the network is randomly chosen to be between 0 and 1. Once set, the network is allowed to iterate. Each neuron is visited sequentially and the weighted input to that neuron, net_i is computed. The neuron is then updated according to (11) and (12). This process is allowed to continue until a steady state condition is reached. In the original Hopfield formulation, the visiting process was randomly selected. However, in simulations, little difference was seen between cases where the ordering of neuron update was deterministic or was random.

Continuous Results

We now apply this network to the MSK problem. Values for the estimated received distribution, **Y** were obtained via a Monte Carlo simulation in the following manner. An interference with an assumed amplitude distribution, **X**, was added with an MSK vector of fixed amplitude and random phase. For each Monte Carlo trial, the amplitude of the received signal was recorded. After a number of trials, a histogram of the received values was calculated to form **Y**. Once the values in **Y** were estimated, they were used with the values from **H** to compute the weights and biases of the neural network. In the computation of the weights and biases, the value of ß, which weights the importance of the second term in the energy function, was set to zero. Therefore, the energy minimization affected only the first term of (4).

In this example, the histogram is divided into 32 bins. Figure 1 shows the value of the actual interference distributions. It is noted that the received distribution, **Y**, includes estimation error, i.e., this distribution does not exactly equal the interference distribution times **H**. Once the biases and weights were set, the network was initialized with random outputs and allowed to iterate according to (11) and (12). In (12), α was set to 0.5. Result of the simulation after 50 iterations is seen in Figure 2, where one iteration is defined as the update of every neuron once. These plots show that the network produced an accurate estimate of the interference distribution.

Subsequent trials using the same **H** but different values of ß between 0 and 1 to calculate the network weights and biases were carried out to determine whether the second term in the energy function affected the final result greatly. These trials produced solutions which did not differ significantly from those presented. This could be expected since it is unlikely that the first term of the energy equation of (4) could be satisfied if the second term was not. Therefore, the second term represented redundant information.

In all of these trials, it was seen that the network produced a good, but not perfect, estimate of the interference distribution. Errors in the estimate are present because of error in the estimate of the received signal amplitude distribution. Additionally, errors could be caused in a Hopfield network if the network settled to a local

instead of a global minimum. In an effort to investigate whether the network was converging to local minima, the simulations were run for a number of times starting with different initial random output values. In all cases, the final equilibrium point was the same. Though this certainly does not prove that the equilibrium point reached is a global energy minimum, it does seem to indicate that the energy surface approximates a convex cup surface. That is, the surface most likely is cup shaped and has only one local minimum which is the global minimum.

Summary

A Hopfield network has been designed to estimate a noise amplitude distribution which is interfering with an MSK communications signal. The network obtains the estimate essentially by beginning with a random distribution and then by iterating the histogram bin values such that an error function is minimized. The paper describes a continuous output network, much like Hopfield's original network formulation in [2], to carry out this task. It was shown that for the specific MSK application, the energy surface for this network most likely is not vulnerable to errors due to local energy minima.

References

1. J.J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proc. Natl. Acad. Sci.* USA 79, 1982, 2554-2558.
2. Y. Zhou, et al., "Image Restoration using a Neural Network." *IEEE Trans. on ASSP* 36, 1988, 1141-1151.
3. M. Minsky and S. Papert, *Perceptrons.* MA:MIT Press, 1969.

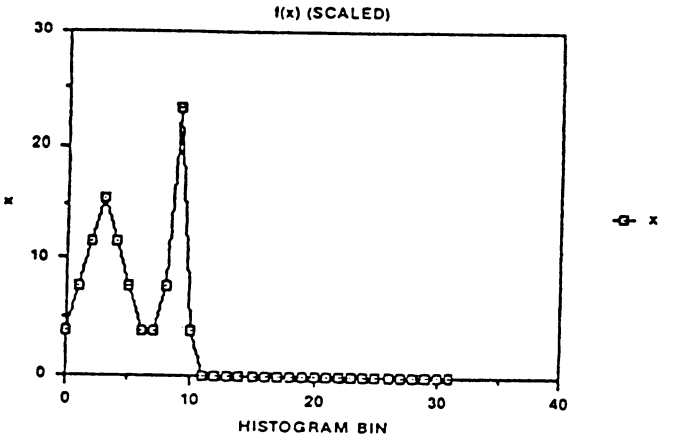


Figure 1 Actual Interference Amplitude Distribution

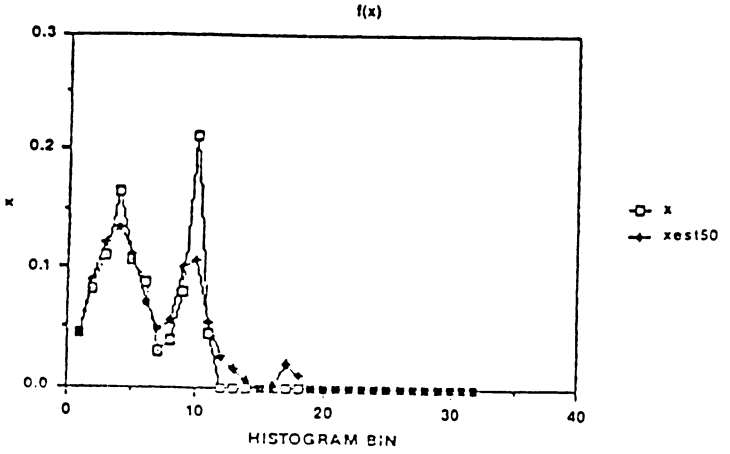


Figure 2 Estimated Interference Amplitude Distribution

Psychophysical Experiments and Computer Simulations of the Binocular Rivalry

Tetsuo Kobayashi

Dept. of Applied Electronics, Hokkaido Institute of Technology
419-2 Teinemaeda, nishi-ku, Sapporo 006, JAPAN

1. INTRODUCTION

Although the binocular rivalry has been studied(Levelt, 1966, Fox, 1975, & Grossberg 1989), the mechanism of the binocular rivalry is not clear. The present work describes the testable features of the alternation process in binocular rivalry examined by psychophysical experiments. The stimulus strength to the central part of the visual field in one eye was varied while the stimulus strength in the other eye remained constant. To explain the experimental results, a model which consists of a group of neural elements is proposed and simulated on the computer. Each neural element generate time-dependent post synaptic potentials by the impulses input at random around a certain repetition rate.

2. BINOCULAR RIVALRY

Binocular rivalry is a process that visual stimuli from right and left eyes are recognized alternately. Total observation duration of the binocular rivalry is

$$T=\sum_{i=1}^{n_L} d_{Li}+\sum_{i=1}^{n_R} d_{Ri} \qquad (1)$$

where n_L and n_R are numbers of the dominance durations for left and right eyes, d_{Li} and d_{Ri} are each duration of the binocular rivalry for left and right, respectively.

The means of the dominance durations for left and right eyes are

$$\overline{d_L}=\sum_{i=1}^{n_L} d_{Li} / n_L \quad , \qquad \overline{d_R}=\sum_{i=1}^{n_R} d_{Ri} / n_R \qquad (2)$$

And variances are

$$u_L=\sum_{i=1}^{n_L} (d_{Li}-\overline{d_L})^2/(n_L-1), \qquad u_R=\sum_{i=1}^{n_R} (d_{Ri}-\overline{d_R})^2/(n_R-1) \qquad (3)$$

Predominance for Left = $\overline{d_L}/(\overline{d_L}+\overline{d_R})$, and for Right = $\overline{d_R}/(\overline{d_L}+\overline{d_R})$ (4)

3. EXPERIMENTS

One horizontal line and one vertical line (Both lines are 0.45°wide, white, 10cd/m^2) on the black screen(possible stimulus size is a 29.86° x 19.54° rectangle) are given to the left and right eyes separately. The observer perceives the horizontal line and the vertical line alternately at the intersection of two lines. To examine the relation by the change of the stimulus strength of the central vision of the right eye, the vertical line is separated by the visual angle d° from the center of the line. During the experiment, the observer continues to press the switch at the duration the left stimulus is felt dominant. The observers were eight 22-year old males. Formal data collection consisted of ten 70 sec observation periods, each separated by 30 sec rests. The last 60 sec of each observation period were used in the data analysis; the first 10 sec were considered a warm-up phase. The frequency histograms for dominance duration in the case d=0.9° and the theoretically generated gamma distribution for left and right are shown in Figs.1 and 2, respectively. The mean and variance of the dominance duration obtained by Eqs. (2) and (3) are shown at the upper right of the each figures. The chi-square value of the fitness to the gamma distribution is also shown at each figure (Significant if it is smaller than 36.42 ($P<0.05$) and 42.98 ($P<0.01$)). Figure 3 shows the dependence of the mean for the duration in respect to the angle d. The same results are obtained when we replaced the stimulus figures to right and left eyes. After the same experiments which had done for the other observers, following two testable results on the binocular rivalry could be deduced.

1) The decrease in stimulus strength to the central vision in one eye while the stimulus strength in the other eye remained constant decreases the mean dominance duration, variance and predominance in the same eye and increases them in the other eye.

2) The frequency histogram of the dominance duration almost fits to the gamma distribution.

4. A MODEL OF THE BINOCULAR RIVALRY

To explain the experimental results, we propose a model shown in Fig. 4. The model consists of a group of neural elements(NEs) located at the same receptive field of the perception level. Each NE receives impulse trains at the input synaptic connections. Impulse generates the excitatory post synaptic potential(EPSP) and the inhibitory post synaptic potential(IPSP) at the excitatory connection and the inhibitory connection, respectively. EPSP generated by the integration of the n_j impulses at d excitatory connections and IPSP generated by the integration of the m_j impulses at g inhibitory connections become

$$Ve(t)=\sum_{j=1}^{d} Pej \sum_{k=1}^{n_j} \exp[-\alpha_{ej}(t-t_{jk})] \qquad (5)$$

$$Vi(t)=\sum_{j=1}^{g} Pij \sum_{k=1}^{m_j} \exp[-\alpha_{ij}(t-t_{jk})] \qquad (6)$$

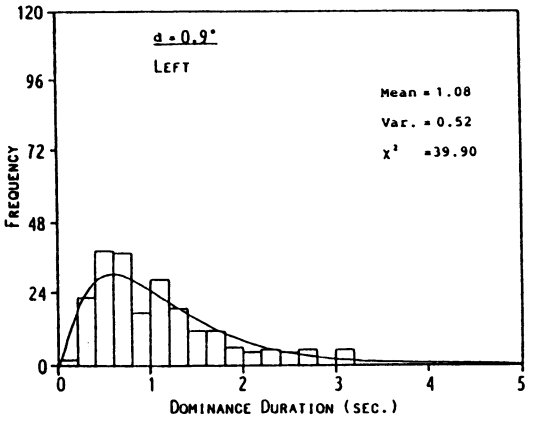


FIGURE 1. Distribution of dominance durations for Left stimulus and density function generated by the gamma distribution in the case d=0.9°.

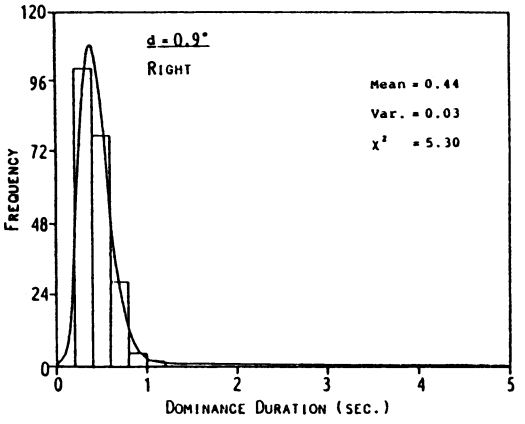


FIGURE 2. Distribution of dominance durations for Right stimulus and density function generated by the gamma distribution in the case d=0.9°.

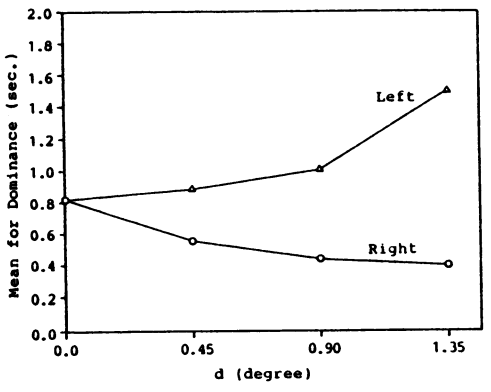


FIGURE 3. The change of mean for dominance durations as a function of stimulus strength in the right eye.

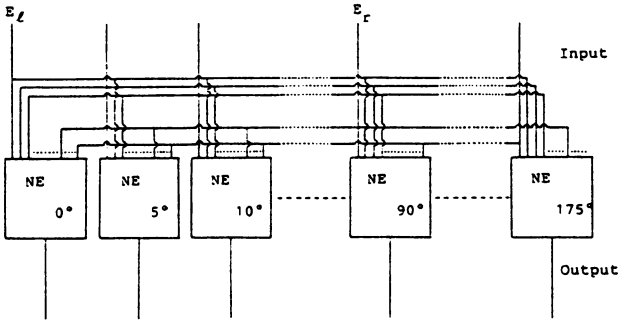


FIGURE 4. A model which explains the binocular rivalry. Each neural element (NE) has an excitatory connection to the pre-level NE which fires for the same orientation and has inhibitory connections to other NEs. E_l and E_r are stimuli from the left and right eyes.

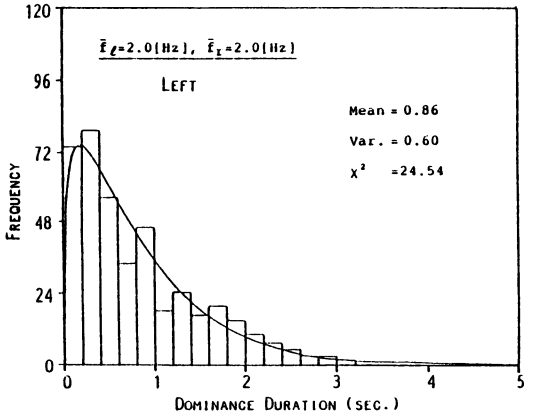


FIGURE 5. Distribution of dominance durations for the left stimulus and density function generated by the gamma distribution in the case $\bar{f}_l$=2.0[Hz], $\bar{f}_r$=2.0[Hz].

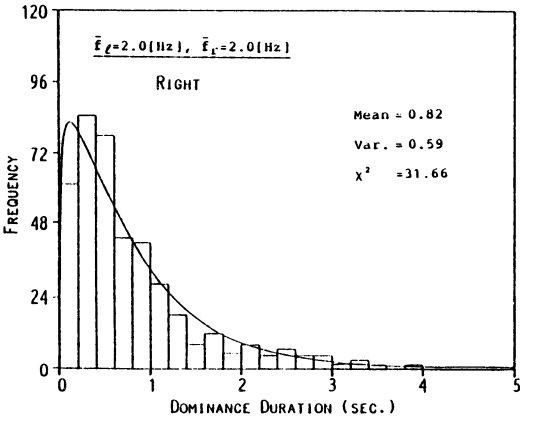


FIGURE 6. Distribution of dominance durations for the right stimulus in the case $\bar{f}_l$=2.0[Hz], $\bar{f}_r$=2.0[Hz].

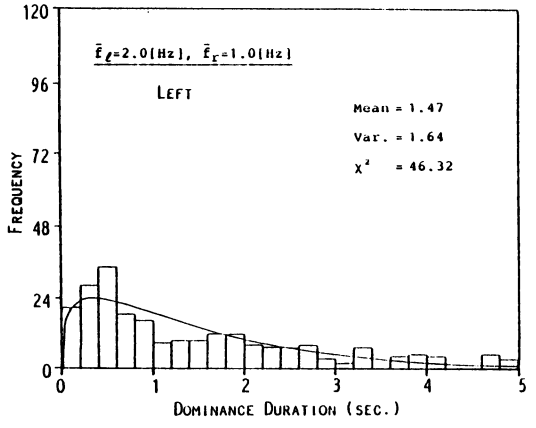


FIGURE 7. Distribution of dominance durations for the left stimulus in the case $\bar{f}_l$=2.0[Hz], $\bar{f}_r$=1.0[Hz].

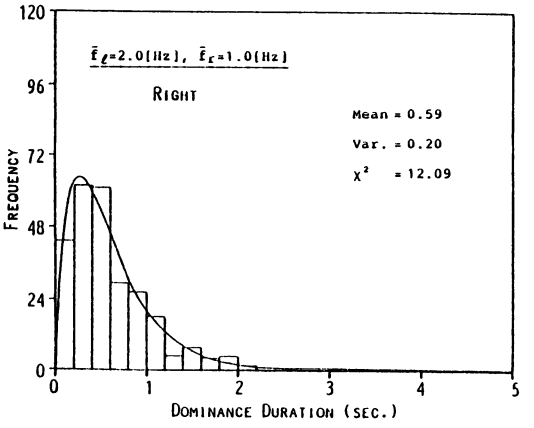


FIGURE 8. Distribution of dominance durations for the right stimulus in the case $\bar{f}_l$=2.0[Hz], $\bar{f}_r$=1.0[Hz].

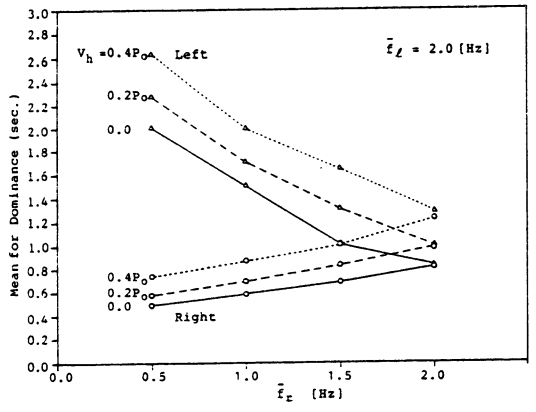


FIGURE 9. The change of mean for dominance durations as a function of $\bar{f}_r$ while $\bar{f}_l$ remained constant in relation to the threshold potential Vh.

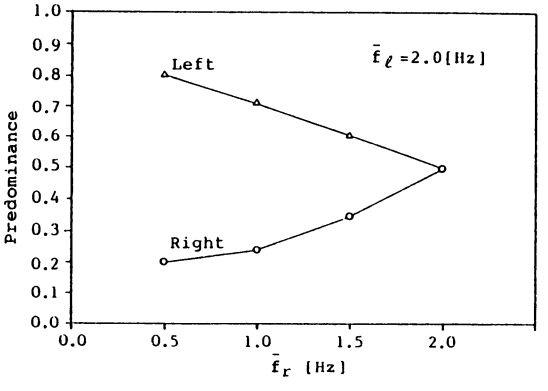


FIGURE 10. The change of predominance as a function of $\bar{f}_r$ while $\bar{f}_l$ remains constant in the case Vh is 0. The predominance hardly depends on the Vh.

where P_{ej}, P_{ij} are initial values of the post synaptic potential(PSP) at the j-th connection of the excitatory and inhibitory connections, respectively. P_{ej} has a positive value and P_{ij} has a negative value. t_{jk} in Eqs. (5) and (6) indicates the time when the k-th impulse input the NE. t_{jk} is written using the average repetition frequency $\bar{f}_j$ of the impulses as $t_{jk}=k/\bar{f}_j$. The PSP V(t) is Ve(t) + Vi(t). The NE is considered fire if V(t) is greater than the threshold potential Vh(t). The model assumes that each NE has the excitatory connection to only for a NE which selectively fires for a certain orientation and has the inhibitory connections for the other NEs. Figure 4 is illustrated as each NE is orientation selective only for every 5° from 0° to 175° to simplify the model. Since it is known that there are neurons which selectively fire in respond only a certain orientation in the primary visual cortex of the monkey(Kuffler et al, 1984), we think it is possible to assume orientation selective NEs as the previous level neurons of the model. To simulate the experiments described before, we assume the stimuli given to the left eye (a horizontal line; direction is 0°) and the right eye (a vertical line; direction is 90°) input signals E_l and E_r to the model as shown in Fig. 4. Since NE0° and NE90° can receive both the excitatory and the inhibitory signals, NE0° and NE90° can fire and the other NEs can't fire. In the case that NE0° and NE90° fire (on) and do not fire (off) together, the NE perceived at the previous moment is treated to continue to be perceived because the dominance of the two NEs can't be determined.

5.SIMULATION

Strength of the stimulus is replaced by the average repetition frequency of impulses. The decrease of the frequency corresponds to the decrease of the stimulus. P_{ej} and P_{ij} in Eqs.(8) and (9) are the same value but have the different sign as $P_{ej} = -P_{ij} = P_0$ (constant). α_{ej} and α_{ij} in Eqs. (8) and (9) are 0.15. Since the stimulus to the left eye was constant in the experiment, the frequency of the impulses is set at the constant frequency 2.0[Hz]. The frequency of the right eye is reduced from 2.0[Hz] to 0.5[Hz]. The impulse is generated by the random value from 0.0 through 1.0 which is generated in every 0.1sec. The obtained frequency histograms for dominance duration and the density function are shown in Fig. 5 - 8. Figs. 5 and 6 show the dominance frequency histograms in the case $\bar{f}_l=\bar{f}_r=2.0$[Hz] which respond that the stimuli from both eyes are equal. Figs. 7 and 8 show the histograms in the case $\bar{f}_l=2.0$[Hz] and $\bar{f}_r=1.0$[Hz]. Figure 9 and 10 show the dependence of the mean for the duration and the predominance to the $\bar{f}_r$, respectively. These results indicate that decrease of the stimulus strength to the right eye while the stimulus strength to the the left eye remains constant decreases the mean for dominance and predominance in the right eye and increases them in the left eye.

6. CONCLUSION

The results obtained by the psychophysical experiments indicate that the variations in stimulus strength alter the mean dominance duration, variance and predominance. And the frequency histogram of the dominance duration almost fits to the gamma distribution. Since simulation results agree with the experimental results, the binocular rivalry can be explained as follows. The binocular rivalry is the process that different two neurons located at the same receptive field of the recognition level in the brain fire alternately by their own integrations of post synaptic potentials. The post synaptic potentials are generated by the impulse trains which repetition rate is changing at random around a certain rate.

REFERENCES

Fox, R., Todd, S. & Bettinger, L. A. (1975). Optokinetic nystagmus as an objective indicator of binocular rivalry. *Vision Res.*, 15, 849-853.

Grossberg, S. & Marshall, J. A. (1989). Stereo boundary fusion by cortical complex cells; A system of maps, filters and feedback networks for multiplexing distributed data. *Neural Networks*, 2, 29-51.

Kuffler, S. W., Nicholls, J. G. & Martin, A. R. (1984). From neuron to brain, 2/e. Sunderland, MA: Sinauer Associates, Inc.

Levelt, W. L. M. (1966). The alternation process in binocular rivalry, *Brit. J. Psychol.*, 57, 225-238.

A Vision Architecture for Scale, Translation and Rotation Invariance †

Mark W. Koch, Morien W. Roberts, and Steven W. Aiken
Clarkson University
Department of Electrical and Computer Engineering
Potsdam, N.Y. 13676

Abstract

In designing vision architectures, we want to avoid training an artificial neural network with every possible object size, orientation and position. High order neural networks invariant to scale, rotation, and translation escape this problem. However, the exponential growth in the number of correlations identifies a serious problem with high order neural networks.

We have developed a receptive field neuron that can learn high order correlations. A receptive field reduces the combinatorial explosion of high order correlations by restricting the interconnection range and by using a subset of high order correlation terms. The back-propagation learning algorithm extracts the high order features that discriminate the objects in the training set.

Introduction

One approach to an artificial neural network's supervised training for vision, is to let a network train on every object that it might encounter. The network could be shown the objects in all possible sizes, locations, and orientations and be taught to make the right classification. This approach has the problem of large learning times, which result from the many possible object configurations plus the many pattern presentations required for the generalized back-propagation learning algorithm.

Some researchers solve these problems by extracting conventional position and orientation invariant features from the image. A neural network then trains with this reduced feature set. Yamada [13] first extracts contour curvature features from an image. He uses this feature vector to recognize handwritten numerals. Troxel [10] recognizes trucks and tanks by preprocessing multifunction laser radar data using a Fourier/log-polar transform. The magnitude of the Fourier transform followed by a log-polar transform first produces a position invariant image and then converts variations in rotation or scale to shifts. The normalize peaks in the Fourier/log-polar space provide the input to a multilayer backpropagation neural network. Brousil [1] has designed a perceptron network to perform the Fourier/log-polar transform.

Unlike the above, our approach lets the vision system determine the best features for object recognition. To solving the training problem we develop neural network architectures that produce output patterns invariant to the viewing perspective. This is done by putting constraints on a network's interconnection strengths.

Fukushima's [2] *Necognitron* uses a hierarchical type of network to produce outputs invariant to position, size, and small distortions of numerical characters. As one moves up the hierarchy, there exists alternating layers of simple/complex cell pairs. The simple cells recognize or combine features of the preceding layer. The complex cells recognize the simple cell features independent of small changes in position. At each level, the cells recognize more complex and larger features with more inherent translational invariance. At the final output layer an active grandmother cell has a one to one correspondence with a unique object in the scene regardless of any translations and small size changes or deformations.

Widrow [12] has developed a neural network architecture using Adaline neurons called the *invariance net* which has a response invariant to position and rotation. Widrow defines a *slab* as a group of random weights to one neuron that receives input from every image pixel. Let N represent the number of all possible translation and 90° rotation combinations. Shifting and rotating the slab in all possible combinations produces N slabs with N corresponding neurons. These N neurons produce a permuted response depending on the object's rotation and translation. A majority element makes this response invariant. Widrow does this for every pixel in the image and derives a position and orientation invariant *scrambled pattern*. The invariance net needs many weights. Widrow also discusses how to extend the network for invariance to scale and perspective. Hosokawa [5] plans to extend Widrow's method by learning the slab weights with back-propagation.

Rumelhart [9] has used a three layer back-propagation network to solve the TC problem, which has a network recognize the letter T or C independent of the letter's orientation or position. The TC problem represents a hard problem with the solution requiring third order correlations [7]. Instead of having every input neuron connect to a hidden node, each hidden node "sees" a subset of the input scene called a *receptive field*. Constraints on the receptive field weights produce an output on the hidden layer consisting of a single feature detector centered and duplicated at every input image pixel. Constraining the weights to the output creates a translation invariant network architecture, but the network still needs to see the objects at every rotation.

Reid [8] and Giles [3] [4] use *high order neural networks* to solve the TC problem and to produce network

† NSF grant No. EET-8806958 supports this work

architectures invariant to scaling, translation, and rotation. A high order neuron has the following form:

$$O_i = F(y_i), \quad \text{and} \quad y_i = W_i + \sum_j W_{ij} X_j + \sum_{jk} W_{ijk} X_j X_k + \ldots,$$

where O_i represents the neuron's output, F the nonlinear activation function, y_i the net input to the neuron, X_i the raw inputs and W_i, W_{ij}, W_{ijk}, ... the 0^{th}, first, second, etc. order interconnection strengths. The 0^{th} and first order correlation typically make up units in a back-propagation network. Second order networks multiply all possible raw input pairs and supply these as inputs to the neuron. These second order correlations make solving problems like the XOR problem trivial and allow the use of simpler learning rules such as the Widrow-Hoff delta rule [11] instead of the generalized delta rule [9] .

Reid uses second order correlations and the Widrow-Hoff delta rule to solve the TC problem for translation and scale invariance. Each second order correlation comes from two input image pixels and forms a line in the input space. By constraining weights corresponding to correlation lines with the same slope to have the same values, a network architecture invariant to translation and scaling results. Likewise third order correlations form triangles in the input space. By constraining weights corresponding to correlation triangles with the same three interior angles to have the same values, invariance to translation, rotation, and scaling results.

Finite resolution makes larger shapes harder to discriminate than the scaled shape versions. For scale invariance, Reid trained the network with the larger shapes, so that it could generalize to the smaller shapes. Giles [3] suggests normalizing the shapes to have the same "energy" regardless of scale. The exponential growth in the number of correlations with order pinpoints a serious problem with high order neural networks.

Receptive Field Neuron

By restricting the interconnection range and using a subset of high order correlation terms one can reduce the combinatorial explosion and still have a network architecture invariant to scale, rotation, and translation. The question remains of which subset to use. We propose the use of a more powerful neuron called the *receptive field neuron* that allows a backpropagation network to learn which high order features discriminate the objects in the training set.

Figure 1 shows the receptive field neuron, r, receiving its inputs from a group of pixels, X_i, forming a circle in the image plane. The W_i's represents the weights to the receptive field neuron and θ its bias. The following equation gives the receptive field neuron's activation, A_r:

$$A_r = \sum_{i=0}^{p-1} \frac{1}{p} logistic(s_i), \quad \text{and} \quad s_i = \sum_{j=0}^{p-1} X_j W_{(j+i)(mod\ p)} + \theta.$$

Each s_i is internal to the receptive field neuron and essentially looks for a single nonlinear feature at different rotations. The activation of r produces a value proportional to the number of times this nonlinear feature appears in its field.

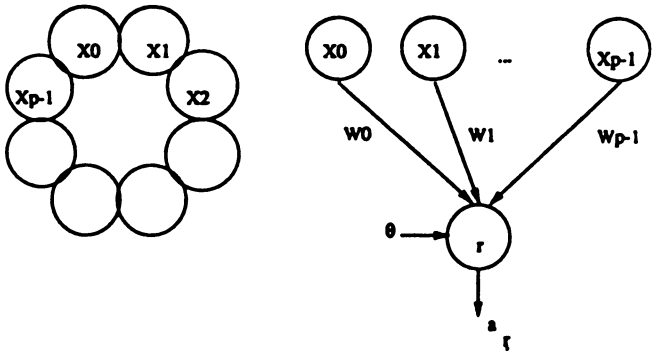


Figure 1. Receptive Field Neuron for Translation and Rotation Invariance.

If we consider the weight vector, $W = [W_0, \ldots, W_{p-1}]$, as a slab, then with Widrow's method the receptive field has invariance to rotation. Widrow's method uses one neuron per slab, where our method

puts the slab groups inside the neuron resulting in a more powerful neuron. Also Widrow uses global invariant features, where here the receptive field represents a local feature. As in Rumelhart's and Reid's solution to the TC problem, by constraining the weights from equivalent receptive field neurons to have the same value we will have translation invariance.

If we assume a binary image with 0 for *on* and 1 for *off*, then the receptive field neuron can compute different high order correlations. Here multiplication of inputs turns into the logical *and* operation which a single neuron can compute. For example with $p = 8$, $W = [4\ 0\ 0\ 4\ 0\ 4\ 0\ 0]$, and $\theta = -10$ the receptive field neuron will compute the *and* of x_0, x_3 and x_5. Using a training algorithm such as back-propagation, a network can learn subsets of high order correlations.

Receptive Field Neuron's Learning Law

We can extend back-propagation's learning law to include the more powerful receptive field neuron. Let δ_r represent the error at the output of the receptive field neuron and η the learning rate. The following equation gives the change in weight, ΔW_i, to reduce the squared error at the output:

$$\Delta W_i = \eta \frac{\delta_r}{p} \sum_{j=0}^{p-1} (1 - s_j) s_j X_{(i-j)(mod\ p)},$$

where X_k, s_j, and p have been defined earlier. Since we can model the bias like a weight with an input of always 1, a similar update rule exists for the bias. Using these learning rules a gradient descent algorithm can compute the change in weights to reproduce a target at the outputs. Here the weights will represent a rotation and translation invariant feature that discriminates between the training set objects.

Scale Invariance using a Receptive Field Neuron

Figure 2 extends the translation and rotation invariance receptive field to include scale invariance. Here concentric circles of pixels form the receptive field, with $X_{k,i}$ representing the pixel at the i^{th} neuron on the k^{th} concentric layer. The following equation gives the activation function for c concentric layers with p pixels per layer:

$$A_r = \sum_{k=0}^{c-1} \sum_{i=0}^{p-1} \frac{1}{c \cdot p} logistic(s_{k,i}), \quad \text{and} \quad s_{k,i} = \sum_{j=0}^{p-1} X_{k,j} W_{k,(j+i)(mod\ p)} + \theta,$$

where $W_{k,i}$ represents the connection weights. As before, each $s_{k,i}$ on the k^{th} layer essentially looks for a single nonlinear feature at different rotations. If we force $W_{k,i} = W_{l,i}$ for $0 \leq k, l \leq c-1$ and $0 \leq i \leq p-1$ then each concentric layer looks for the same feature. The only difference is that the concentric layers scale the features with respect to each other and A_r produces a value proportional to the number of times this nonlinear feature appears in the receptive field.

Like Reid we need to consider the problem of finite resolution making larger shapes harder to discriminate than the scaled shape versions. Reid solved the problem by training with larger objects first where Giles normalized the "energy" of the shapes.

Preliminary Results

Using the software by McClelland and Rumelhart [6] modified for a transputer, we experimented with the TC problem using a single simulated translation and rotation invariant 3 x 3 receptive field neuron centered at every image pixel. The network trained on the T and C at a standard position and orientation. After training, the network could recognize a T or C at any position or orientation. The network learned the following receptive field weights and bias:

$$\theta = -10 \quad \text{and} \quad W = \begin{matrix} 4 & -10 & 6 \\ -3 & & 10 \\ -9 & -2 & -6 \end{matrix}$$

Future Work and Conclusions

We are now parallelizing the back-propagation algorithm for 16 transputers. With the parallelize code we can test our receptive field neuron on more realistic examples with faster learning times.

We have proposed a new scale, translation and rotation invariant receptive field neuron. A group of receptive field neurons can learn a subset of restricted range high order features and will not explode combinatorially like high order neural networks. Using the receptive field neuron, a network needs only to see the object in a standard position and orientation. After learning the objects in the training set the network can recognize objects at any position or orientation. An extension allows the receptive field to include scale invariance.

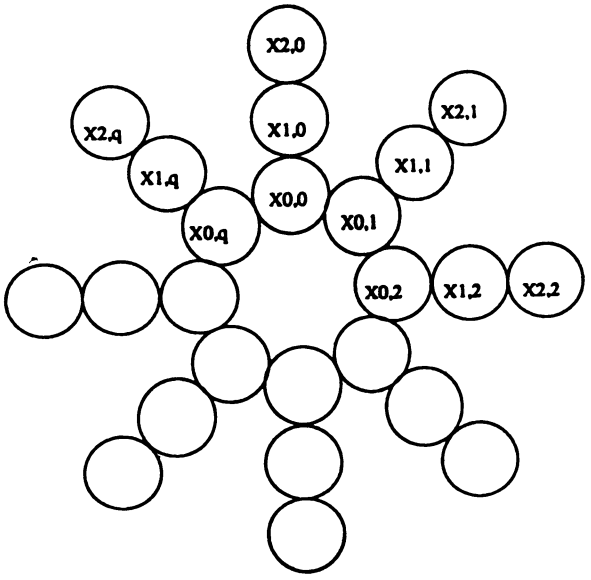


Figure 2. Receptive Field Neuron for Translation, Rotation, and Scale Invariance ($q = p - 1$).

References

[1] Brousil and Smith, "A threshold logic network for shape invariance," *IEEE Transactions on Electronic Computers*, Vol. EC-16, 6, December 1967, pg. 818-28.

[2] Fukushima, K., "Neural network model for selective attention in visual pattern recognition and associative recall," *Applied Optics*, Vol. 26, 23, December 1, 1987, pg. 4985-92.

[3] Giles, C. L., Griffin, R. D., and Maxwell, T., "Encoding geometric invariance in high-order neural networks" *Neural Information Processing Systems*, American Institute of Physics Conference Proceedings, 1988, pg. 301-9.

[4] Giles, C. L., and Maxwell, T., "Learning, invariance, and generalization in high-order neural networks" *Applied Optics*, Vol. 26, 23, December 1, 1987, pg. 4972-78.

[5] Hosokawa, M., Omatu S., and Fukumi, M. "A new approach for pattern recognition by neural networks with scramblers," *International Joint Conference on Neural Networks*, Washington DC, Vol. I, June 18-22, 1989, pg. 183-8.

[6] McClelland, J. L., and Rumelhart, *Explorations in Parallel Distributed Processing* MIT Press, 1986, pp 348-52.

[7] Minsky, M. L. and Papert, S., *Perceptrons*, MIT Press, 1969.

[8] Reid, M. B., Sprikovska, L., and Ochoa, E., "Rapid training of higher-order neural networks for invariant pattern recognition," *International Joint Conference on Neural Networks*, Washington DC, Vol. I, June 18-22, 1989, pg. 689-92.

[9] Rumelhart, D. E., Hinton, G. E., and McClelland, J. L., "Learning internal representation by error propagation" *Parallel Distributed Processing, Vol. 1: Foundations*, MIT Press, 1988.

[10] Troxel, Rogers, and Kabrisky, "The use of neural networks in PSRI target recognition" *IEEE International Conference on Neural Networks*, San Diego California, Volume I, June 24-27, 1988, pg. 593-600.

[11] Widrow, G., and Hoff, M. E., "Adaptive switch circuits," *Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record, Part 4*, 1960, pg. 96-104.

[12] Widrow, and Winter, and Baxter, "Layered neural for pattern recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 36, 7, July, 1988, pg. 1109-18.

[13] Yamada, Kami, Tsukumo and Temma, "Hand written numeral recognition by multi-layered neural network with improved learning algorithm," *International Joint Conference on Neural Networks*, Washington DC, Volume II, June 18-22, 1989, pg. 259-66.

ADAPTIVE POLE PLACEMENT FOR NEUROCONTROL

Sanjay S. Kumar and Allon Guez
Dept. of Electrical & Computer Engineering,
Drexel University, Philadelphia, PA 19104

Abstract : In this paper we deal with the study of a self organizing neural network architecture called the *Adaptive Resonance Theory* (ART-II) [Grossberg and Carpenter 1987(a)], in its application to a simple problem of adaptive control, through real time dynamic system identification. An adaptive pole placement controller for a linear second order system is implemented based upon this architecture to assess the performance of the network and the overall control scheme with the neural network in the control loop. The network employed demonstrates the capabilities of *fast classification* and *learning*. These attributes of the architecture are exploited to train a network to dynamically identify parametric variations of a plant in response to changes in its environment. The control strategy is based upon identification of changes in the time response characteristics of the system to standard test signals which are assessed by the neural network. A pole placement algorithm is utilized to relocate the poles of the overall closed loop system by altering the gains of the process controller to obtain desired system response. Experimental studies on a simulated second order system, employing a Proportional Derivative controller show that the neural network considered is indeed capable of system identification and simple indirect adaptive control of low order plants that are subjected to parametric variations reflected by changes in their operating environments.

Problem Statement : Our goal is to implement an adaptive pole placement algorithm using a neural network architecture. Let the possibly slow time varying linear dynamical system (Plant) G_p, be given as in equation (1).

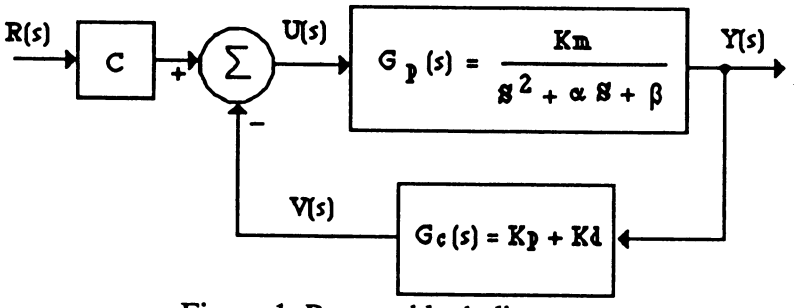


Figure 1 Process block diagram

Plant transfer function : $G_p(s) = \frac{Km}{s^2 + \alpha s + \beta}$ (1)

Let the reference input be a known periodic function

Reference input: $r(t) = r(t + T)$, for all $t > 0$ (2)

Find online, the Proportional and Derivative (PD) gains, (Kp, Kd) and the D.C. bias (C) (see Figure 1), such that the control law give by equation (3),

Control law : $u = C\,r - K_p y - K_d s\,y = C\,r - G_c y$ (3)

will result in the ideal closed loop transfer function

$$G_O^*(S) = \frac{Y(s)}{R(s)} = K^* \left[\frac{\omega_n^{*2}}{s^2 + 2\zeta^* \omega_n^* s + \omega_n^{*2}} \right] = G_o \qquad (4)$$

where, K^*, ζ, ω^*_n are respectively the desired D.C. gain, damping coefficient and the natural frequency of the system.

Neuromorphic approach to the controller design : The general block diagram of the overall control scheme is depicted in figure 3. The common a priori assumption in the design of controllers for partially known processes is adopted with the design procedure being divided into two steps: *identification* and *control* (indirect adaptive control strategy) [Astrom & Eykhoff,1971].

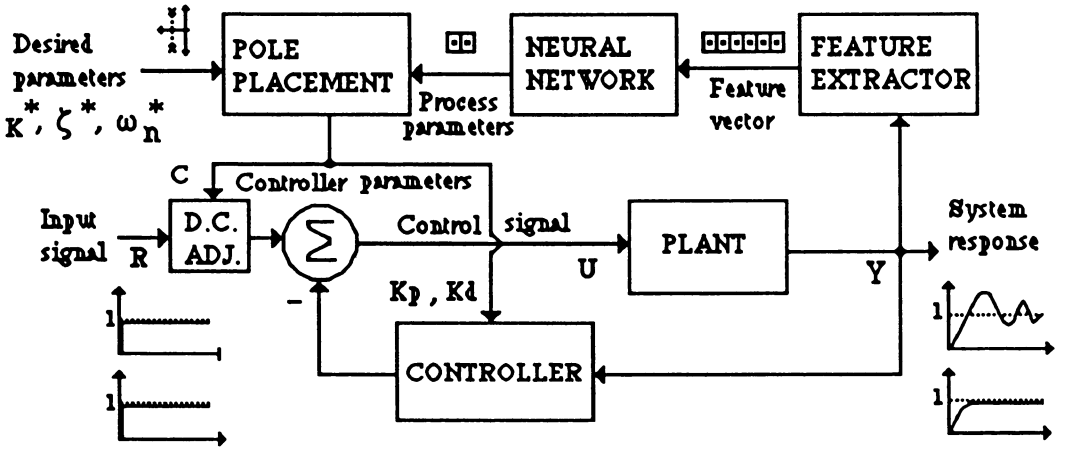


Figure 2 Block diagram of the neuromorphic adaptive control scheme

Objective : Given a second order plant with unknown constant or time varying parameters, the objective of the control scheme is to tune the controller by modifying its arbitrarily assigned initial gains such that the overall closed-loop system response matches the one given by the assumed ideal model, based upon parameter identification provided by the neural network in the control loop. It should be noted that although the plant and the controller are linear the process is overall nonlinear.

Methodology : When the input is a unit step function, identification of the plant parameters is achieved by extracting the features of the system's closed-loop transient time response via a feature extractor. In the case of the square function and the square wave function, identification is restricted to the step portion of the input signal. The neural network in the control loop which is trained to map the features of a system's time response to its parameters, gives the current parameter estimates of the closed loop process. The pole placement algorithm in the loop utilizes the estimates provided by the network to compute new controller gains to suit the desired response specified by the ideal model. A D.C. adjustment mechanism is incorporated to compensate for any D.C. bias that might be associated with the response. The system identifier comprises of the feature extractor, the neural network and the pole placement algorithm. The purpose of the identifier is to determine the plant parameters in response to changes within the plant environment. The time response of the system is characterized by its *features* or *performance indices* which are nonlinear functions of the system parameters.These include the rise time, the settling time, overshoot and the delay time of the response. A *feature extractor* incorporated in the control loop determines the performance indices associated with the response to enable system identification via the neural network. It must be noted that identification is dependent on the features of the time response rather than the response per se. This is done in order to compress the information contained in the response such that the input vector to the neural network remains compact [Kumar & Guez,1989]. This procedure restricts the dimension of the neural network to a minimum thereby increasing its computational speed and overall efficiency of the process. The overall transfer function is obtained by using equations (1),(2) and (3) as follows:

$$Y(S) = G_p(S)\,U(S) = Gp(S)\,[C\,R(S) - G_c(S)\,Y(S)] \qquad (5)$$

$$G_O(s) = \frac{Y(s)}{R(s)} = \frac{C\,G_p(s)}{1 + G_p(s)\,G_c(s)} = \frac{CK_m}{s^2 + (\alpha + K_m K_d)\,s + (\beta + K_m K_p)} \qquad (6)$$

multiplying and dividing the numerator by $(\beta + K_m K_p)$,

$$G_O(s) = \frac{CK_m}{(\beta + K_m K_p)} \left[\frac{(\beta + K_m K_p)}{s^2 + (\alpha + K_m K_d)s + (\beta + K_m K_p)} \right] \quad (7)$$

In order to obtain the ideal transfer function, we set $G_O(S) = G_O^*(S)$ which implies:

$$\frac{CK_m}{(\beta + K_m K_p)} \left[\frac{(\beta + K_m K_p)}{s^2 + (\alpha + K_m K_d)s + (\beta + K_m K_p)} \right] = K^* \left[\frac{\omega_n^{*2}}{s^2 + 2\zeta^* \omega^*_n s + \omega_n^{*2}} \right] \quad (8)$$

the new controller gains obtained from the above equations are then given by:

$$K^*_p = \frac{\omega_n^{*2} - \beta}{K_m} \qquad K^*_d = \frac{2\zeta^* \omega^*_n - \alpha}{K_m} \qquad C^* = \frac{K^*(\beta + K_m K^*_p)}{K_m} = \frac{K^* \omega_n^{*2}}{K_m}$$

Training: The network employed is used in the *supervised learning mode* and is trained off line before its inclusion into the control loop.The *training data module* comprises of a data generator whose inputs form the approximate ranges of the system parameters which, in the present application, are the *natural frequency* and the *damping ratio* of the generalized second order system. A typical range of the natural frequency is estimated from the knowledge the plant time constant used in the process.The ranges selected thereof cover approximately the entire gamut of the systems time response to maximum deviation in the plant parameters. A *system emulator* that consists of the process model generates the actual system time response to the various input signals using different settings of the parameters within the parameter ranges specified. The parameters are incriminated in discrete steps according to a prescribed resolution that is dependent upon the trade off between accuracy of identification desired and the size of the network. The time responses of the assumed system model are then passed on to the *feature extractor* that determines the various features or performance indices of the response. The performance indices along with the respective system parameters form the training data set for the neural network. Once the training data is available, the network is configured such that the number of nodes in the feature representation field corresponds to the dimension of the input feature vector,while the number of processing nodes in the category representation field are generally set greater than total number of input patterns in the prototype set. Each pattern in the prototype set is sequentially presented to the network once. A second cyclic presentation of the prototype set may be made for a stable category confirmation. During training, the attentional vigilance parameter is set at its highest value (0.999) to ensure a high resolution of the resulting category structure. The category structure represents the state space partitioning of the neural network depending on the number of stable categories established during training.When the network is presented with a feature vector for the first time, it is encoded in the LTM through modification of the LTM connection weights. The parameters associated with the feature vector now get assigned to this allocated node in the category representation field. On presentation of the subsequent feature vectors, the network's orienting subsystem first determines closeness of match between the pattern currently imposed on the network and any of the patterns that have previously been seen. Since the vigilance parameter is set so high a new node is allocated for the pattern. However, if the current pattern happens to be exactly similar to the one the network has seen before it is clustered into the same category. It is therefore possible to partition the networks state space such that each partition serves as an attractor for a particular type of response characterized by its feature vector. After completion of training the top down and the bottom up connection weights of the network along with the network parameters are saved. To make the network uniformly sensitive to all components of the feature vector, the feature vector components were enhanced by passing the feature vector through a nonlinear function given by:

$$f_i(x) = e^{\gamma x} \quad \text{where } \gamma = 0.05$$

Testing: When the network is introduced in the control loop, identification proceeds almost instantly. Search for the category associated with the right parameters is achieved by dynamically altering the attentional vigilance parameter until an "optimal vigilance" is found, [Kumar & Guez, 1989].

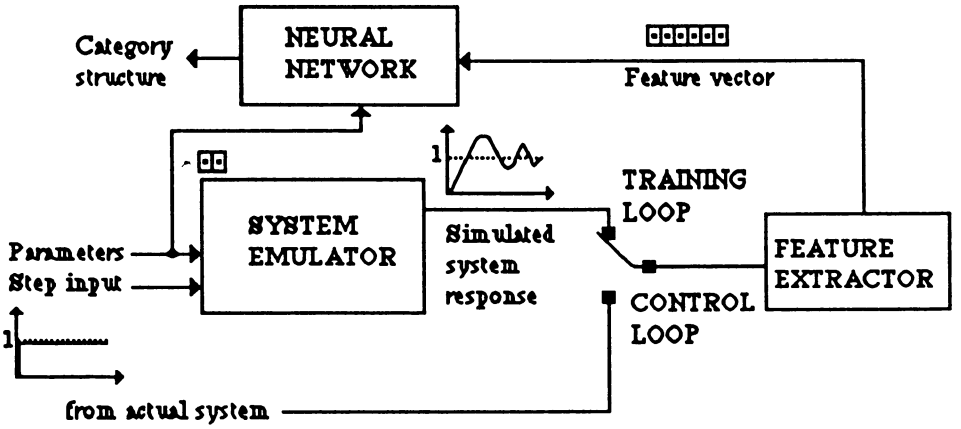


Figure 3 Block diagram of network training module

Experimental Results : The time response of the system is obtained using a Runge-Kutta fourth order differential equation solver (RK-4).Each of the following figures depict two plots. (a) The response of the actual system to the unit step input signal with arbitrary initial parameters α, β, and K_m, and (b) the final system response with the *D.C. bias* or the *steady-state error* compensated through the parameter C. The neural network employed is trained off-line on the features of the response obtained using the generalized second order equation with the following parameter variations:

$$0.1 \leq \zeta \leq 1.50, \ \Delta\zeta = 0.10$$
$$0.5 \leq \omega_n \leq 2.50, \ \Delta\omega_n = 0.50 \qquad (9)$$

These parameters are not made available to the system identifier but are estimated through the neural network in the control loop, based upon the features of the system time response to the standard test signals. The only available information to the feature extractor module which precedes the neural network in the overall process block diagram is the time and the value of the system response at that time instant. Specification of the parameters α_o and β_o enable in the computation of the desired damping and the desired natural frequency of the system response based upon the desired location of the system closed loop poles within the left half of the S- plane. In the case of the square function and the square wave, it should be noted that feature extraction was restricted to the step portion of the response (12 seconds). Figure (a) and (b) represent the nominal responses of the assumed ideal model to the unit step and the square function. Figures (c) and (d) depict the time variation of the plant parameters during the identification process. Figures (e) and (f) show the ability of the proposed neurocontroller to modify an overdamped and an underdamped system response respectively, with the input being a unit step function. It should be noted that the original system response indicated refers to arbitrarily assigned initial plant parameters. The final controller parameters are those obtained after the plant parameters are estimated by the neural network system identifier and the relocation of the overall closed loop system poles by the pole placement module. Figures (g) and (h) refer to the square input function. In Figures (i) and (j) are shown the output responses of the plant when the square wave is imposed at the input. The time scale incorporated in the simulation is adaptive for the unit step input and depends upon the duration of the system transient response.The sampling frequency selected is common to both the response sampling rate and the RK-4 differential equation solver. It is well above the Nyquist frequency of the system.

Range Image Analysis Using Neural Network

Robert Y. Li and Huaxiao Si
Department of Electrical Engineering
University of Nebraska, Lincoln, NE

Abstract

Range images contain three-dimensional information. Human can use 3-D information to identify or classify objects. This capability is simulated by artificial neural networks using features extracted from range images. Two models, back-propagation and counter-propagation, are used and their results are evaluated.

Introduction

3-D target recognition is a very important topic in computer vision. Range information from a CO_2 laser sensor can provide 3-D information and these information can be used for target recognition. In many ways, it is similar to the human's ability to use depth information to identify the target. A person can discriminate between different objects based on approximate geometric estimation. In this report, the geometric information is first extracted by processing laser range images. Then the geometric features are analyzed by models of artificial neural nets for classification purpose. Two models are used: back-propagation and counter-propagation. The results of these two models will be compared.

Range Image Processing

In this research effort, range imagery was obtained from a multi-modal CO_2 laser radar Range data offer significant advantage because they preserve the 3-D geometry of the scene viewed from the sensor. The laser data set includes relative range images with 15 meter range ambiguity[1] By applying the median filtering and conversion algorithm, the relative range data is converted into absolute range data. Figure 1 shows a tank scene in a converted absolute range image. The converted absolute range image shows that the range values change gradually and smoothly from pixel to pixel, and the range ambiguity is removed.

The 3-D Hough transform is then used to detect and measure planar surfaces. A plane in the 3-D space can be described by the following equation:

$$\cos\delta \cdot \cos\theta \cdot x + \cos\delta \cdot \sin\theta \cdot y + \sin\delta \cdot z = D$$

where θ, δ and D are pan angle, tilt angle and normal distance.

To detect a plane, we check for any peak in the accumulator of these three parameters (θ, δ, D). When θ, δ, and D extend over a large range, the calculation can be very time-consuming. However, since most surfaces are either vertical or horizontal, the Hough transform for 3-D space can be further simplified[2,3] Our first goal in applying the Hough transform to the data is to estimate target orientation by detecting a dominating vertical surface in the

target. This is done by locating the peak in the (θ, D) accumulators using the xy values of each pixel. We reasoned that most vehicles had either a flat top or a large deck, and the portion below that flat horizontal surface was more box-like. Therefore, vertical pixels below the major flat surface are more suitable for the calculation of target orientation. Using this approach, correct orientation angles were obtained for most targets under our consideration.

After target orientation is determined, geometric features such as height, width, and length of the target and the percent of pixels above the target deck can be calculated. Target orientation was first found by using only vertical pixels below the major horizontal surface, and using only angles from -90 to 90 degrees in the Hough space. Then with the angles fixed, we found the target dimensions by counting valid intervals of the normal distance along these angles. See Figure 2 for Hough space representation in left-facing projections. Finally, we also calculated the percentage of pixels above and below the major horizontal surface (which is the deck in the case of the tank). Table 1 lists the results of computing target orientation and dimensions for six different targets. Those measurements on height, length, width, and the percent of pixels above the deck are used as feature inputs to the artificial neural networks to be described below.

Neural Networks

(1)Back-propagation: This model is to compute the derivative of the error function with respect to any weight in the network and then change that weight according to the learning rule below[4]:

$$\Delta W_{ij} = \varepsilon \cdot \delta_i \cdot a_j$$

In other words, the weight on each line should be changed by an amount proportional to the product of an error signal δ, available to the receiving ith unit, times the activation, a_j, of the jth unit sending activation along that line. The symbol ε is a constant. In this case, we consider that the output of a unit is equal to its activation. The determination of δ is a recursive process that starts with the output units of the network.

If a unit is an output unit, then its δ is given by

$$\delta_i = (t_i - a_i)\, f_i(\mathrm{Net}_i)$$

where $f_i(net_i)$ is the derivative of an activation function with respect to a change in the input to the unit, and t_i is the target value. Then, the error signal for hidden units for which there is no specified target is determined recursively; by summing up the δ terms of the units to which it directly connects and the weights of these connections. In other words,

$$\delta_i = f_i(\mathrm{Net}_i) \cdot \sum_k \delta_k W_{ki}$$

In terms of overall operation, the back propagation model can be divided into forward and backward phases. After two phases are completed, we obtained the weight change ΔW_{ij} needed for each weight using the equation shown earlier.

(2)Counter Propagation: This neural network model constructs a mapping from a set of input vectors X to a set of output vectors Y. See Figure 3 for the configuration. It essentially acts as a hetero-associative nearest-neighbor classifier[5]. After input features are normalized, they were sent to the Kohonen layer which acted as a nearest neighbor classifier. The neurons in this layer compete with each other. The one with the highest output wins. The Kohonen learning rule is the following:

$W_i' = W_i + A (X - W_i)$ where W_i is the weight , X is the input and A is the learning rate.

The output layer (Grossberg Outstar) is basically a processing element which learns to produce a certain output when a particular input is applied. Sine the Kohonen layer produces a single output, this output layer is essentially a decoder for specified output category.

(3)Experimental Results: Six feature patterns from the Table 1 were used as training inputs to the neural network. Then during the recall or the test stage; many simulated patterns, with small statistical variation from the standard measurement, were analyzed by the network. The classified result has been 100% correct for the test patterns using either model. The pattern is determined to be either a tank or a truck. Figure 4 shows the plots of the cumulative errors as a function of the number of training patterns. In learning stage, both models converge quickly to the desired error tolerance. It can be seen that the counter-propagation model has a faster convergence rate. It takes a fraction of the time needed for the back-propagation model.

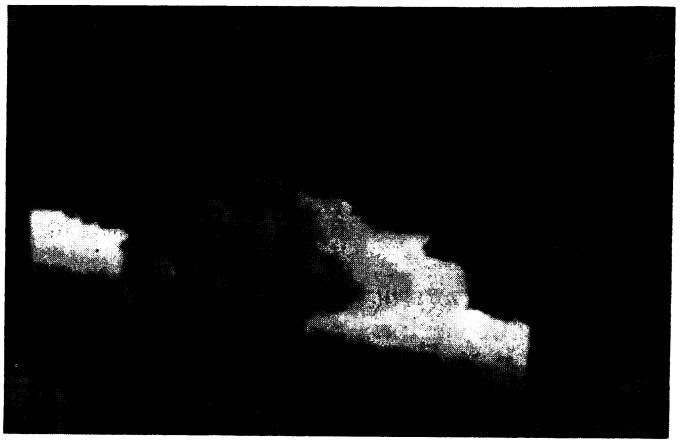
Summary

Range images provide 3-D information. Human can use 3-D information to classify object category. The same capability is simulated by using artificial neural network. Both models, back-propagation and counter-propagation, work well. More research should be done in using neural network to extract information from range images.

References:

1. G. S. Bain and W. K. Cunningham, "Multifunction Laser Radar," Technical Report, LTV Aerospace and Defense Co., May 1986.
2. P. Besl and R. Jain, "Range Image Understanding," *Proceedings of Conference of Computer Vision and Pattern Recognition*, June 1985.
3. R. Y. Li, "Range Image Processing for Object Recognition," *Proceedings of the 30th Midwest Symposium on Circuits and System*, Syracuse, N. Y., Aug., 1987.
4. D. E. Rumelhart, J. L. McClelland, *Parallel Distributed Processing*, pp.318 - 332, The MIT Press, 1986.
5. C. C. Klimasauskas, NeuralWorks User's Guide, Sewickley, Pa. 1989.

Figure 1. Converted absolute range image (T



gure 2. Hough space in the left-facing direct
Each intensity level represents an in... al.

Target Type	M-60 tank-	M-60 tank	M-60 tank	M-60 tank	fuel truck-	fuel truck
Analysis Result						
Angle, deg-	60	10	10	-70	20	-70
Distance, meter	314	307	880	300	300	371
Width, meter	4.0	5.0	3.75	3.5	3.75	3.0
Length, meter	6.5	5.25	5.0	6.0	12.0	12.0
Height, meter	3.0	3.5	3.5	3.0	3.25	3.25
% pixel above deck	29.8	46.1	30.5	33.3	6.4	3.8
Total no. of target pixels	1940	1365	466	1255	1551	1448

Table 1 Extracted target features from range images

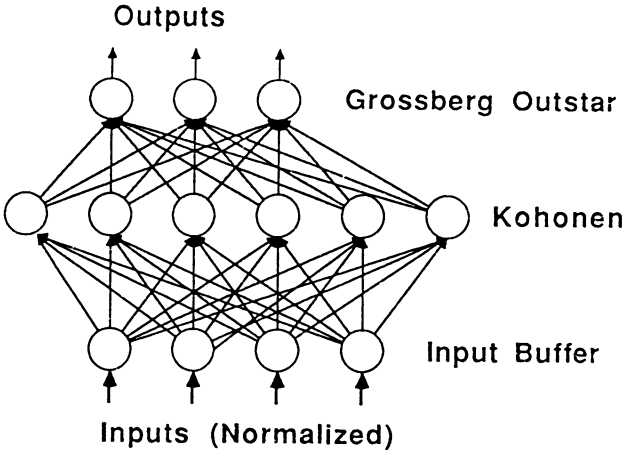


Figure3. A counter-propagation network.

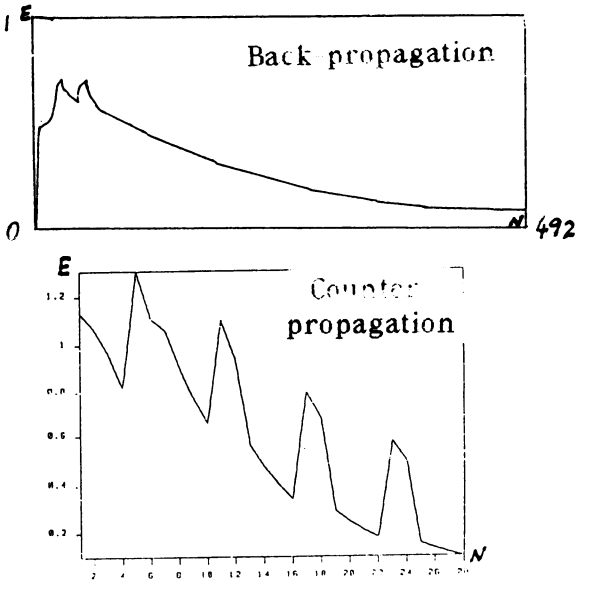


Figure 4. Plots of cumulative errors (E) vs.
# of training patterns (N)

A Self-Organizing Recursive Network for Object Recognition

Herwig Mannaert*, André Oosterlinck
ESAT-MI2, Department of Electrical Engineering
K.U.LEUVEN, Kardinaal Mercierlaan 94
B-3030 Heverlee, BELGIUM

Abstract

A parallel, self-organizing recursive network is proposed for object recognition in images. Based on the self-similar structure of the network, conceptual learning mechanisms are presented enabling the network to recognize learned reference patterns under a well defined class of transformations. In order to use the system in image understanding, some algorithmic aspects are discussed. Finally, some applications of the system, operating on real images, are shown.

1 Introduction

A system *learns* a pattern by an *imprint* which the pattern makes in the system. The Hebb-rule in classical ANN is an example of such an imprint. The learning by imprint principle however is far more general than the Hebbian rule and should not necessary be restricted to this specific implementation. We propose here the use of parallel, recursive pointernetworks, to implement the learning and the creation of pattern attractors. These *parallel* and *self-organizing* networks are highly suited for our modern computers and can be treated in a purely static way.

We describe in a first section the self-similar structure and learning mechanisms of the pointernetworks. A second section deals with some algorithmic aspects if the system is used for image understanding. Some examples of applications will be given.

2 Learning in Self-similar Networks

The architecture of connections is embedded in the tree-like pointer-structure of the cells in our network. Every cell in the self-similar structure has a connection to the next cell on the same hierarchical level, and a list of connections to its components on the lower level. A code specifies how its value should be determined from the cell-values of the lower hierarchical level. The leaf-cells contain additional information. We now show how the system can learn or assimilate information by creation or modification of a cell on some recursive level:

*Herwig Mannaert is a Research Assistant of the National Fund for Scientific Research (Belgium).

1. A pattern is a m-dimensional binary or analog vector function (features) of n variables (space dimension): $rf_i(x)$. The assimilation of a pattern creates a *pattern-cell*, pointing to a list of leaf-cells, each containing the identification of a sensor, activated by the reference pattern. In the non-binary case, an extra attribute (weight) specifies the value of the reference variable-unit. A multiplication code in the leaf cell and a summation code in the pattern cell results in a classical correlation, though other possibilities exist (mean or median for the pattern cell, similarity measure for the leaf cells).

2. Objects, appearing at different scales, orientations and positions in images are in fact equivalence classes of patterns. We define a pattern $rf_i(x)$ to be equivalent with a reference pattern $\overline{rf_i}(x)$ if there exists a direct similitude Φ for which: $rf_i(\Phi(x)) = \overline{rf_i}(x)$. The similitudes and therefore the equivalence class can be characterized by

$$1 + n + C_n^2 = \frac{n^2 + n + 2}{2} \tag{1}$$

parameters in n-dimensional Euclidean space, guiding the rotation (C_n^2), scaling (1) and translation (n) which transform the reference pattern into the equivalent pattern. The entire equivalence class is assimilated by the network if we create an *object-cell*, pointing to pattern-cells for all possible transformations. For transformation Φ for instance, one creates a list with connections ($\overline{rf_i}(x)$ is an active sensor in the reference pattern):

$$\forall \overline{rf_i}(x) : \phi \rightarrow I_i(\Phi(x)) \tag{2}$$

Each transformation cell Φ will compute a probability measure for the presence of pattern $rf_i(\Phi(x)) = \overline{rf_i}(x)$. For classical correlation (multiplication and summation), it computes:

$$P(\Phi) = \sum_x \sum_{i=1}^{m} I_i(x) \times \overline{rf}_i(\Phi^{-1}(x)) \tag{3}$$

The object cell takes usually the maximum of all transformation cells.

3. Different objects result in a *classification cell*, pointing to a list of object cells. The classification can also be made at the pattern level for every possible transformation.

4. The introduction of OR-cells at a certain level defines different patterns or objects to be equivalent. At the level just above the leaf-cells for instance, this accounts for small distortions. We generalise the OR-ing for analog values:

$$\bigcup_{i=1}^{N} a_i = \max_i a_i \tag{4}$$

5. One can tell the system that certain sensors or sensor combinations are essential to the nature of the object. This is implemented by the generation of AND-cells, generalised to account for analog values in a similar way.

6. It is possible to assocciate symbols with certain primitives of the network, like names with cells at the object level, comparisons of size between different objects based on the scale, relative position of various objects, names of essential features of an object, ... As a next step, one could try to create semantic relations between these symbols.

3 A Network for Image Understanding

As a first step in obtaining a practical algorithm, we decouple the parameter set of the transformations. Unlike the subspace approach of [1], we make a conceptual distinction between the n translation parameters on one side, and rotation-scale parameters on the other. Translation parameters are considered as a sequential wandering through time or space during evaluation or operation, while the others are used in a real invariance network constructed during the imprint or assimilation. The use of *and-, mean-, median*-cells resulting in a selective activation of the network, is a possible way to *attract the attention.* The system could stop the wandering process and start a zoom action.

The sensor model we use is an edge-detector for object boundaries, consisting of four sensors: two directions (horizontal and vertical) which can be positive or negative. The sensors compute a horizontal and vertical derivative of the illuminance function in every point[1]. Every region between two zero-crossings z_1 and z_2 of the derivative function in a given direction σ is considered as an edge. This edge is located at the point where the derivative is maximal (in absolute value) and has a value corresponding to the total illuminance change:

$$\Delta_{z_1\sigma}^{z_2\sigma} I(z\sigma) = \int_{z_1\sigma}^{z_2\sigma} \frac{dI(z\sigma)}{dz\sigma} dz\sigma \tag{5}$$

The equations represent a *competition* and a *cooperation* between the sensor cells. We also introduce a preliminary competition between horizontal and vertical sensor at every point.

The discretisation of rotation and scale parameters demands the use of *regions* as receptive fields, in order to detect all possible equivalent patterns. The sensor field is therefore converted into a *logarithmic-polar* mapping. This mapping, known to be used in biological systems, is geometrically quite natural as the sensor areas keep their morfological form (both arc and radius length of the areas evolve proportional to the radius), and a scale-rotation transformation results in a simple addition of the coordinates. Tough it would be more elegant to use the $\theta - r$-directions of the log-polar as sensor directions, we use the cartesian directions to avoid the recomputation of the image for every possible translation. The cartesian sensor pattern is converted into a log-polar sensor field. To avoid the normalisation problem (our curvilinear boundaries give an input proportional to r^1 while noise is proportional to surface area and therefore r^2), we take simply the strongest value in every region.

At the moment, reference sensor images are created by the user. The system needs an integer image for the horizontal and one for the vertical edge sensors, and uses some rules for the interpretation of the reference patterns. It will for instance create OR-cells for identical changes on one line in a given direction and AND-cells for complementary changes. In the future however, reference patterns will be pointed out with a mouse on real images. The system will also ask itself for names and symbols corresponding to certain objects and features.

The system runs on a VAX-8530 and requires less then an hour CPU-time for 128 × 128 images with a single pixel as translation step. It has been tested succesfully for the detection

[1] The sensor model requires n orthogonal directions in n-dimensional space. If σ is the orthogonal direction with respect to the hyperplane approximating the object boundary, there is always at least one direction α for which:

$$(\alpha, \sigma) \geq \frac{1}{\sqrt{n}}$$

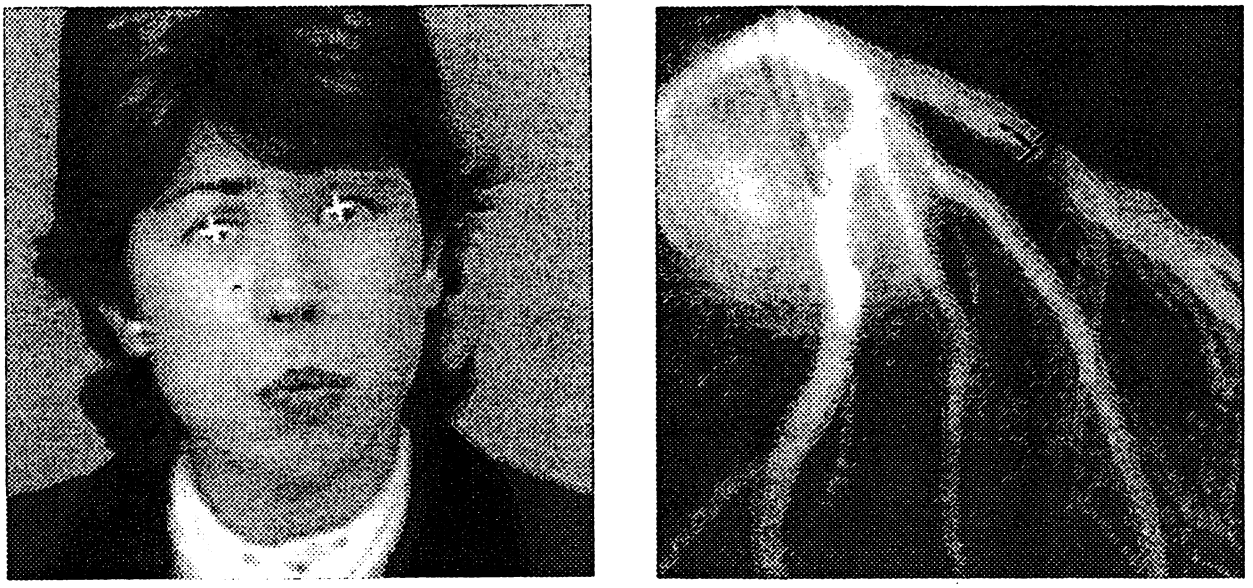


Figure 1: Detection of eyes in a human face (left), and stenose in an angiogram. The system puts crosses at the appropriate place, size and orientation.

of all kinds of simple objects (registration plates of driving cars, contours of human faces) or curvilinear structures (blood vessel, roads on sattelite images). Figure 1 gives two other examples: the detection of the eyes in a human face, and a stenose in a blood vessel angiogram.

References

[1] Ballard D.H., "Cortical Connections and Parallel Processing: Structure and Function", The Behaviour and Brain Sciences, Vol.9, pp.67-120.

[2] Hutchinson J.E., "Fractals and Self Similarity", Indiana University Mathematics Journal, Vol.30, No.5 (1981), pp.713-747.

[3] Trehub A., "Neuronal Models for Cognitive Processes", Journal of theoretical Biology 65 (1977), pp.141-169.

Shape Recognition by Ring Hidden Markov Models

W.D. Mao and S.Y.Kung

Department of Electrical Engineering
Princeton University, NJ 08544

Abstract

A two dimensional shape recognition approach utilizing Hidden Markov Model (HMM) as knowledge source is discussed. The shape is represented by a sequence of curvature values. A Ring Hidden Markov Model (RHMM) neural network which incorporates a ring structure and local connectivity is proposed. A new learning algorithm is specifically designed for the network. Simulation results indicate that the proposed RHMM system can achieve almost 100% recognition accuracy at a very fast learning speed.

1 Introduction

Automatic target recognition requires fast, yet accurate classification technique based on efficient description of the objects. The two dimensional shape contains much information of an object. Many conventional approaches to 2D shape recognition have been proposed, e.g. moments and Fourier descriptors methods. Despite of some success, they are inefficient for classifying shapes with local feature ambiguity. In addition, their capability is limited by the lack of compact knowledge source and efficient learning mechanism. In this paper, we propose a stochastic *Ring Hidden Markov Model (RHMM)* approach for shape recognition which specifically uses the curvature features of the shape.

1.1 Representation of Shape

Human eyes are known to be sensitive to the inflection points of shape contour, which correspond to large curvature values [1]. Therefore in the feature representation stage, we employ an approach based on the curvature signal of the shape. Given a planar curve, the curvature signal $\kappa(t)$ is defined as the instantaneous rate of change of the tangent angle $\theta(t)$ with respect to curve length t. i.e. $\kappa(t) = \frac{d}{dt}\theta(t)$, where curve length t is normalized for scale invariance. However, the derivative can not be reliably computed if there is noise on the boundary of shape. Thus, a Gaussian mask is used for filtering the noise [1]. The new curvature signal $K(t)$ is:

$$K(t) = \frac{d}{dt}(G(\sigma,t) * \theta(t)) = G'(\sigma,t) * \theta(t)$$

where $G(\sigma,t)$ is a Gaussian function with zero mean and standard deviation σ, $G'(\sigma,t)$ is its derivative w.r.t.t.

It can be shown that inflection points always appear at local maximum and minimum of the filtered curvature signal $K(t)$. These curvature values correspond to sharp corners of the contour. The curvature signal between these points can be uniformly sampled to represent smooth arc in the contour.The final *curvature sequence* is represented by $\{K(t_1), K(t_2), \ldots, K(t_T)\}$, where $t_1 < t_2 < \ldots < t_T$. The *local features* of the shape are curvature segments along the contour of shape corresponding to a sharp corner or a smooth arc or some combination of them. There are also some *global constraints* of the shape, including: (1) the local features along the contour have a specific order; (2) the contour should be closed. Thus the shape recognition task is matching the local features under the global constraints.

1.2 Recognition of Shape

HMM has been widely explored in speech recognition [5] [4], and recently it is used in character recognition [6] and texture segmentation [2]. It is a doubly stochastic knowledge source which can model various kinds of nonstationary signal. Basically, we construct a HMM for each class of shape. Curvature sequence of the shape is sequentially generated by each model according to the probability distributions in each state and the state transition probabilities. The HMM uses the Baum-Welch learning algorithm [5]. By the training procedure, all the stochastic parameters in HMM can be estimated. In the recognition stage, the model with the highest probability of producing the curvature sequence is selected.

There are two major problems when directly applying conventional HMM to shape recognition:

1. HMM is a fully connected network which cannot impose the global constraints such as the specific order of local features along the contour and the closed boundary constraint.
2. The original Baum-Welch learning algorithm converges slowly for training the cyclic shifted observation sequences generated by rotated shapes. This will makes it difficult for rotation invariant shape recognition.

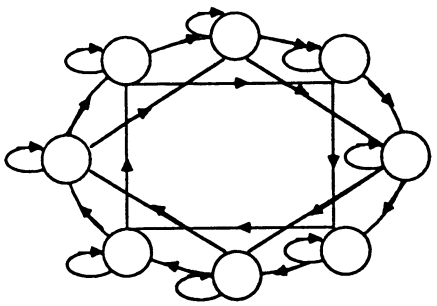


Figure 1: The locally connected ring HMM (RHMM) network

To solve these problems, we propose a modified *Ring Hidden Markov Model (RHMM)* networks for shape recognition. The model is a unidirectional ring network with local connections. The proposed RHMM system has some unique features, compared with the conventional HMM:

1. It employs a unidirectional ring structure to incorporate the global constraints that the sequence of curvature value along the contour should be specifically ordered and the contour should be closed.
2. It only has local state transitions. Thus the neighboring local features along the shape contour can be merged or split based on transition probabilities. This will provide certain degree of robustness for learning local feature.
3. A modified Baum-Welch learning algorithm is proposed for the RHMM, which provides training procedure for all stochastic parameters of the network under the global constraint. It can also learn shapes with rotation invariance.

Simulation on aircraft recognition will show that the proposed RHMM networks can achieve better classification accuracy and faster learning compared with the conventional HMM approach.

2 RHMM System for Shape Recognition

In this section, we will describe the ring HMM (RHMM) system for shape recognition. Basically, we construct a RHMM for each class of the shapes. Each model is an unidirectional locally connected ring network shown in Figure 1. At the beginning, the model parameters are chosen randomly or based on some initial guess. There are two phases in the approach:

- **Scoring Phase:** For a shape curvature sequence (*observation sequence*), the probabilities of the sequence given each model are computed. The model associated with the maximum probability is selected.
- **Learning Phase:** For a training sequence and the corresponding model, the model parameters such as probability distributions in each state and state transition probabilities are reestimated.

The system parameters are defined as:
N = number of states (nodes) in the network.
M = number of possible observation symbols.
T = length of the input observation sequence.
Q = $\{ q_1, q_2, \ldots, q_N \}$ set of states.
O = $\{ o_1, o_2, \ldots, o_T \} = \{ K(t_1), K(t_2), \ldots, K(t_T) \}$ observation sequence.
$\pi = \{ \pi_i \}$, $\pi_i = Prob\{q_i \ at \ t = 1\}$, initial state distribution.
A= $\{a_{ij}\}, a_{ij} = Prob\{q_j \ at \ t+1 \mid q_i \ at \ t\}$, state transition probability.
B= $\{b_j(k)\}, b_j(k) = Prob\{o_t = k \ at \ t \mid q_j \ at \ t\}$, probability distribution in state j.
λ= complete parameter set of the model.

2.1 Scoring Algorithm

To find out which model is most likely to have produced the observation sequence, we need to calculate the probability $P(O|\lambda)$. The locally connected ring HMM has the constraint:

$$a_{ij} = 0 \ \ if \ j \notin \Delta_r(i) \ \ or \ \ a_{ij} = 0 \ \ if \ i \notin \Delta_{-r}(j).$$

where $\Delta_r(i) = \{(i+l) \ mod N \mid l = 0, 1, 2, \ldots, r\}$, $\Delta_{-r}(j) = \{(j-l+N) \ mod N \mid l = 0, 1, 2, \ldots, r\}$, r is a positive integer (e.g.2).

The *forward probabilities* are defined as $\alpha_t(i) = Prob(O_1 O_2 \ldots O_t, q_i \ at \ t \mid \lambda)$, which can be recursively calculated by the procedure:

1. $\alpha_1(i) = \pi_i b_i(O_1)$, $1 \leq i \leq N$;
2. for $t = 1, 2, \ldots, T-1$, $1 \leq j \leq N$

$$\alpha_{t+1}(j) = [\sum_{i \in \Delta_{-r}(j)} \alpha_t(i) a_{ij}] b_j(O_{t+1});$$

3. $Prob(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i)$.

Similar procedure can be used to calculate the *backward probabilities*:

$$\beta_t(i) = Prob(O_{t+1} O_{t+2} \ldots O_T \mid q_i \text{ at } t, \lambda).$$

1. $\beta_T(i) = 1$, $1 \leq i \leq N$;
2. for $t = T-1, T-2, \ldots, 1$, $1 \leq i \leq N$

$$\beta_t(i) = \sum_{j \in \Delta_r(i)} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)$$

Note that in the scoring phase, the procedure is essentially the same as HMM except all the Σ is taken only on r local neighborhood values, which implies a computation saving by a factor N/r.

2.2 Learning Algorithm

We propose a modified Baum-Welch learning algorithm for the RHMM network:

Define $\alpha_t^s(i) = \{\alpha_t(i) : \alpha_1(i) = b_i(O_1) \text{ for } i = s, 0 \text{ otherwise}\}$. Define $\beta_t^s(i) = \{\beta_t(i) : \beta_T(i) = 1 \text{ for } i = s, 0 \text{ otherwise}\}$.

Define:

$$\begin{aligned} \xi_t^s(i,j) &= Prob(q_i \text{ at } t, q_j \text{ at } t+1 \mid O, \lambda, q_s \text{ at 1 and } T) \\ &= \frac{\alpha_t^s(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}^s(j)}{Prob(O \mid \lambda, q_s \text{ at 1 and } T)} \end{aligned}$$

The algorithm has two key steps:

- Step 1 identifying starting state. Chose $s^* = arg \max_{1 \leq s \leq N} \alpha_T^s(s)$, which is the most likely initial as well as final state of the sequence. This new addition to HMM will estimate the starting state before learning. In this way, the cyclic shifted sequence generated by rotated shapes can be learned.
- Step 2 training model parameters. Define $\gamma_t^s(i) = \sum_{j=1}^{N} \xi_t^s(i,j)$. Then the learning procedure is:
  1. $\bar{\pi}_i = \gamma_1^{s^*}(i)$, *for* $1 \leq i \leq N$;
  2. $\bar{a}_{ij} = \sum_{t=1}^{T-1} \xi_t^{s^*}(i,j) / \sum_{t=1}^{T-1} \gamma_t^{s^*}(j)$, *for* $\{i, j | j = \Delta_r(i)\}$;
  3. $\bar{b}'_j(k) = \sum_{t=1, O_t=k}^{T-1} \gamma_t^{s^*}(j) / \sum_{t=1}^{T-1} \gamma_t^{s^*}(j)$, *for* $1 \leq j \leq N, 1 \leq k \leq M$;
  4. $\bar{b}_j(k) = \sum_{m=1}^{m=M} \bar{b}'_j(m) G(\frac{k-m}{\sigma}) / \sum_{k=1}^{M} \sum_{m=1}^{m=M} \bar{b}'_j(m) G(\frac{k-m}{\sigma})$, *for* $1 \leq j \leq N, 1 \leq k \leq M$.

Here step 4 is a new addition to HMM. It introduces a normal distribution $G(x)$ to provide robustness on the state probability distributions $b_j(k)$. This is necessary in order to tolerate distortion of the contours.

3 Simulation

In this section, we use eight aircraft images (512 x 512) for shape classification experiment. Figure 2 shows the eight types of aircraft used for simulation. The conventional HMM and proposed RHMM approach are compared in terms of recognition accuracy and learning speed. In both approach, we set up 30-state model for each kind of aircraft. Five training patterns are generated for each aircraft by rotating and scaling the image, together with small local variations. A number of experiments have been made on classifying 40 (5 from each class) aircraft shapes into the eight class. The Table 1 illustrates the recognition accuracy versus learning sweeps.

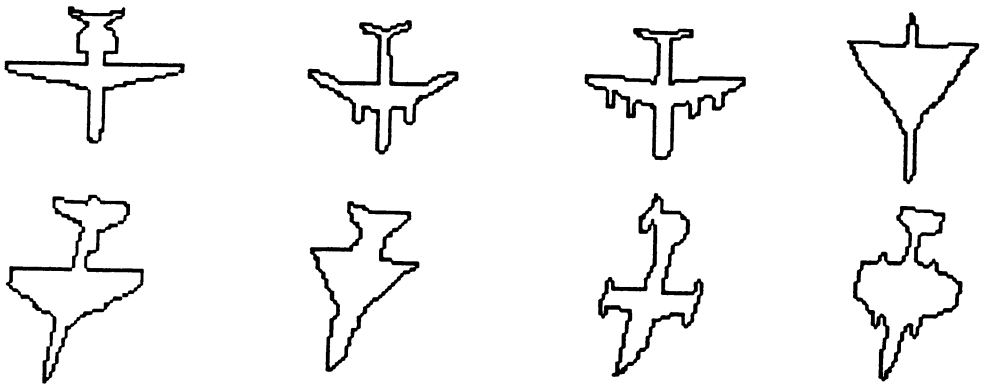


Figure 2: Eight types of aircraft used in simulation

Experiment I

Models	*Learning Sweeps*				
	1	5	10	15	20
HMM	90.0%	90.0%	92.5%	92.5%	97.5%
RHMM	97.5%	99.9%	99.9%	99.9%	99.9%

Experiment II

Models	*Learning Sweeps*				
	1	5	10	15	20
HMM	87.5%	87.5%	87.5%	87.5%	95.0%
RHMM	97.5%	97.5%	99.9%	99.9%	99.9%

Table1: Performance of HMM and RHMM (percentage of correct recognition)

The simulation results have shown that the RHMM approach offers a faster learning speed and higher recognition accuracy compared with the conventional HMM. Indeed the RHMM system can achieve almost 100% of the correct recognition rate by a small number of learning sweeps. We believe that the improved performance of the RHMM is due to the following reasons: (1) its ring structure can better incorporate the contour constraints; (2) the new learning algorithm uses reliable estimation of starting state to handle rotation invariance; (3) robustness are provided in probability distribution within each state to handle shapes with distortion.

4 Conclusion

In this paper a stochastic neural network for shape recognition which is invariant with scaling, rotation and distortion is proposed. The system uses a modified Ring Hidden Markov Model (RHMM) knowledge source. The proposed scoring and learning algorithm can capture the statistics of the local features and yet preserve the global constraints. Simulation results on aircraft recognition have shown superior performance in terms of speed and accuracy. Finally, the locally connected ring HMM network can be efficiently implemented in a systolic array which permit real time processing [3].

References

[1] H. Asada and M. Brady. The curvature primal sketch. *IEEE Trans. on Pattern Anal. Machine Intell.*, January 1986.

[2] X. Gong and N. K. Huang. Textured image recognition using hidden markov model. In *Proc. IEEE International Conf. on ASSP*, pages 1128–1131, New York, NY, 1988.

[3] J. N. Hwang, J. A. Vlontzos, and S. Y. Kung. A systolic neural network architecture for hidden markov models. *To appear in IEEE Trans. on ASSP*, 1989.

[4] B. H. Juang and L. R. Rabiner. Mixture autoregressive hidden markov models for speech signals. *IEEE Trans. on ASSP*, 33(6):1404–1413, December 1985.

[5] L. R. Rabiner and B. H. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, 3(1):4–16, January 1986.

[6] J. A. Vlontzos and S. Y. Kung. A hierarchical system for character recognition with stochastic knowledge representation. In *Proc.IEEE International Conf. on Neural Networks*, pages I601–I608, San Diego, CA, 1988.

Human Face Recognition Using A Multilayer Perceptron

John L. Perry and Jeanne M. Carney
ENSCO, Inc.
Springfield, Virginia 22151

Abstract

In this research we investigate using a multilayer perceptron (MLP) neural network for human face recognition. The results of our implementation demonstrates the ability to recognize faces independent of translation, rotation, and perspective transformations. In addition, the system is able to classify facial images that are corrupted with noise and with sections of the image removed. The recognition accuracy on our database of faces is 100%. We also compare our results with a conventional nonparametric classifier.

I. Introduction

Face recognition is a task which involves a variety of important aspects of perception and representation of complex visual patterns. People are capable of recognizing faces that are distorted, rotated, viewed from different angles and distances, and missing certain parts. The process of recognizing faces involves both the extraction of local features, such as the nose, eyes, lips, and eyebrows, as well as a global interpretation of these features, which integrates and views them as a whole.

A review of some of the pioneering research in face recognition is described in [1], [2], and [3]. The majority of face recognition research uses feature-based techniques. These techniques consist of three steps: 1) extracting facial features, 2) reducing the dimension of the feature vector set (if required), and 3) matching the reduced feature vector to a database of feature vectors using some type of distance metric that rates the quality of match. The research by Harmon [3] represents the embodiment of this approach for facial recognition. His research consists of extracting a set of 38 simple, local geometric features of a profiled face. From this set of features, Harmon uses principal component analysis to reduce the number of features to 17. The features are then matched to a database of similar features extracted from other individuals.

There are problems associated with feature-based methods. First, feature-based methods assume that segmentation of geometric features is always reliable and accurate. In reality, segmentation is generally very difficult, and segmentation inaccuracy will result in poor classifications. Further, even if it is possible to segment local features, it is difficult to organize this local information into a sensible global percept. Second, feature-based approaches are not robust to transformations. This results from a lack of reliable ways to encode a facial transformation, such as a smile or frown. The inaccuracy of encoding these transformations produces inaccuracies and inconsistencies in the associated feature values, which in turn cause classification mismatches.

In this report, we extend the research by using neural networks for face recognition. We address the problem of recognizing faces irrespective of translation, rotation, or perspective, as well as being tolerant to severe noise distortion and missing data. The specific neural network model that we have implemented in this report is a multilayer perceptron (MLP). There are three main reasons why we have implemented a MLP for face recognition. First, it can perform classifications in parallel. The ability to process in parallel allows the system to perform in a real-time environment. Second, a MLP does not make assumptions about the underlying distribution of the problem space. This is specifically applicable to the face recognition problem since it is difficult to characterize probability density functions (PDFs) accurately. Further, even well defined PDF modeling is seldom adhered to in a real world scenario. Third, a MLP has the ability to generalize. Generalization is the ability to classify on input patterns that the network has not seen before. In addition to being able to recognize human faces, we examine what effect variation in network architecture has on recognition accuracy. The

parameters that we vary include the number of hidden units, the learning rate, and the number of learning sweeps (epochs). The results derived here for face recognition also have relevance to many other vision applications.

This report is organized as follows. Section II presents the results of our implementation on actual faces. In Section III, we discuss the implementation of the face recognitio task using a conventional nonparametric classification technique. In Section IV, we present an analysis of our implementation and compare it with the nonparametric classifier. Finally, Section V provides a summary of our research and suggests avenues of further research.

II. Implementation

Description of Data Set: Our input nodes consist of raw grey level intensity information. We do not use facial features, as commonly done in most conventional approaches, but, rather, we allow the network itself to form its own set of "features" in the hidden layer. Our database set consists of six 482x499 grey level images. These images contain the faces of three people with their heads straight and tilted slightly to the left. From the six original digitized images we compose a data set of seventy-two images by applying various transformations to each image to simulate translation, rotation, and perspective changes. We also generate noisy images and images with missing data. After applying these twelve noise functions to the six images, we reduce the 482x499 images to 32x32 images. We reduce the images by implementing a consolidation operation which involves averaging the pixel values in a 16x16 window placed in overlapping positions across the entire image. The 1024 pixel values in each image are then normalized to the range [0.0,1.0].

Neural Network Architecture: We are using a three layer neural network. We vary the number of hidden units over the set of values {4, 8, 12, 16, 20, 24} and the learning rate over the set of values {0.1, 0.5, 0.9, 1.3, 1.7, 2.1}. We consider each person to be a separate class and use three output units to identify each class.

Training: To train our neural network, we use a momentum value of 0.9 and initialize the network's weights to random values ranging from -0.3 to 0.3. We randomly select a set of twelve face image patterns from the total set of seventy-two faces to serve as a test set, while the remaining sixty images are used to train the network.

Recognition: After training the network on sixty images, we present the network with the remaining twelve test images for classification. We use two figures of merit for recognition accuracy, as defined below:

> **0.8/0.2 Threshold Criteria:** For a given input class, if the activation level of the output processing unit which corresponds to the unit defined above to be 0.9 is 0.8 or more, and the activation levels of the other output processing units are 0.2 or less, the network is credited with a correct classification.
>
> **Greatest Activation Level Criteria:** For a given input class, if the network's output processing unit which has the greatest activation level corresponds to the unit which has the greatest desired activation level defined above, the network is credited with a correct classification.

III. Comparison to Non-Parametric Classifiers

The performance of the network was compared with a k-nearest neighbor (k-NN) classifier (k=1) which served as a performance benchmark. A nearest neighbor rule classifies an unknown signal as a nearest neighbor according to a pre-specified measure of similarity or distance. The k-NN classifier is useful for comparison against the performance of the MLP classifier since as the number of sample face images increases, k-NN approaches the optimal Bayes classifier. That is, with an unlimited number of samples, the k-NN error rate is never worse than twice the Bayes rate [4].

IV. Discussion of Results

Figure 1 graphically displays the results of using the MLP for face recognition. Figure 1a shows the network's recognition accuracy, using the Greatest Activation Level Criteria, at each of the hidden

unit and learning rate values we tested. As shown, a 100% recognition accuracy is obtained using many combinations of hidden units/learning rate values. The minimum number of hidden units required to give perfect recognition is 8 using a learning rate of 1.3. The lower bound on recognition accuracy is 83%, which occurs when using the {(4,1.3), (4,1.7), (4,2.1), (12,0.1)} hidden units/learning rate combinations, respectively. Figure 1b displays the network's recognition accuracy, using the 0.8/0.2 Threshold Criteria, at the same hidden unit/learning rate combinations. Again, a 100% recognition accuracy is obtained using the {(16,1.7), (24,2.1)} hidden units/learning rate combination. A decrease in the number of output processing unit activation levels able to yield 100% accuracy is noted, especially those produced in a network consisting of few hidden units. Figures 1c and 1d display the recognition accuracy percentage of the two classification criteria with the results of the learning rates averaged for each hidden unit. The Greatest Activation Level Criteria has a recognition accuracy average lower bound of 88.9%, while the 0.8/0.2 Threshold Criteria yields a 62.5% lower bound. It appears that recognition accuracy does increase with the number of hidden units up to a point. In some cases, however, going above this value even produces a slight decrease in recognition accuracy.

It is instructive to compare the nearest neighbor (k-NN) classifier to our implementation using a MLP. One comparison is how well the network performed compared to the k-NN in terms of recognition accuracy. The recognition accuracy for the k-NN on our test set is 100%, which is the same as the neural network implementation. The 100% recognition accuracy of the k-NN is based on choosing the distance metric that has the largest value, similar to the Greatest Activation Criteria. It is interesting to note that under different classification criteria, the k-NN technique is error prone. For example, if we analyze the out-of-class to within-class distribution we compute that 66% of the highest out-of-class value are greater than the lowest within class values. Further, if we use a criteria that all within-class values be above 0.8 and that all out-of-class values be below 0.2 (like the 0.8/0.2 Threshold Criteria), the k-NN recognition accuracy gives no correct classifications. The difference between these two approaches lies in the discriminating ability of the MLP. Even though both techniques give 100% recognition accuracy on our database using the Greatest Activation Criteria, the MLP is better able to discern between different classes. We expect that the MLP is better able to piecewise linear fit a more accurate decision surface between classes.

Another comparison between the MLP and the k-NN relates to the computational complexity of the two algorithms. For the MLP, we need to evaluate the space/time complexity for both training and recognition. The time complexity to train the network is $O(\alpha N_v N_h (N_i+N_o))$ where α is a constant relating to the number of training sweeps, N_v is the number of vectors in the database, N_h is the number of hidden nodes, N_i is the number of input nodes, and N_o is the number of output nodes. For recognition, the space/time complexity is $O(N_h(N_i+N_o))$. If we consider a trained network for our data set, this translates into 32,864 bytes of memory (using single precision) and on the order of 8216 multiply-adds. For the k-NN classifier, the space/time complexity is $O(N_v N_i)$. For our specific problem this translates into 245,760 bytes of memory and approximately 61,440 multiply-adds. In comparison, a k-NN requires 7.5 times more storage and computational resources than the MLP neural network approach. This difference gives the neural network a clear advantage in terms of a real time implementation.

V. Conclusions

In this study a MLP using a backpropagation learning algorithm is used to recognize faces irrespective of translation, rotation, and perspective transformations. In addition, the MLP is able to classify faces that are corrupted with noise and with sections of the face removed. Further, the MLP also demonstrates the capability to generalize.

The recognition accuracy on our database of faces is 100%, using a variety of hidden units/learning rate combinations. Recognition accuracy appears to increase with the number of hidden units up to a certain point. We achieve the same recognition accuracy using the k-NN classifier. However, the MLP appears to be better able to discriminate between classes, and it has an advantage in computational speed. Our future plans are to increase the size of the image database. This will allow us to obtain a more accurate estimate of the performance of the neural network approach and the relative comparison to the k-NN technique.

A major impediment to the analysis of the MLP's performance is the excessive amount of training time required. Though this is not a consideration for using a neural network in a deployable system, it does prohibit a timely analysis of intermediate results. With this in mind, we plan to port our simulation of the MLP from the Sun 3 workstation to a SIMD massively parallel processing environment. In particular, we are going to run our simulation on a Connection Machine-2 (CM-2) [5]. A properly designed implementation of our network on the CM-2 should result in a performance of 16×10^6 interconnections/second, as compared to approximately 34×10^3 interconnections/second on the Sun 3. This translates into a 470 times speed-up in performance. The advantage of this approach is that once we train a network, we can simply download the trained network onto a uniprocessor for the recognition phase. This will also allow us to experiment with different network designs in a more timely manner.

A variety of applications can prosper from the results of this research. An important application is in the area of security systems. Many portal entry security systems today utilize a series of redundant checks in verifying human entry, the two major checks being voice and signature. However, these systems are neither compositely nor individually fool proof and demand another level of verification. A face recognition system meets this requirement by being fast, robust, and easily adaptable to modification. A requirement for such a portal entry security system is a reliable method to actually locate human faces. In a recently developed system [6], researchers demonstrated the ability to locate human faces in photographs. They used a model-based technique and a semantic network representation to reason about locations of faces in photographs. Their results were rather impressive. The combination of this methodology for face location and the approach we have presented in this report certainly lays down the framework for building a practical, real-time security system.

VI. References

[1] A.J. Goldstein, L.D. Harmon, A.B. Lesk, "Identification of Human Faces", *Proceedings IEEE*, 59 (5), pp. 748-760, May 1971

[2] T. Kanade, **Computer Recognition of Human Faces**, Birkhauser Verlag, Basel and Stuttgart, 1977

[3] L.D. Harmon, M.K. Khan, R. Lasch, P.F. Ramig, "Machine Identification of Human Faces", *Pattern Recognition*, Vol. 13, No. 2, pp. 97-110, 1981

[4] R.O. Duda and P.E. Hart, **Pattern Classification and Scene Analysis**, Wiley, 1973

[5] *Model CM-2 Technical Summary*, Technical Report HA87-4, Thinking Machines Corporation, Cambridge, Massachusetts, April, 1987

[6] V. Govindaraju, D. B. Sher, R.K. Srihari, "Locating Human Faces in Newspaper Photographs", *IEEE Proceedings Computer Vision Pattern Recognition*, San Fransico, Ca., pp. 549-554, June, 1989

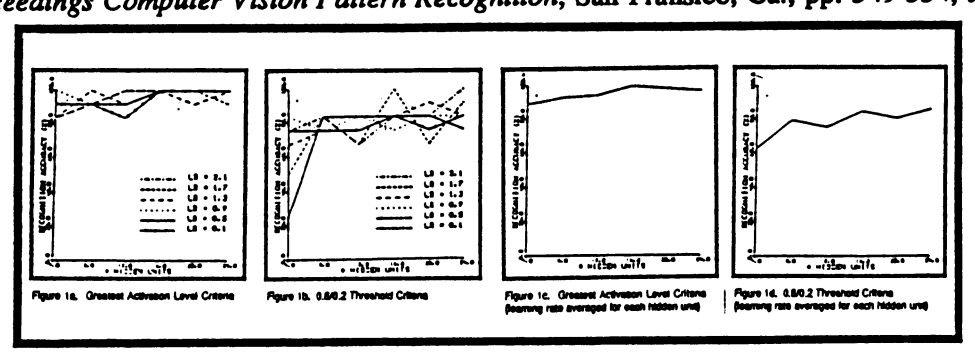


Figure 1. Recognition Accuracy of Multilayer Perceptron

An Application of Neural Networks to the Guidance of Free-Swimming Submersibles

D.P. Porcino
J.S. Collins
Engineering Dept., Royal Roads Military College, FMO Victoria B.C. Canada VOS 1B0

ABSTRACT

Most neural network models use *learning with a teacher* to train a great many simple processing elements. However the Adaptive Search Element/Adaptive Critical Element (ASE/ACE) model developed by Barto, Sutton, and Anderson, and elaborated on in this paper, is comprised of just two relatively complex units using *learning with a critic*. The ASE controls the physical system, and the ACE criticizes the performance of the ASE in an effort to accelerate learning. In this paper, the basic model is extended to the realm of complex numbers, and the output is time averaged, yielding a system which has a continuously valued control output. The system learns quite quickly.

In the simplest organisms possessing nervous nets, mechanisms exist to control reflex behaviours. In more complex organisms, these reflexes become hidden by the sophistication afforded by a more complex nervous system, but, they still exist. (Blackburn & Nguyen, 1988) In light of these facts, we will examine a most basic reflex behaviour: a reflex that keeps a submersible robot a certain distance above the ocean floor.

INTRODUCTION

A.H. Klopf suggested in 1982 that neurons could learn using operant conditioning: they would learn to attain certain states while avoiding others. In the field of neuronal modelling therefore, the principles of behavioural psychology provide a way to train unsupervised systems.

The basis for the ASE and the ACE is the "boxes" system of D. Michie and R.A. Chambers (1968). State space is partitioned into "boxes," each with its own resident "demon" which nondeterministically chooses a control action appropriate to that state. All the demons together form a metaphor for the input synapses of a single neuron.

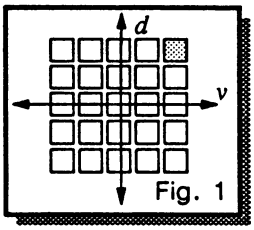


Fig. 1

The state space diagram in Figure 1 shows displacement versus vertical velocity partitioned into boxes. The shaded box shows velocity and displacement both large and positive. The demon within this box must choose an action that will be most likely to prevent a failure from happening.

Learning — adaptation — is based on an estimate of how long the system would continue to operate after particular actions. When the system fails, the individual estimates in each box must be updated to make another similar failure less likely.

Figure 2 shows the simulation's ASE/ACE controlling a submersible robot equipped with a thruster for forward motion, and up and down facing thrusters for maneuvering:

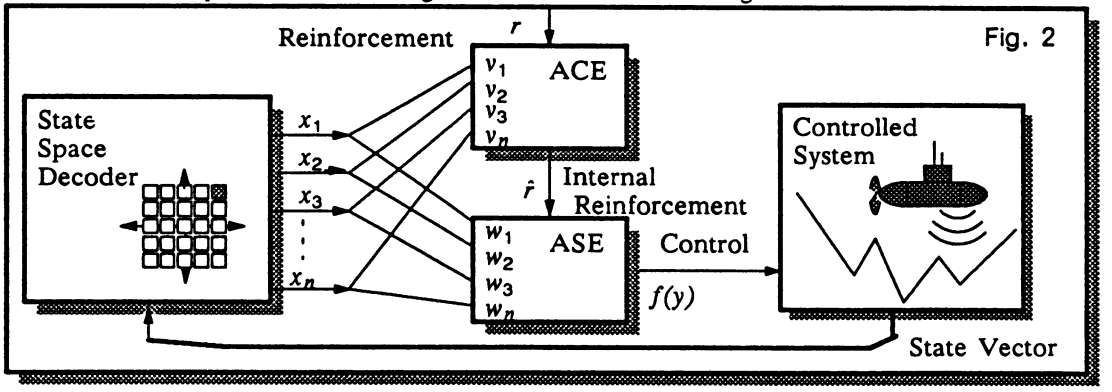


Fig. 2

The only information available to the robot is the return from its sonar, which gives the distance to the ocean floor (actually the displacement from the desired distance above the bottom), and a failure signal (reinforcement) which is supplied whenever the robot exceeds the allowable vertical error. The robot doesn't *know* what criteria generates the failure signal, and must work through trial and error to avoid the punishing failure signal.

DESCRIPTION OF THEORY AND APPLICATION - THE ADAPTIVE SEARCH ELEMENT

The state space decoder divides the state vector up — and generates therefore a unique and finite set of combinations. The element whose quantization parameters best match the state vector gives an output of one, and every other element outputs zero.

As in Figure 1, Figure 3 shows a state with a large positive velocity and displacement.

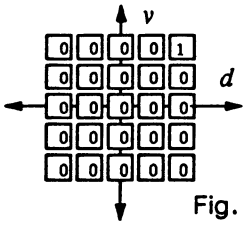


Fig. 3

Upon failure, every element of the decoder will be set to zero. This is an important condition for proper operation of the equations.

The output of the decoder, x, is passed to the ASE, each element being connected to a particular synapse, having a weight of w.

This diagram shows how the synaptic weights, w, might appear before much learning has taken place. Dark squares represent negative weights — or a high likelihood that system will decide to to thrust up, and light squares represent positive weights - the probability of thrusting down.

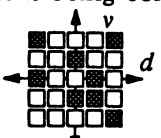


Fig. 4

The original model of Barto, Sutton, and Anderson used real synaptic weights to generate a binary output. In this version, complex weights consisting of a real and an imaginary part were used to allow a four valued output. A binary output allows up and down thrust — the complex four valued output allows the thrusters to be shut off as well.

The following equation describes the output of the ASE, $y(t)$, at time t:

$$y(t) = f\left[\sum_{i=1}^{n} w_i(t)x_i(t) + noise(t)\right] \qquad (1)$$

The addition of a *noise* signal having a zero-centered Gaussian distribution causes the ASE's output to be governed by chance.

The noise is important during early training to ensure that the system fully explores the state space. The noise should be *gradually* shut off to fine tune the weights, and *completely* shut off after training is complete to ensure consistent and repeatable behaviour.

$$f(x) = \begin{cases} +1, & \text{if } x \geq 0 \\ -1, & \text{if } x < 0 \end{cases} \qquad (2a)$$

$$f(x) = \begin{cases} +1, & \text{if } x_r > 0 \text{ and } x_i > 0 \\ -1, & \text{if } x_r < 0 \text{ and } x_i < 0 \\ 0 & \textit{otherwise} \end{cases} \qquad (2b)$$

The output of the ASE is passed through a thresholding function f. This quantizer provides the system's nonlinearity — utterly necessary for any computation. Equation (2a) describes the quantizer for real numbers, and (2b) shows the quantizer extended to use complex numbers.

The controller's most recent choices were most likely responsible for the current success or failure of the system, and therefore the most *eligible* to have their weights changed. A complex eligibility trace e decays at a rate governed by δ, $(0 <= \delta <= 1)$. The inclusion of the output of the ASE, term $y(t)$, means that eligibility will be positive decaying to zero if the ASE's output was positive, and negative decaying to zero otherwise.

$$e_i(t+1) = \delta e_i(t) + (1-\delta)y(t)x_i(t) \qquad (3)$$

The eligibility remembers how long ago a choice was made at a particular synapse, and what that choice was.

$$w_i(t+1) = w_i(t) + ar(t)e_i(t) \qquad (4)$$

This equation shows how weights change through time.

As shown in (4), the system learns through operant conditioning — setting the reinforcement signal r to -1 is an aversive stimulus — punishment. A positive value for reinforcement would be reward, or positive reinforcement. (The reinforcement signal as supplied to the ASE in our complex-valued model is real, but internally it is treated as $r+ri$.)

As long as the system is functioning properly, the reinforcement signal is held at zero, and *the weights do not change*. On the other hand, if a failure occurs, the negative value of the reinforcement will penalize the most recent decisions (highest eligibility synapses) which undoubtedly led to the failure, making those decisions less likely to occur in the future.

DESCRIPTION OF THEORY AND APPLICATION - THE ADAPTIVE CRITICAL ELEMENT

When failure finally occurs, one particular sequence of decisions would probably have been better than the sequence that occurred. Based on that, the system learns. Unfortunately, this learning only occurs upon failure — which is to say, infrequently. The ACE is added to provide a continuous feedback signal which is fed in to the ASE as $\hat{r}$. Reinforcement now arrives all the time — for good choices as well as for bad — so learning progresses at a *much* quicker rate.

The ACE receives the same decoder signal as the ASE, and also has a memory trace for every box, like the ASE. Rather than storing a best decision probability in each box, the ACE stores a prediction, $v(t)$, of the reinforcement the environment will provide for a particular state. Each synapse therefore contains a rating of that state's dangerousness — synapses corresponding to dangerous states will contain a strong prediction of failure.

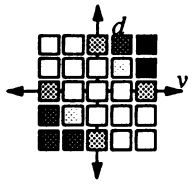


Fig. 5

Figure 5 shows what the ACE eventually learns about the submersible control problem – if displacement and velocity are both large and of the same sign, failure will soon occur. The white squares show where no reinforcement arrives from the environment, and the darker squares show that a failure signal is likely to occur very soon. This equation shows how the ACE makes its prediction:

$$p(t) = \sum_{i=1}^{n} v_i(t)x_i(t) \tag{5}$$

Where: v holds the ACE's internal weights, and x is the singleton output of the decoder. At failure the ACE's prediction is zero, as every decoder element will be zero.

The ACE sends the ASE the internal reinforcement signal:

$$\hat{r}(t) = r(t) + \gamma p(t) - p(t-1) \tag{6}$$

Upon failure, the output of the decoder, and consequently the ACE, $p(t)$, will be zero, but the external reinforcement, r, will be -1. (Usually r is zero.) The internal reinforcement $\hat{r}$ will be the difference between the previous reinforcement signal, $p(t-1)$, and r. *A fully predicted failure causes no reinforcement.* Incorrect actions leading up to the failure will be *punished* in accordance with how much the ACE's eligibility traces have decayed. The predictions of failure will be reinforced, because they were accurate predictions.

In the absence of external signals, the *discount* term, γ, causes the internal signals to gradually die away. The internal reinforcement, $\hat{r}$ is the *difference* between the *discounted* predicted reinforcement $\gamma p(t)$, and the previously predicted reinforcement, $p(t-1)$. Understanding (6)'s *difference* calculation is the key to understanding the whole operation:

If the system moves from a low danger state to a high danger state, the internal reinforcement will be negative, punishing the system. If, on the other hand, the system moves from a high danger state to a lower danger state, internal reinforcement will be positive: reward. From this information, the system learns to avoid dangerous states.

The equations governing learning in the ACE and the ASE are very similar:

$$v_i(t+1) = v_i(t) + \beta[r(t) + \gamma p(t) - p(t-1)]\overline{X}_i(t) \tag{7}$$

Notice that the term inside the square brackets is the same as the internal reinforcement signal $\hat{r}$ in (6). In form and substance, (7) is almost the same as the ASE's weight change (4), except for the reinforcement term, and the form of the eligibility, in this case, $\overline{X}_i(t)$. The ACE's eligibility trace equation is almost the same as the ASE's (3).

$$\overline{X}_i(t+1) = \lambda\overline{X}_i(t) + (1-\lambda)x_i(t) \tag{8}$$

THE SUBMERSIBLE MODEL AND SIMULATION RESULTS:

Simple equations describe the simulated physical system's kinetics. Friction was ignored, and the submersible was given a unit mass and neutral buoyancy. The horizontal component of the robot's velocity is held constant. State space is quantized into 64 boxes, as finer partitioning offers no particular advantage. Partitioning is exponential, giving finest control near the equilibrium point. If the system state moves into one of the extreme displacement partitions, a crash or surfacing (failure) has occurred, and the trial is terminated.

The ASE/ACE parameters were set as in Barto et al 1982, as these were quite reasonable. Real parameters were changed to complex with the imaginary and real components equal to each other. It is the interdependencies of the two components of an imaginary number as it undergoes manipulation that makes the solution robust and useful.

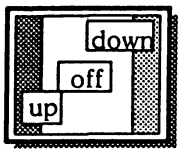


Fig. 6

The vertical thrusters (governed by the ASE's output) yield thrusts of -1, 0, or $+1$. The solution achieved by the system is quite interesting: The robot drifts back and forth within the safe corridor, thrusting only when necessary to stay within it.

The solution was refined by adding continuous control of the output through time averaging over four or five steps. The weights generated are more difficult to understand than those shown in Figure 6, as the solution is characterized by short, graduated thrusts which quickly return the system to an almost zero vertical velocity. Unlike the previous solution which had clear cut boundaries surrounding the decision regions, the time average solution has a scattering of different choices associated with each synapse. When averaged with neighbouring synapses, a graduated thrust results.

For use in practical situations, a successful solution should be *frozen* beforehand and the ACE removed so that the behaviour of the system is completely predictable in the field.

The ASE/ACE system learns quickly and generalizes readily to varying terrain. It was able to take advantage of the time averaging of the output. Figure 7 shows a number of trial runs. Truncated lines show where failures occurred.

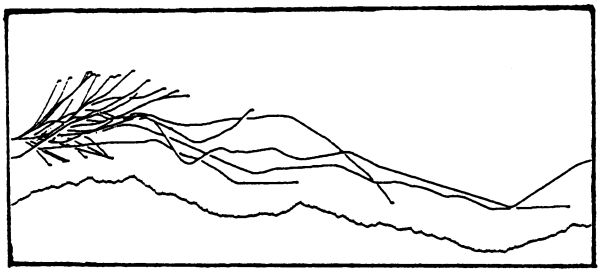


Fig. 7a
The earliest stages of training. The system makes many mistakes at first, but soon learns to negotiate the terrain.

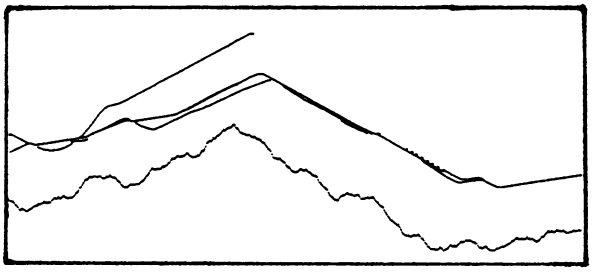


Fig. 7b
The next time interval, with new terrain supplied. The system fails once, then successfully negotiates the terrain.

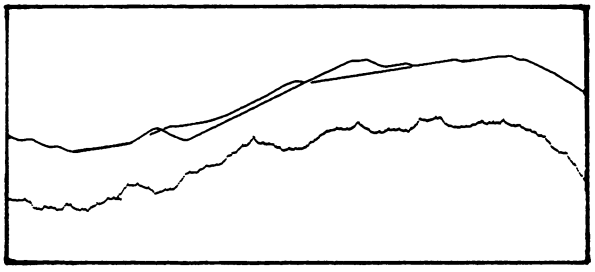


Fig. 7c
The terrain is changed again, and it is apparent that the system has generalized its solution, and can function well over any terrain.

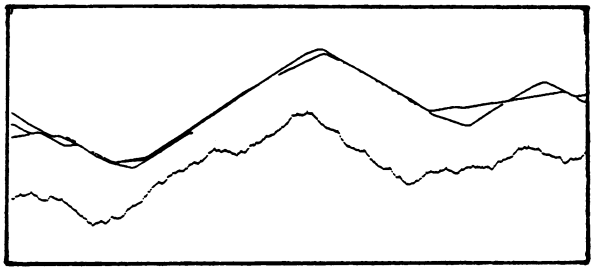


Fig. 7d
Much later in the training. The system has no problems with arbitrary terrain. Performance is qualitatively similar over all terrains, the solution is robust.

REFERENCES

Barto A.G., R.S.Sutton, & C.W. Anderson. **Neuronlike adaptive elements that can solve difficult learning control problems.** IEEE Transactions SMC, 1982. SMC-13: 834-846

Blackburn M.R., & H.G. Nguyen. **An Artificial Neural Network for Autonomous Undersea Vehicles.** Naval Ocean Systems Center, San Diego California. Technical document 1318, July 1988.

Klopf, A.H. **The Hedonistic Neuron: A Theory of Memory, Learning, and Intelligence.** Hemisphere Press, Washington DC, 1982.

Michie D., & R.A. Chambers. **BOXES: An experiment in adaptive control,** in *Machine Intelligence 2*, E. Dale and D. Mitchie, Eds. Oliver and Boyd, Edinburgh. pp. 137-152 (1968)

A Multilayered Neural Network to Determine the Orientation of an Object†

Morien W. Roberts, Mark W. Koch, and David R. Brown
Clarkson University
Department of Electrical and Computer Engineering
Potsdam, N.Y. 13676

Abstract

This paper describes a multilayered artificial neural network which is used to determine the orientation of an object. This network is a sub-network of a larger network capable of recognizing objects at any position and orientation, i.e. it is translation and rotation invariant. The sub-networks can be trained independently of each other resulting in faster learning times and the possibility of simulating larger networks.

Introduction

The process of computer vision is the inverse of computer graphics [PAV82]. In computer graphics the computer takes a description and produces the equivalent two-dimensional picture of the scene. Descriptions include the position and orientation of the objects, the type of three-dimensional objects, the material makeup of the objects, the observer's position, and the position and type of light sources. In computer vision, a machine receives a two-dimensional picture and produces a description of the scene. Many applications could make extensive use of a machine vision system with examples ranging from commercial robotics to government defense applications. Unfortunately, the intractable inverse process of transforming the two-dimensional picture into a description poses many difficulties for a machine vision designer. Relatively simple biological systems like the insect perform the vision task effortlessly. Our approach uses neural networks, inspired by biological computing structures, to help solve the difficult problems of computer vision. Our work involves the training of a neural network to recognize and locate objects that may appear in different positions and orientations. We assume only rigid planar objects can appear in the scene. The planar assumption can be side stepped by considering each finite stable state of the object as a different planar object. In this case, most three-dimensional objects can be described as a set of multiple planar objects. Many applications such as factory automation deal only with stamped or casted objects which do not violate the rigid object assumption. It is also assumed that multiple objects do not appear in the scene. In future work, we plan to investigate selective attention models to relax this constraint.

The Problems with a Brute Force Approach

By examining a brute force solution to the problem of recognizing and locating objects using a neural network, the main problems to overcome become apparent. A brute force solution would have a digitized image providing the inputs to the network, several internal layers processing the image, and an output layer encoding the object's position, orientation, and type. Current supervised learning techniques such as back-propagation [RUM86] require presentation of the entire object set many times before the network can discriminate between the objects. In addition, the objects need to be presented in all possible locations and orientations for the system to have rotation and translation invariance. This would make an already slow learning algorithm impractical.

Digitized images typically have a high data to information ratio. The Intelledex Intellevue™ 200, a commercial vision system for robotics applications, has 238x256 pixels with 64 gray levels. This picture carries over a quarter of a million bits of information and produces a neural network with many neurons and even more connections. A network of this size will increase the training time tremendously, since simulated network training times increase proportionally with neuron and connection numbers. A DARPA Neural Network study [DAR88] has shown that current neural network simulators would have a hard time handling networks of this size.

Solutions to a Brute Force Approach

Most researchers solve these problems by identifying certain object features as important to the object discrimination task. A neural network then trains on this reduced feature set. Yamada [YAM89] uses contour curvature as a feature vector to recognize handwritten numerals. Troxel [TRO88] recognizes trucks and tanks by preprocessing multi-function laser radar data using a Fourier/log-polar transform. The magnitude of the Fourier transform produces a position invariant image. By following this with a log-polar transform, variations in rotation or scale become shifts. The normalized peaks in the Fourier/log-polar space provide the input to a

†NSF grant No. EET-8806958 supports this work

neural network. Brousil [BRO67] has designed a perceptron network to perform the Fourier/log-polar transform. Unlike the above, our approach lets the vision system determine the best features for object recognition, and position and orientation determination.

Another way of solving the training problem is to develop neural network architectures invariant to the viewing perspective. This can be done by specialized neural network architectures and by putting constraints on the interconnection strengths. This approach allows a neural network to determine the best features for recognition.

Fukushima's [FUK87] "Neocognitron" uses a hierarchical type network and unsupervised learning to recognize translated and noisy characters. As one moves up the hierarchy, each simple/complex cell layer pair recognizes larger features with more and more inherent translational invariance. At the final output layer the grandmother cell represents a unique object regardless of any translations and small deformations. Fukushima uses selective attention to take care of multiple objects that may appear in the scene. Widrow [WID88] has developed a neural network architecture called the "invariance net" which has a response invariant to position and rotation. To achieve the position/orientation invariance the network has massive copies of weights for every up-down and left-right translation and every 90 degree rotation. The invariance net needs many weights. Widrow also discusses how to extend the network for invariance to scale and perspective.

Our Approach

Our approach consists of designing modular neural networks. Analogous to the process of coding a high level computer program, we subdivide the neural network into a group of networks (procedures) that can be trained (coded) independently of each other. After training and testing each module or sub-network, we can combine the modules to form the entire network (program). Several researchers have designed neural network modules that in combination can solve problems that individual modules could not [CAR87,MUR89,HIR89]. There are three major sub-networks in our architecture. These sub-networks are known as the translation, rotation and recognition networks. There are several advantages in using a collection of sub-networks in solving a problem. Each sub-network can be trained independently of the other, resulting in faster learning times, and the possibility of simulating larger sub-networks. Given sufficient computing resources the training of each sub-network can occur simultaneously resulting in a substantial improvement in development time.

Some problem dimensions: The image scene is a square array of M*M pixels. The training set consists of Q objects. The standard orientation of each object is such that the object fits entirely within an N*N region; known as the object field. The smallest region that will enclose all possible rotations of an object is a P*P square array; known as the object region. It follows that P = sqrt(2)*N.

The first sub-network is responsible for the task of producing an image which is invariant to translation. The M*M scene containing the object to be recognized is presented to the first sub-network. The output of this sub-network is a P*P object region. An object present at any position in the M*M scene will always be reproduced at a standard position in the P*P object region. The sub-network that performs this transformation is currently being developed.

The second sub-network is responsible for the task of producing a rotation invariant image. The P*P object region produced by the first sub-network is presented to the second sub-network. The output of this second sub-network is a N*N object field. An object present at any rotation in the P*P object region will always be produced at a standard orientation, of 0 degrees rotation, in the N*N object field.

The third sub-network is responsible for the task of recognizing an object. The N*N object field produced by the second sub-network is passed on to the third sub-network. The output of this third sub-network is a Q*1 array of neurons. Presenting an object from the set of Q known objects at the object field causes one, and only one, of the Q output neurons to be set.

The rotation network

The rotation sub-network itself consists of two sub-networks. The first of these detects the rotation of the object present in the object region. The output of this network together with the object region is used to produce a restored version of the object. The restored version is the object at the standard orientation of 0 degrees.

The scope of the problem was initially limited to the task of recognizing objects which have only four possible rotations. The strategy adopted in solving this problem can be easily extended when enlarging the scope to cover more rotations. A result of this limitation is that P = N, i.e. the object region is the same size as the object field.

The architecture of the rotation detection network consists of three layers of neurons. There are P*P

neurons in the first layer, 4 neurons in the hidden (middle) layer and 3 neurons in the output layer. The three output neurons produce a thermometer encoding which corresponds to the four possible rotations of the object. Each of the neurons in the first layer are connected to each of the four neurons in the second layer. The weights of the connections between the two layers are constrained so that the hidden nodes are 90 degree rotations of each other. For example, as illustrated in Fig. 1, the weight from $Pixel_{1,2}$ to hidden node 1 is constrained to the weights from $Pixel_{2,P}$ to hidden node 2, $Pixel_{P,P-1}$ to hidden node 3 and $Pixel_{P-1,1}$ to hidden node 4.

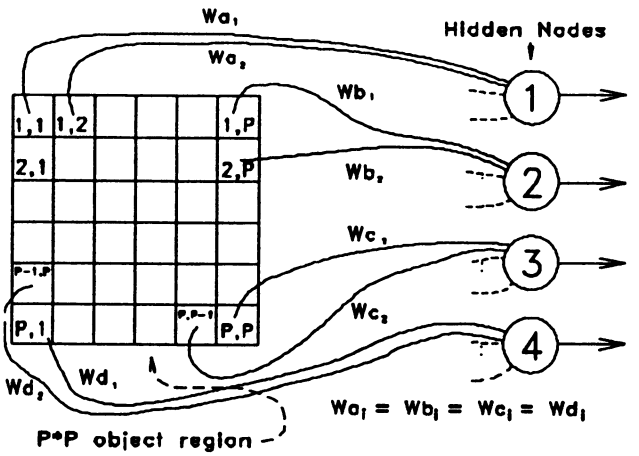


Fig. 1 Rotation detection network

A problem arose with objects symmetrical about one or more axis. It was found that if the training set contained no symmetrical objects then the trained network would generalize well for every unsymmetrical object in the test set but would perform poorly with any symmetrical objects. A training set containing both unsymmetrical and symmetrical objects did not help as the network would then perform poorly on the test sets, i.e. the ability to generalize had been lost. A possible solution to this problem is to break down the rotation detection network into two separate networks. One of these determines the rotation of symmetrical objects while the other detects the rotation of unsymmetrical objects. A third network determines which of these two networks's outputs feed the rotation restoration network. This third network is a symmetry detection network.

The rotation restoration network is conceptually the same as an N*N bank of multiplexor switches. In our limited scope of 4 possible rotations each of the multiplexors would have four inputs corresponding to the four related positions in the P*P object region. For example the multiplexor feeding $Pixel_{1,2}$ would have inputs from $Pixel_{1,2}$, $Pixel_{2,P}$, $Pixel_{P,P-1}$ and $Pixel_{P-1,1}$. Common to all the multiplexors are the three inputs which originate from the rotation detection networks. Depending on the values on these three inputs the multiplexors route the relevant pixel input to the output.

Standard orientation

Before the network could be trained it was necessary to determine the standard orientation for each object. This was achieved by selecting from the four possible rotations the one which maximized the number of pixels that were on in the top left hand corner of the P*P object region. It was discovered that while we could train a network to determine the rotation of our training set it did not fair too well when presented with a test set. In general, the network gave a unique thermometer output for all the rotations. However the standard orientation chosen often differed from the orientation that would have been selected using our criterion of maximizing top left hand corner pixels.

Instead of trying to analyze the scheme adopted by the network to select the standard orientation it was decided to use a more flexible approach. The approach is based on the observation that a network is better able to determine an object's rotation if no restrictions are placed on standard orientation. Such a network is capable of learning much faster and is capable of better generalization for objects not present in the test set. The redesigned network architecture automatically selects the object's standard orientation.

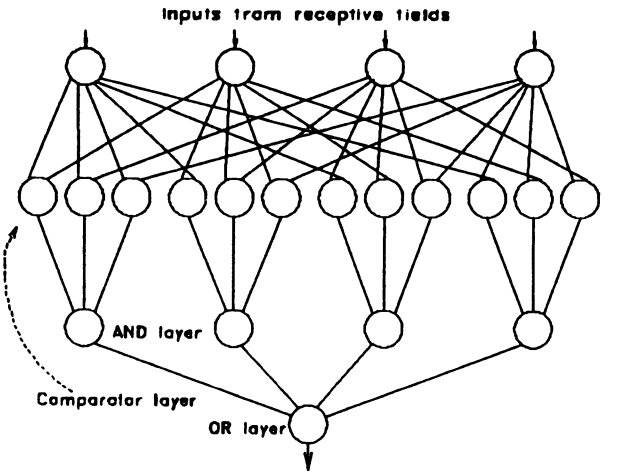


Fig. 2 Maximum network

Our initial approach was to use a "maximum network". This network is basically a noisy 1-out-of-4 code detector and its structure is shown in Fig. 2. The outputs of the rotation detection network's hidden layer supply the inputs to the maximum network. These four inputs are essentially the outputs of the same feature extractor which is applied to the object region at four 90 degree rotations. This maximum network replaces the three temperature

encoding output neurons from the rotation detection network. Its output is 1 provided that the feature extractor input with the largest value is at least a certain amount, β (typically 0.2), larger than every other of the inputs. For all other combinations of inputs the output of the network is 0. Fig. 2 shows the structure of this network. The 12 neurons in the second layer act as comparators between two first layer outputs. These comparators output a 1 provided that their two inputs differ by at least β. The neurons in the third layer are essentially 3-input AND gates and output of 1 provided that all three of their inputs are 1. The single neuron in the fourth layer acts like a 4-input OR gate. Each component of this maximum network is trained independently. Once trained the weights in the maximum network remain fixed even when the network it forms a part of, the rotation detection network, is being trained.

Results and discussion

The networks were trained on a training set of nine random shapes. After training the network were tested on a test set of about 90 random shapes. The network was deemed to have recognized the rotation of a shape if a unique 1-out-of-4 code was obtained by the feature extractors for each rotation of that shape. The experiments were repeated several times to remove the effects of the random initial weights. For the thermometer network about 10-30 thousand epochs (an epoch is one presentations of the set of test patterns) were required before a solution was obtained. The solution was fairly general, capable of determining the rotation of 75% of the random shapes.

Our initial experience with the maximum network was fairly mixed. On the plus side was the speed by which the network trained. The maximum part of the network took about 100-500 epochs to train while the full rotation detection network took about 200-500 epochs. This is much faster than the thermometer network. On the negative side is the capacity of this network to generalize, which averaged about 15% worse than the thermometer network. Two factors seem to play a role in this result:

i) Though the network selects its own standard orientation we have imposed our own structure on to the representation learnt by the receptive fields, namely a noisy 1-out-of-4 code. Of the numerous solutions obtained from the thermometer network only in one solution did the hidden layer learn the 1-out-of-4 representation. In all other solutions the representation learnt was a 2-out-of-4 code.

ii) Manual selection of the standard orientation allows the thermometer network to generalize better because the test shapes contain some intrinsic knowledge. As the standard orientation has been predetermined the network can concentrate solely on extracting features present in the standard orientation. On the other hand the maximum network has to both deduce the standard orientation and then extract features present in that orientation. A more realistic comparison is to compare the performance of the networks where no intrinsic knowledge is present. During experiments where the standard orientations were randomly preselected it was found that the generalization of the thermometer network was poor. The network was only able to determine the rotation of about 42% of the test set.

Perhaps the greatest promise of the maximum network is its training speed. This opens up the possibility of training the network on large training sets and hence improve the generalization of the network. We are currently investigating the effect that the 2-out-of-4 representation and the larger training set have on the generalization property.

References

[BRO67] Brousil and Smith, "A threshold logic network for shape invariance," *IEEE Transactions on Electronic Computers*, Vol. EC-16, 6, December 1967, pg. 818-28.

[CAR87] Carpenter and Grossberg, "ART 2: self-organization of stable category recognition codes for analog input patterns," *Applied Optics* Vol. 26, 23, December 1, 1987, pg. 4919-30.

[DAR88] *DARPA neural network study*, Executive Summary, Lincoln Laboratory, MIT, Lexington Institute of Technology, July 8, 1988.

[FUK87] Fukushima,"Neural network model for selective attention in visual pattern recognition and associative recall," *Applied Optics*, Vol. 26, 23, December 1, 1987, pg. 4985-92.

[HIR89] Hirsch, "Convergence in cascades of neural networks," *International Joint Conference on Neural Networks*, Washington DC, Volume I, June 18-22, 1989, pg. 207-8.

[MUR89] Murre, Phaf, Wolters, "Calm networks: a modular approach to supervised and unsupervised learning" *International Joint Conference on Neural Networks*, Washington DC, Volume I, June 18-22, 1989, pg. 649-56.

[PAV82] Pavlidis, "Algorithms for Graphics and Image Processing," Computer Science Press, 1982.

[RUM86] Rumelhart, D. E., Hinton, G. E., and McClelland, J. L., "Learning internal representation by error propagation," *Parallel Distributed Processing, Vol. 1: Foundations*, MIT Press, 1988.

[TRO88] Troxel, Rogers, and Kabrisky, "The use of neural networks in PSRI target recognition" *IEEE International Conference on Neural Networks*, San Diego California, Volume I, June 24-27, 1988, pg. 593-600.

[WID88] Widrow, and Winter, and Baxter, "Layered neural for pattern recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 36, 7, July, 1988, pg. 1109-18.

[YAM89] Yamada, Kami, Tsukumo and Temma, "Hand written numeral recognition by multi-layered neural network with improved learning algorithm," *International Joint Conference on Neural Networks*, Washington DC, Volume II, June 18-22, 1989, pg. 259-66.

LOCALLY OPTIMIZING NEURAL NETWORKS IN ADAPTIVE ROBOT PATH PLANNING

Frank Rudolph

Univ. of New Hampshire, Elect. Eng. Dept., 603-335-2693

E-mail: fjr@unhd.unh.edu

Reductionist methods in AI have attempted for almost 50 years to achieve successful and truly adaptive learning and problem solving abilities. Success has been demonstrated in specific problem areas. Game playing algorithms have reached an astonishing level of competency, and expert systems have eased the task of hand coding expert level bodies of rules, heuristics, and logically obscure lines of reasoning. True adaptive behavior, however, is still far from achieved, and in fact, may be theoretically impossible, using "traditional" AI techniques. These methods generally suffer from brittleness and computational complexity, even with the best weak methods of search reduction. But some simple neural like structures in software and hardware exhibit adaptive behavior, and are robust as well.

Path planning for robotics is a particularly thorny problem, in that it is difficult to define. Albus, et al, represent path planning as 3 levels: world model/task decomposition, (TASK), elemental move level, (E-MOVE), and primitive level, (PRIM). If one constructs a hierarchical model composed of these three modules, one will have all of the motor functions of the brain emulated above approximately the brain stem/spinal cord reflex level of a complex organism. The most primitive reflex or SERVO level, would then complete the system.

It's doubtful at the current level of artificial neural network development that the TASK or "intentional" level can be implemented using other than traditional AI techniques. Houk and Albus have each proposed cerebellar models of relevance to the lower and middle hierarchical levels. Houk talks about the cerebellum as an array of adjustable central pattern generators, (CPGs). This model gives insight into the nature of E-MOVE and PRIM level path planning activities. Albus has proposed and implemented a simple CMAC neural network model that has been developed and refined here at UNH. The CMAC model has proven itself to be extremely computationally efficient, exhibits adaptive properties, and seems an excellent adaptive element for implementing SERVO level control functions.

This paper will discuss path planning at a "modified" PRIM level. This is exclusive of "intentional" and SERVO level processing as described in NIST Tech. Note 1256, but would include some of the processing proposed therein at the E-MOVE level. Notably, the reverse kinematics and redundancy resolution would be handled in the proposed adaptive model.

A model is discussed herein that consists of hierarchically arranged CMAC elements, (as originally suggested by Albus), to implement a cerebellar model capable of computing relaxed spatial trajectories and using cerebellar habituation, (a biologically inspired paradigm), to adaptively learn to recognize world imposed constraints. Learning is via experience with a representation of such constraints embedded in a hyperspatial representation of a world model mapped directly into the robot's kinematic coordinate system. This hyperspatial representation of "analog" features will largely replace the symbolic state of the world model used in "traditional" AI systems which consist of discrete tokens embedded in a digital database.

The problem of specifying trajectories through a robot manipulator's world space involves computing a transformation from the manipulator's joint angle space into world space, so that predictions of hand movements can be made given joint angle changes. The problem is complicated though, by the fact that the problem presented to a robot path planner is rarely framed in terms of joint angle space, but rather in terms of world space, and if the manipulator is of the most useful type, it is "redundant". If redundant, its members can occupy many different regions of space for the same position of the hand, making operation in an "obstacle rich" environment possible. But such an arm has no unique transform in the reverse direction, (from world to joint space). Currently, path planners use "pseudoinverse" techniques, that find approximate matrix inverses to transform a world space incremental movement vector into a joint space incremental vector. Two problems arise in this process. First, the pseudoinverse is computationally expensive, being at best O(d**3) complex, where d is the square of the degrees of freedom. Secondly, it is not conservative, i.e., the transform of a "move" in world space to joint space and back to world space may diverge significantly.

Modern cerebellar function theory sees the cerebellum as a black box containing fixed action pattern "subroutines". Houk (1988) characterizes the cerebellum as an array of adjustable CPGs. At the level of a path

planner, it would be a huge computational advantage to have a cerebellar-like black box that we could rely on to produce path segments, on command, that can go from point to point, in hand coordinates, and that would have an implicit model of the state of the world built into it. The ability to store and reconstruct these segments is a model of habituation.

Previous efforts to do adaptive neural based robot control include the work of Mel at Northwestern and Jordan at MIT. They have shown good success for classes of problems useful in robot path planning for redundant manipulators. In both cases, they use "flailing about" in order to learn. I.e. if you don't know how to do something definite, try something random and perhaps a behavior will emerge that approaches what you wanted in the first place. Though this is advantageous for generating "relaxed", (non-optimal energy), paths, it is in fact theoretically not even algorithmic, in that it isn't guaranteed to produce a putative move that is acceptable in finite time! (In practical terms, it only means it is inefficient during **early** learning.) The hope here is that by using a locally optimizing neural network model, the chances for real time operation improve.

In our work, an effective fast means of computing the reverse kinematic transform, (RKT), for a redundant robot arm will be achieved. Proven, though computationally expensive, pseudo inverse techniques mentioned before, could be used as the teacher, in lieu of, or in addition to, "flailing about" The teacher's role would decrease rapidly, and so the system would learn over time to compute the transform for previously visited parts of the arm's state space in a manner that would improve in time, and would improve from worst case of O(d**3) to O(g), where g, (generalization size), is a small parameter of locality. Practically, O(g) is of constant time complexity!

The objective of this research is not a single applicable means of computing arm trajectories, but rather a continuum of trajectory planners that can produce a variety of trajectory types from tightly constrained optimal trajectories, in which PRIM assumes no obstacles, and obstacle avoidance is resolved via a higher level AI type path planner, to a planner that computes very relaxed trajectories, in which part of the E-MOVE obstacle avoidance and path planning is achieved by the neural net based PRIM module. In the relaxed trajectory mode, the planner can incorporate some of the robot's world model into the RKT itself rather than computing the transform based on ideal minimum travel or minimum energy point to point trajectories. It is proposed here that "early learning" would rely more heavily on a pseudo inverse teacher for the constrained optimal energy model, and more on "flailing" for the unconstrained model.

A rule based or other supervisory source acting from above, (at the "intentional" level), for postulating gross plans in the forms of sequences of trajectories, is assumed and our system concentrates on fast adaptive performance of only PRIM level processing to resolve the individual trajectories in world based terms into implementable joint space trajectories.

A preliminary model exists via simulation, that uses one CMAC to learn relaxed trajectories through space without the aid of a non-adaptive underlying controller. The currently implemented pilot model suggests that CMACs can in fact learn these loosely constrained quantities fast enough to be practical adaptive devices for such an application. Extension of the model can now be pursued using a "real" robot to show the applicability of neural elements like CMACs for replacing AI methods at the PRIM level of robot path planning.

The currently completed program simulates a 2D planar, minimally redundant arm. The three joint angles are α, β, and γ. The forward kinematic transform (FKT) is straightforward,

$$x = T * \cos(\alpha) - H * \cos(\alpha+\beta) + F * \cos(\alpha+\beta+\gamma)$$
$$y = T * \sin(\alpha) - H * \sin(\alpha+\beta) + F * \sin(\alpha+\beta+\gamma).$$

Where F = forearm, H = humerus, and T = torso. No attempt to model the kinetic response of this robot arm was made. The kinematic response is suitably non linear, to demonstrate that CMAC is a sufficiently powerful computational model to solve a relatively hard supervised learning problem.

The CMAC was viewed as a mapping: $A : (\alpha,\beta,\gamma,\delta x,\delta y) \rightarrow (\delta\alpha,\delta\beta,\delta\gamma)$, where all are the obvious quantities, except, δx, δy, which will be treated as unit vectors along a **desired** trajectory. The lookup steps will be provided with the desired trajectory along with the current joint position in order to excite a response, and the learning will train in the observed response at the **actual** observed trajectory step. Forcing the steps to be unit vectors results in an attempt at a constant velocity solution, and simplifies the input address space. For true obstacle constraints, the input address will probably need a new degree of freedom to be able to distinguish the difference between 2 target points along a straight line separated by an obstacle.

The primary "weak gradient" heuristic rule is: If the CMAC returns an RKT = (0,0,0), try a random step, $\underline{R}$, and observe the result, $\underline{S}^* = \underline{S} + FKT(\underline{R})$. There are 2 possibilities: the step generated an $\underline{S}_{x,y} = (x,y)$ component such that the dot product $\underline{S}_{x,y} \cdot \underline{S}^*_{x,y} < 0$ or $> = 0$. In either case, train on the step, (i.e. train the CMAC at $\underline{S}^*$, not at $\underline{S}$), but for the former case, don't take the step, because it moves the hand in the wrong direction.

By judicious adjustment of the following parameters: physical memory size, virtual memory size, unstructured map random index shuffling algorithm, resolution of joint angle measurement, resolution units of δx, δy, η, (learning rate), and generalization size, a compromise model that could operate well in most of its address space was realized. Some tendency towards slow learning or spastic activity was observed, and some heuristic remedies investigated.

The training algorithm is $\delta = \eta/C(\underline{S} - \underline{S}^*)$

The following extensions are proposed to complete the project:

1. Add noise when local minima are reached or learning slows to unacceptable rates, ala Boltzmann.
2. Exploit synergies in the robot: In a fashion similar to Hinton's "stick figure balancing heuristics", apply rules concerning how the joints in a manipulator cooperate or compete in selecting modes of behavior. This should speed learning and reduce the possibility of the spontaneous emergence of undesired alternate solutions.
3. Add models of physical constraints to force the CMAC to mimic a body of AI rules by exhibiting habituation.
4. Add a 6th degree of freedom.

The resolution levels were set using a rule given in the seminal Miller/Glanz/Kraft paper (1986), that states a relationship between resolution and generalization size. The summed absolute differences of the components of input state vectors must be less than C if those two vectors are to generalize. This differential quantity was determined empirically, by having the control program monitor the differential sum during several experimental runs.

An interesting observation made was that the experimental system was very sensitive to over- as well as under-generalization. Good results were impossible from both C = 32 and C = 128. The former learned too slowly, and the latter was unstable in its learned data. 256 was tried and showed even more instability. This can perhaps be explained as the result of retroactive inhibition due to the structured map rather than the unstructured map.

In practical CMAC implementations, consider the possibility of a variable learning rate, η. This could take the form of an adaptive damping function to smooth out instability caused by regions of noisy input, and still allow for fast convergence rates, while $\eta = 1.0$, everywhere else. One promising approach to this is the construction of an "aging" CMAC, which has coarse resolution and is present strictly to monitor the relative age of the data in the control CMAC. As the data ages, (is trained repetitively), the aging CMAC will adjust down the learning rate in a fashion similar to, but more locally specific than, some well known implementations of Kohonen map training algorithms.

Variable η could be coupled with a focusing of the generalization window. Rather than having a generalization window that specifies that the delta value learned be uniformly applied at each component point of excitation comprising the current receptive field, it would be desirable to bias the delta to have more effect (i.e. a larger value of η), for points near the center of the receptive field, than for points on the periphery of the field. This sounds easier than it really is, because in the current CMAC, there is no easily calculable and reliable metric for measuring how far any constituent receptive point is from the "center" of its receptive field. All we have is the membership roll for any receptive field. By the time we get to the computation of delta, the addresses are likely in terms of the A' memory. Ranking the indices of the array giving the membership roll of the constituent points ranked according to a euclidean distance formula in the A memory would inject a high computational penalty!

Observations bore out that generalization didn't seem to be uniform. Since no effort to come up with a tailor made structured map was undertaken, this wasn't surprising at all. The paradigmatic CMAC structured address map elects a receptive field for an input from sequence of points in the input space such that each address coordinate is incremented C times, which tends to smear the receptive field along diagonals in the hyperspatial memory. A recent idea we have started exploring at UNH to avoid this problem is to use a locally tuning neural network, like a Kohonen map to implement the front end (structured) address map, rather than the usual algorithmic front end. There would be a slight computational penalty in doing this, but the benefits of better fault tolerance and the inherent presence of a distance metric within the receptive field would likely be a great benefit.

Another problem that concerned us was that of the redundancy of the inverse solution. In numerous sources,

notably Atkeson and Jordan, reference has been made to a commonplace problem in redundant manipulator control that the average of two good solutions is not necessarily a solution. Concern over that prompted the implementation of an added heuristic level in the learning step of the program. In the event a point is reached when all other indications favored training the CMAC at a particular point, before training, the CMAC is excited and read, to see what response is already stored there. If the dot product of the response to be trained in at this point with what's already there is 0, don't train! This dot product is in joint coordinate space by virtue of being a product of incremental joint commands. A dot product constraint has already been imposed on the system in hand coordinates by requiring the weak gradient descent described before.

There is a "catch 22" problem with the dot product heuristic in that if it is enabled, and a hashing collision injects a contrary response vector into a state space location, that invalid response won't go away, because this dot product rule won't allow training to override it!

Observations showed that areas of learning defects seemed to cluster around the obstacles, indicating that the removal of an active address coordinate from the state space response may herald the coming of some spastic responses, and in such cases, the dot product heuristic should be disabled. This is desirable in fact because if a new obstacle enters the workspace, the robot very definitely may encounter instances of needing to reverse stored response vectors, and the enabled dot product heuristic would effectively block such recovery. It was observed, by manually enabling or disabling the two dot product heuristics, the catch 22 could be circumvented, while still leaving the dot product heuristics on most of the time to prevent undesirable spontaneous path modality changes. It may be that rules can be imposed, allowing the system to automatically enable or disable these heuristics at various times. An obvious one is "if the percent of steps that are random increases above some threshold, disable the dot product heuristic". The manual operation results were promising, but since the only obstacles observed so far have been boundary conditions and these don't appear and disappear, the results are still somewhat inconclusive.

Although this project is framed as one in robotics, it should not be missed that this is a widely applicable methodology. There exist a huge, if not infinite, number of applications in industry, science, business, and elsewhere where processes go on that have matrix space definitions, and in which an optimal setting for a set of input parameters must be known in advance to assure the success of the process. Typically the forward process transform is easy, but the solution of the matrix equations for the reverse transform (which would allow you to set those input parameters, a priori, to guarantee some desired output response), can't be found because of the existence of non invertible matrices in the system equations. This adaptive method of learning a representation of a reverse transform may be of wide and profound interest to other industries.

Abbreviated Reference List:

James Albus, Brains, Behavior and Robotics, McGraw Hill, 1981.

James C. Houk, et al, "An Adaptive Sensorimotor Network Inspired by the Anatomy and Physiology of the Cerebellum", NSF Workshop: Application of Neural Networks to Robotics and Control, MIT Press, 1989, (in press).

Bartlett W. Mel, "MURPHY: A Neurally-Inspired Connectionist Approach to Learning and Performance in Vision-Based Robot Motion Planning.", Tech. Report CCSR-89-17A, Center for Complex Systems Research, Univ. of Ill., Urbana-Champaign.

W. Thomas Miller III, Filson H. Glanz, and L. Gordon Kraft, III: "Application of a General Learning Algorithm To the Control Of Robotic Manipulators", Internal paper of UNH ECE Dept. 1986, (NSF Grant ECS-8405701)

Michael I. Jordan, "Supervised learning and systems with excess degrees of freedom", U. Mass. Amherst, COINS Tech. Report 88-27, May 1988.

Wavering, Albert J., "Manipulator Primitive Level Task Decomposition", NIST Technical Note 1256, U.S. Dept. of Commerce, NIST, Oct. 1988.

Rumelhart and McClelland et al, Parallel Distributed Processing, vol. I, MIT Press, 1986.

Data Expressions Suitable for Size- and Rotation-Invariant Pattern Classification

Kazutaka Sakita Makoto Kosugi Isamu Yoroizawa
NTT Human Interface Laboratories
1-2356 Take, Yokosuka-shi, Kanagawa 238-03 JAPAN

Abstract- The systems that can recognize and understand some figures or scenes must have abilities of size- and rotation- invariant pattern classification. Invariant pattern classification models proposed before, however, have some difficulties in real applications in spite of their usefulness in principle. To overcome these difficulties, a PDP model in combination with data expressions for invariant pattern classification is introduced. This model is used to classify some simple figures.

1. Introduction

Although computer technology has made remarkable progress, a computer's capabilities in pattern processing have made rather slow progress and have fallen behind human abilities. In order to implement more intelligent machines, it is necessary to refer to the manner in which human beings process patterns[1]. One of the manners is parallel processing, and another is adaptable processing. Although PDP models[2] are advantageous in processing data in those manners, the perceptron type PDP model's capability for shift-, size- and rotation- invarient pattern classification is poor[3]. Therefore, it is important to select the proper input data expression when PDP models are applied to the pattern classification problem.

A model based on log-polar coordinate transform[4] and translation-invarient transform[5] is used to choose a proper data selection.

2. Approaches to the size- and rotation- invariance problem

The retinal mapping to the striate cortex of some mammls can be expressed by a space-variant cortical magnification factor CM. Under the assumption that CM is locally isotropic and inversly proportional to the retinal eccentricity, the retinal projection to area 17 can be approximated by a log-polar coordinate tranform[4]. A log-polar coordinate transform converts scaling and rotation into translations, if the objective image is centered in the coordinates. The R-transform[6] is applied

to subsequent procedures to pick up translation-invariant data expressions. The R-transform is a class of translation-invariant transform, and its algorithm is similar to a flow graph of the FFT, and is represented by a layered network. A unit in the network computes only the sum and the absolute difference of the two values. As a result of the log-polar coordinate transform and the R-transform mentioned above, a size- and rotation-invariant pattern expressions is obtained.

In a similar model that has been reported[7], patterns are classified by the normalized distances between two vectors correspond to the patterns. However, it is sometimes difficult to classify patterns, because of deviations due to low resolution and to special features of the R-transform. For instance, normalized distances shown in Fig.2 correspond to size scale variations of the samples shown in Fig.1. It is found that classification is difficult if the scale factor is more than 1.3 (image resolution is 64 x 64 and pixel value is binary in Fig.1). To overcome these difficulties, a PDP model in combination with data expressions for invariant pattern classification is introduced.

3. Robust method for pattern classification by applying PDP model

To make the model more robust for pattern classification, it is necessary to correct vectors corresponding to R-transformed patterns separate in the vector space. A perceptron type PDP model subsequent to the R-transform is applied. This kind of model is suitable for this purpose because it has the capability to adaptively simulate arbitral mapping from sets of input/output data. Moreover, we composed the PDP model including a mapping function, which is suitable for the correction of vectors, because this new model can make its structure in a short time compared with the existent method in which the initial value of each unit is set at random and is learned repeatedly[8]. A three layered PDP model that has the above mentioned structure at hidden layer units is composed using the following procedure.

1. Vectors which represent given patterns well are picked up and are regarded as anchor points from the distribution of data in the vector space.

2. All points in the vector space are projected to the surface of a hyper sphere by one dimension in order to set all vectors and anchor points on the surface of the same hyper sphere. Thus, the anchor points are regarded as hidden layer units, and coordinate values of the anchor points are also regarded as weights of linkages from input layer to hidden layer.

3. The hidden layer has a threshold function or sigmoid function. Parameters of these functions are defined so that linear combination of the hidden layer's output is the output of the proposed model.

sample1 sample2 sample3

Fig.1 Sample Figures

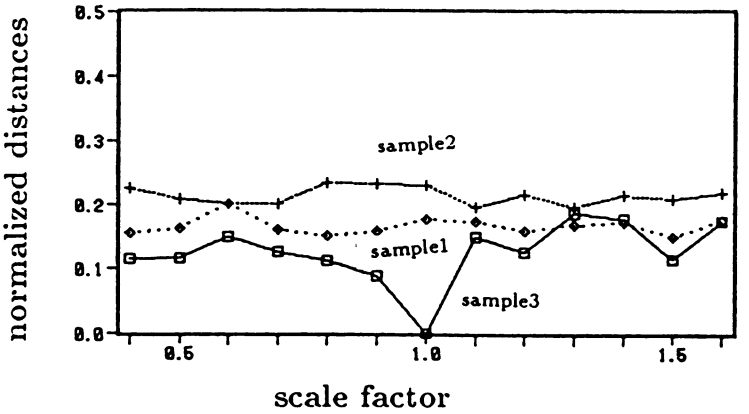


FIG.2 Distances between Vector obtained from Sample and Reference Vector

(reference vector corresponds to sample3 with scale factor 1.0)

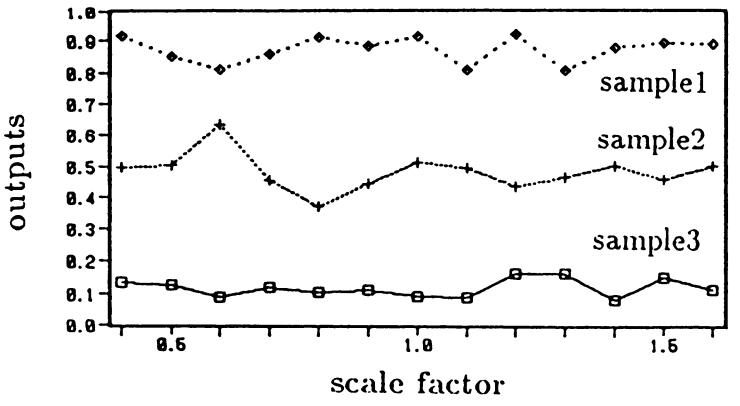


Fig.3 Outputs of proposed Model

4. The input data for learning the model are arbitrarily selected from some vectors in the vector space where R-transformed patterns exist. Thus, hidden layer outputs are obtained. Moreover, corresponding desirable data are inversly transformed by the output layer by adopting a method of least squares for a set of output data of the hidden layer and inversely transformed data by the output layer.

According to these procedures, the model outputs are shown in Fig.3 where the desirable values of the three sample figures shown in Fig.1 are set to 0.1, 0.5 and 0.9 respectively. As shown in Fig.3, the model outputs corresponding to size scale variations of a sample figure are distributed separately from those of another, therefore the proposed method can use the model outputs as better classification criterion than the existing method which classify patterns only by vector distances obtained from the R-transformed pattern.

4. Conclusion

A size- and rotation- invariant pattern classification model is proposed by combining log-polar coordinate tranform and translation-invariant transform. The model is tested using some simple figures, and is shown to have better classification criterion than the existing method which classify patterns only by vector distances.

References

1. K.Watanabe, I.Yoroizawa and M.Kosugi: "A proposal of human interface architecture for advanced information processing systems", HCI International '89 (1989).
2. D.E.Rumelhart and J.L.McClelland: "Parallel distributed processing Vol.1, MIT Press, Cambridge (1986).
3. H.Midorikawa: "The face pattern identification by back-propagation learning procedure", INNS 1st Annual Meeting, p.515 (1988).
4. E.L.Schwartz: "Computatinal anatomy and functional architecture of striate cortex", Vision Res.20, pp.645-669, (1980).
5. H.Burkhardt and X.Muller: "On invarient sets of a certain class of fast translation-invarient transforms", IEEE Trans.ASSP-28, pp.517-523, (1980).
6. H.J.Reitboec and T.P.Brody: "A transformation with invarience under cyclic permutation for applications in pattern recognition", Inf.Control 15, pp.130-154, (1969).
7. H.J.Reitboeck and J.Altmann: "A model for size- and rotation-invarient pattern processing in the visual system", Biol.Cybern.51,pp.113-121,(1984).
8. H.Kawahara and T.Irino: "A procedure for designing 3-layer neural networks which approximate arbitrary continuous mapping: Applications to pattern processing", IEICE Vol.PRU88-54, (1988) (in Japanese).

SIPS - II :

A Spatial Information Processing System on Perceptual Grouping

Dr. Chen-Han Sung
An-Hoang Nguyen

Center for Artificial Intelligence and Neural Networks
San Diego State University
6330 Alvarado Court, Suite 203, San Diego, California 92120, U.S.A.

The proposed spatial information processing system (**SIPS**) is a self-organizing multi layer neural network paradigm modeled by the neural dynamics on boundary feature perception, brightness perception, and color perception, of three-dimensional real world images. As an image processing system, SIPS at low level extracts features from spatial information stored in the image for the high level part of SIPS to recognize objects in the image and/or to perform scene analysis. We believe that human usually isolates sighted object boundaries before recognizing them. In this paper, we only concern the performance of SIPS on segmentation and boundary contour detection. Without involving any domain specific knowledge, the perceptual grouping principles used in SIPS are domain independent. They are based on the characteristics such as continuity, curvilinearity, collinearity, proximity, color, and other phenomena of boundary feature in images, as well as on the brightness constancy, brightness contrast, brightness assimilation, and other brightness phenomena in images. Detail discussion on theory, performance and limitation of SIPS is in [8].

In [7], we present the performance of SIPS-I which is based on the first order characteristics of boundary feature in images, the perception of continuously shaded surface, and other surface perception such as illuminants and shape-from-shading. In [7] and [8], simulation results of SIPS-I has been compared with the Boundar Contour Systems ([4], [5], [6]) developed by Grossberg and his colleague. In this paper, we present some simultation results of SIPS-II which is based on the second order characteristics and proximity of boundary feature in images as well as other perceptual grouping such as textures, boundaries, and emergent segmentations. In [8], we first analyze the neural network dynamical equations at each level of these two SIPSs to present their structures, data flows and parameter functions, then discuss how these models are consistent with neurophysiological data and compatible with some components of human vision system.

In our computer simulations, we first demonstrate how SIPSs work on 16x16 idealized images such as Figure 1 - 3 below with the left one being the input and the right one being the output of SIPS-II. These show the capability of SIPS-II to handle perceptual organization rules on the extraction and grouping of curved boundaries and contours, brightness constancy, brightness contrast, and boundary completion. We then show some results of SIPS-II applied to real world images with much more complicated structures and in much larger size. In Figure 4, the boundary completion and extraction of fingers, hairline, eyes of Iceman are well done. In Figure 5, it is clear that SIPS-II can differentiate noise from important tiny data areas even they have about the same grey level, as noise around the face are suppressed but eyeball outline is kept and nose profile is completed. In Figure 6, the level contours on Cheryl's face and hair are well grouped or merged.

In Table 1, we list the CPU seconds for each image taken by the SIPS-II which is coded in "C" on a UNIX-based 5-MIPS workstation from our elaborated tests. SIPS-II can be easily implemented on a parallel-pipeline computer or in microprocessor chips on array processor boards due to its architecture design [8]. Under those circumstances, the execution time for SIPS-II on those images in Figure 4 - 6 can be further reduced to less than one second, which is near the so called "real-time" range. The processing speed and the quality of the output which can be seen in the Table 1 and Figure 1 - 6, are better than known segmenters and boundary contour detectors based on other neural network architectures (such as [1], [2], [3]) for perceptual grouping.

REFERENCES

[1] D. Diamond and D. Holden, "Implementing the Boundary Contour System on a Multi Vector Processing iPSC/2 HyperCube", 1st International Joint Conference on Neural Networks, Washington D.C., I, 135-143, June 1989.

[2] G. Carpenter, S. Grossberg, and C. Mehanian, " Invariant Recognition of Cluttered Scenes by a Self-Organizing Art Architecture, I: CORT-X Boundary Segmentation", Technical Report, Boston University, Nov. 1988.

[3] S. Grossberg, and E. Mingolla, "Neural Dynamics of Perceptual Grouping: Textures, Boundaries, and Emergent Segmentation", Perception and Psychophysics, 38, 141-171, 1985.

[4] S. Grossberg, and D. Todorovic, " A Neural Network Architecture for Brightness Perception under Constant and Variable Illumination Conditions", International Conference on Neural Networks, San Diego, California, IV, 185-192, June 1987.

[5] S. Grossberg, and D. Todorovic, " Neural Dynamics of 1-D and 2-D Brightness Perception: A Unified Model of Classical and Recent Phenomena", Neural Networks and Natural Intelligence, Cambridge, MIT Press, 1988, 128-189.

[6] S. Grossberg, and E. Mingolla, "Computer Simulation of Neural Networks for Perceptual Psychology", Neural Networks and Natural Intelligence, Cambridge, MIT Press, 1988, 195-211.

[7] C.-H. Sung and A.-H. Nguyen, "SIPS-I: Spatial Information Processing System on Surface Perception", Technical Report, Center for Artificial Intelligence and Neural Networks, San Diego, March 1989 (also submitted for publication).

[8] C.-H. Sung and A.-H. Nguyen, "Spatial Information Processing System", Technical Report, Center for Artificial Intelligence and Neural Networks, San Diego, May 1989 (to be submitted for publication).

FIGURES and TABLE

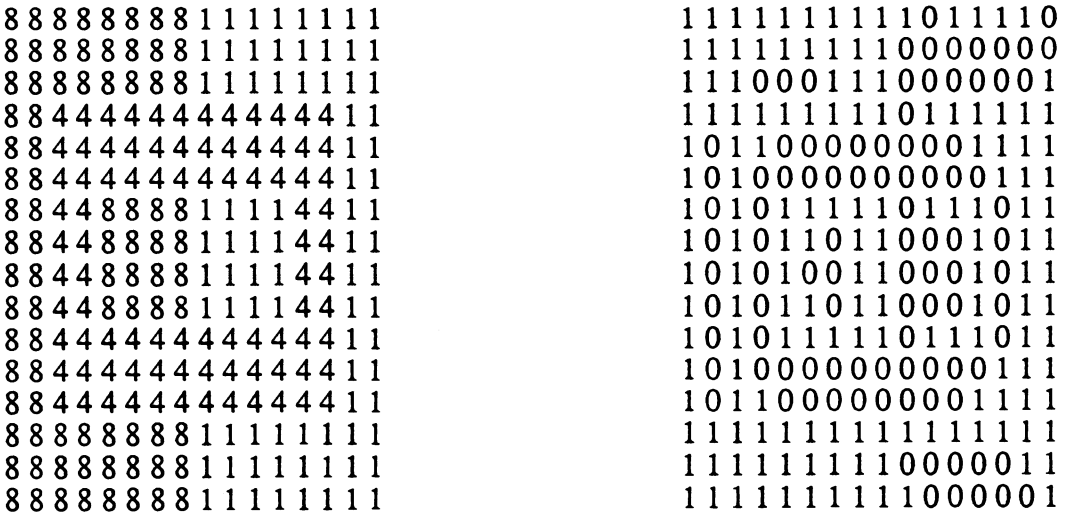


Figure 1: The Koffka-Benussi Ring.

9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
3	3	3	3	3	3	3	3	9	9	9	9	9	9	9	9
3	3	3	3	3	3	3	3	9	9	9	9	9	9	9	9
3	3	3	3	3	3	3	3	9	9	9	9	9	9	9	9
3	3	3	3	3	3	3	3	9	9	9	9	9	9	9	9
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1
1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1
1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 2: The Yin Yang Square.

1	1	1	1	2	2	2	2	2	2	10	10	10	10	10	10
1	1	1	1	2	2	2	2	2	2	10	10	10	10	10	10
1	1	1	1	2	2	2	2	2	2	10	10	10	10	10	10
1	1	1	1	2	2	2	2	2	2	10	10	10	10	10	10
1	1	4	4	4	4	2	2	2	2	10	10	10	10	10	10
1	1	4	4	4	4	2	2	2	2	10	10	10	10	10	10
1	1	4	4	4	4	8	8	4	4	4	4	4	4	4	4
1	1	4	4	4	4	8	8	4	4	4	4	4	4	4	4
1	1	1	1	8	8	8	8	4	4	4	4	4	4	4	4
1	1	1	1	8	8	8	8	4	4	4	4	4	4	4	4
1	1	1	1	2	2	10	10	10	10	8	8	8	8	8	8
1	1	1	1	2	2	10	10	10	10	8	8	8	8	8	8
8	8	8	8	2	2	10	10	10	10	8	8	8	8	8	8
8	8	8	8	2	2	10	10	10	10	8	8	8	8	10	10
8	8	2	2	2	2	10	10	4	4	4	4	4	4	10	10
8	8	2	2	2	2	10	10	4	4	4	4	10	10	10	10

0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0	1	1	0	1	1	1
0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1
0	0	1	1	1	1	0	0	0	0	1	0	0	0	1	1
0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	1
0	0	1	0	0	0	1	1	1	0	1	1	1	1	1	1
0	0	1	1	0	1	1	0	1	0	0	0	0	0	0	1
0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	1
0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	1
0	0	0	0	0	1	1	0	1	1	1	1	1	1	1	1
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
1	1	1	1	1	0	1	0	0	0	0	0	0	0	1	1
1	1	1	1	0	0	1	0	0	0	0	0	0	0	1	1
1	1	1	1	0	0	1	0	1	1	1	1	0	1	1	1
1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1

Figure 3: An Uneven Illuminated Image

Image	Image Size	SIPS - I_1	SIPS - I_2	SIPS - II
Idealized Patterns	16x16	545s		21s
Cheryl	320x320	3410s	2318s	966s
Starview	320x320	3413s	2340s	974s
Iceman	350x400	3408s	2342s	1353S

Table of Execution Time in CPU Seconds

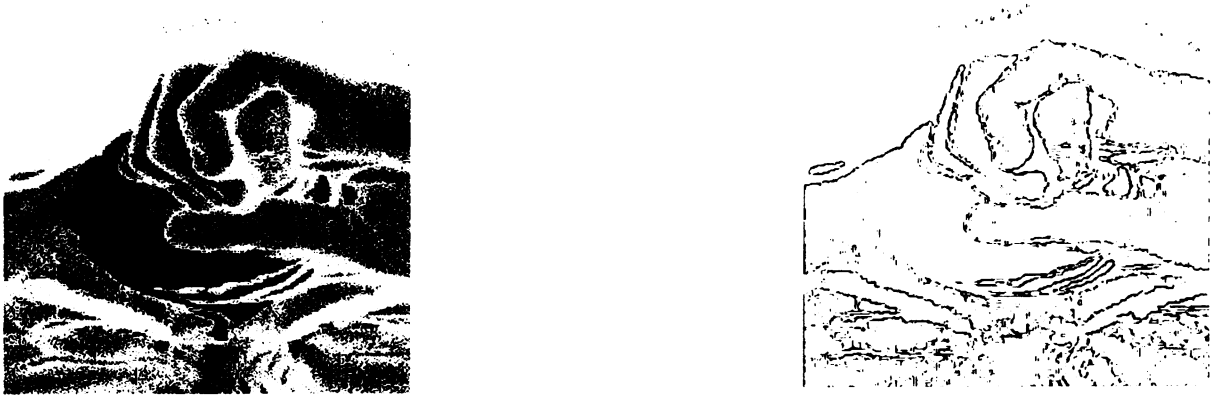


Figure 4: The Iceman Image

Figure 5: The Starview Image

Figure 6: The Cheryl Image

A Speech Recognition System Featuring Neural Network Processing of Global Lexical Features

Dr. Chen-Han Sung
William C. Jones, III
Center for Artificial Intelligence and Neural Networks
San Diego State University
6330 Alvarado Court, Suite 203, San Diego, California, U.S.A.

ABSTRACT

This paper presents an isolated-word speech recognition system which utilizes global lexical features to access words rapidly. Depending on whether the confusability of the lexicon is sufficiently low, the system can function as a complete recognizer, or as a hypothesization component of a larger speech recognition system. The system begins by extracting simple acoustic parameters, such as bandlimited energies, zerocrossing densities, and autocorrelations. These are combined in unique ways to produce seven phonetic features: voicing, frication, tongue height, tongue advancement, rounding, orality, and syllabicity. These phonetic features are summarized over the entire word using normalized polynomial moments, which results in a set of forty-eight global lexical features. The global lexical features participate as inputs to a neural network, which selects the correct word corresponding to the input. The entire system, including the neural network simulator, is coded in the "C" language and runs on a UNIX system.

INTRODUCTION

The goal of our isolated-word speech recognition system is to take any spoken English word from a given lexicon as an input, elicit a sufficiently small set of plausible candidate words and/or phrases to correctly hypothesize the given word (word hypothesization).

In one study [Marslen-Wilson, 1973], some hearers were able to recognize and repeat fluent speech with a delay of only 0.5 second. On average, this is enough time for only the first syllable-onset to reach the hearer. The number of words having a given onset (such as "tra") is about 30, on average. This kind evidence from close shadowing gives some indication of the degree of performance which should be demanded of a word hypothesizer.

The proposed isolated-word speech recognition system is a multi-level system with four stages of processing which are: Acoustic, Phonetic, Lexical, and Evidence Combination via a neural network architecture [Sung and Jones, '89]. The system can function either as a word hypothesizer or as a recognizer, depending on the size and confusability of the lexicon. The strategy is to first calculate global lexical features which characterize words as a whole, then submit it to a neural network architecture for evidence combination. It is anticipated that by combining information from a large number of lexical features, a small set of word candidates can be activated by elimination.

In the first stage of processing, acoustic features are extracted. These fall into three or four categories: autocorrelations, bandlimited energies and their ratios, and formant frequencies. Formant estimation uses our own algorithm, one which avoids the usual pitfalls.

In the second stage of processing, several phonetic features are calculated using acoustic data. These fall into several categories: voicing, frication, energy-based features, and formant-based features (vocoid articulations). A pitch detector is also built to be a stress detector.

In the third stage of processing, various methods for summarizing phonetic features over an entire word are used. These methods include moment computation, extremum evaluation, and arclength computation. Besides polynomial moments, some tests have been made using Fourier expansions.

In the last stage of processing, the resulting vector of lexical features is used as input to a neural network architecture, or as features of a classical pattern recognizer. We have found that the neural network performs about as well as the classical pattern recognizer in this context. Our performance results are typically in the 90th percentile.

A Neural Network Approach to Evidence Combination

Because a plurality of lexical features arise in this system, as in Miwa and Kido's [1986], there comes a point when one must consider how various pieces of evidence are to be combined to form a composite decision. As noted, Miwa and Mido take the classical path of the Bayesian classifier. Yet, this is inadequate to compensate for discrete variation, or non-unimodal variation (since a Gaussian assumption underlies their work), or even dependencies among features (because they employ a diagonal covariance matrix). However, there is some indication that a neural network may be capable of compensating for all these kinds of variation [Sung and Priebe, 1988].

In this paper, a special neural network architecture called multi-layer perceptron is employed. Such a neural net typically consists of a set of input nodes, each of which is connected to a set of hidden nodes by weighted connections; each hidden unit is in turn connected to all the output nodes by more weighted connection. The weights can be systematically learned using training data, via a learning rule known as the "Generalized Delta Rule" [Rumelhart and McClelland, 1986].

It is speculated that neural networks may have at least the same capacity for performance as classical pattern recognition systems, since both combine evidence multiplicatively using some type of confidence coefficients (connection weights in the former, variances in the latter). If a perceptron has at least two levels of connections, it has the potential of performing as well as the classical technique of principle components analysis, in which an optimal set of "rotated" (in the feature space) axes are calculated from the covariance matrix, so that the new feature axes are as independent as possible. In the multi-layer perceptron, computation of the hidden unit output signals is equivalent to matrix multiplication, which allows rotation and dilation of the feature space. The second layer of neurons combines the evidence present in the new, rotated vectors. Likewise, multi-layer perceptrons are probably less sensitive to peculiarities of the distributions of the lexical features (given a particular word), to the extent that even discrete variation can be compensated for. One can simply imagine a given output unit to be strongly connected to more than one hidden unit, which could reflect a multiplicity of corresponding acoustic-phonetic realizations. Thus neural networks might compensate for the inadequacies of classical pattern recognition techniques.

Recognition is done by performing a single forward pass, followed by selection of the output unit having the highest score. Hypothesization is done by selecting the set of output units having scores within a certain fraction of the maximum.

LEXICON

The following words were used for training and testing. This set of words is a mixture. The first set consisted of vocabulary that might be used to train a robot, and of words from Basic English. The second set consisted of disyllables containing voiced non-fricative intervocalic consonants (ostensibly for testing a segmenter). Finally, the ten digits were added. The list was scanned to ensure that each English phoneme occurs at least once.

meaning	universal	come	kennel	inning	go
a nook	Mimi	give	mumu	animal	take
Jimmy	woman	right	singing	singer	left
either	smoothing	close	wither	T. V.	open
movie	evil	true	giving	easy	false
oozing	razzle	top	isn't	fizzle	bottom

seizure	usual	I	Musial	vision	you
Edie	ado	forward	Aden	edition	backward
Iggy	giggle	yes	piggy	boogie	no
maybe	booboo	up	able	tibia	down
a book	beady	on	booty	addle	off
idiot	pudding	grip	tuner	humor	release
laser	butter	now	bigger	labor	then
button	botany	here	there	start	stop
above	below	in	out	function	relation
move	return	get	replace	translation	rotation
continue	interrupt	pencil	paper	all	only
almost	enough	do	make	ready	wait
where	when	much	little	together	separate
variable	constant	put	keep	lift	carry
see	look	hold	adjust	point	home
think	say	load	send	wrong	again
book	at	collision	hello	zero	one
two	three	four	five	six	seven
eight	nine				

System Performance

In an earlier test, five sets of the ten digits (by the same speaker) were digitized under uniform recording conditions (a quiet office). The first four set of data were presented four at a time, until all ten had been used; this entire process was repeated about four hundred times. Using the fifth word for each digit to be the test data, the system was able to recognize all ten words uniquely.

In the next test, forty fairly non-confusable words from the above lexicon were used to train, this time with each word being presented three times in succession (three identical instantiations). The entire list was presented about five hundred times. Even the training set was then used for testing, it is still interesting that the recognition score was one hundred percent. The system even managed to distinguish between potentially confusable pairs such as left / lift, and say / see. In both of these two tests, a smaller number of hidden neurons was used, namely thirty.

Finally, a training set was digitized which consisted of the 146 words in the above lexicon, and each word was digitized three times by the same speaker under the same recording conditions (quiet room). After digitization the data were processed to extract the moments. The number of hidden units was set at fifty. In training, each word was presented twice (using two of the three digitizations), and the list was presented in sequence (as opposed to random training). Each pass through the list constitutes one major iteration. Periodically, training was stopped and testing was performed. Testing was done by sequential presentation of the data in the third data set. For each word presented, several quantities were noted. First, the maximum score attained at any output neuron was noted. The number of words which scored at a level greater (i. e., whose corresponding ouput neurons attained higher activation) than a fixed fraction of this maximum (the inclusion threshold) were printed out, and they were counted. Finally, it was noted whether the right word was in this hypothesized list, and which word (output neuron) was the highest scorer.

Once these data had been prepared, two facts were sought. The first was the relative number of occasions when the correct word was found in the hypothesized list. Naturally, this varies as a function of the number of training iterations. The largest value obtained occurred when the number of training iterations was at its maximum (600), and the inclusion threshold was at its minimum (1%). It was equal to 132 words correctly hypothesized out of 146, or 90.6% , and the average number of words in the hypothesization set was only 5.47. On the other hand, further training does not seem to be useful, given the lexical features used and the state of the lexicon at present.

It is perhaps more useful to lower the inclusion threshold. In the beginning it was stated that the goal was to hypothesize a sufficiently small set of lexemes given a particular input waveform. From test results, it is clear that this goal has been fulfilled, with room for improvement. It will be interesting to see how low the inclusion threshold must be before hypothesization performance reaches 100%, and what the average number of hypothesized words is at this level.

The existing results are good, but fall short of our hopes. Even so, there are many examples where the system performed very well, considering the crudeness of our acoustic processing (8-bit) and the "leveling" of distinctions which lexical feature extraction can induce. The system distinguishes between "mumu" and "booboo," with each being uniquely hypothesized at the 10% inclusion level. After 600 iterations at the 50% inclusion level, a total of 85 words, or 58.2%, were distinguished correctly and uniquely, despite the potential confusability of the lexicon.

There is one other positive development which was not anticipated. In those cases in which the correct word does not appear among the hypothesized words, the maximum score attained by any word is almost invariably rather low, certainly much less than 0.5. This could provide a way of systematically lowering the inclusion threshold so that the correct word has a greater likelihood of being present among the hypothesized words.

Finally, to assess the value of the neural network approach in relation to traditional pattern recognition techniques, a simple classical pattern recognition system was constructed. The system calculates class-specific means and variances for the same data used to train the neural network. These data are then used to classify samples by placing them in the class whose mean is closest, relative to the normalized Mahalanobis distance for the particular class. Since inclusion thresholds can be set to induce any desired degree of hypothesizer performance, but the number of hypotheses submitted soon becomes so large as to render the hypothesizer useless, any salient measure of hypothesizer performance must address both these issues. Thus, our results shows how the number of hypotheses increases as a function of the number of words correctly hypothesized. From these results, and those inferred earlier, it can be seen the the neural network's performance is comparable to that of the classical pattern recognizer.

In summary, we have presented a system which can be used as a word recognizer for a small, non-confusable lexicon, or as a hypothesizer for larger lexicons. The system begins by extracting parameters on each time frame, proceeding from the concrete level (acoustic) to the abstract (phonetic). These parameters are combined systematically to produce global lexical features, which characterize an entire word. These are used as inputs to a neural network, which attempts to find the best match(es) for the given input. Overall, there have been many encouraging results.

Reference

Marslen-Wilson, W. D., "Linguistic Structuring and Speech Shadowing at Very Short Latencies", Nature, 244: 522-523, 1973.

Miwa, J., and K. Kido, "Speaker-Independent Word Recognition for Large Vocabulary Using Pre-Selection and Non-linear Spectral Matching," Proc. 1983 IEEE ICASS, 2695-2698, 1986.

Rumelhart, D. E., J. L. McClelland, and the PDP Research Group. Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Cambridge: MIT Press, 1986.

Sung, C. H., and C. E. Priebe, "Temporal Pattern Recognition," IEEE 1988 International Conference on Neural Networks, I: 689-696.

Sung, C.-H., and W. C. Jones, "A Robust Speech Recognition System Featuring Neural Network Processing of Global Lexical Features", Technical Report, CAINN, San Diego, August 1989.

AN ANALOG-DIVIDER-DESIGN BASED ON A PERCEPTRON-NEURAL-NETWORK

Axel Thomsen, Martin A. Brooke
Georgia Institute of Technology
School of Electrical Engineering
Atlanta, Ga. 30332-0250

ABSTRACT

An analog divider circuit design is presented. It is a perceptron neural-network consisting of neurons designed to approximate the desired output in a certain region of input values. Neuron and network design are described and results of circuit simulations are presented.

INTRODUCTION

As one of the fundamental mathematical operations division is important in signal processing applications. The idea to build a divider circuit based on neural network techniques originated in the comparison between the abilities of single-layer perceptron networks and the requirements for a divider:
A single-layer feed-forward perceptron network is able to decide, on which side of a hyperplane in the hyperspace of the input vectors a set of input values are located, or in other words, if the weighted sum of input values exceeds a certain threshold. For the special case of two input variables a and b and a threshold of zero the weighted sum equation is

$$Wa * a + Wb * b = 0 \qquad \text{or} \qquad a / b = - Wb / Wa$$

and thus a certain ratio is defined by the weights Wa and Wb.
Figure 1 shows a graphical illustration of the plane of input values and the lines of constant ratio defined by the weights in the divider design proposed here.
The divider works in the following way: Input and output values are represented by currents, here simply called a, b and (a/b). Neurons calculate whether the ratio a/b is larger or smaller than the neuron's built-in ratio. Each single neuron corresponds to a line of constant ratio (Wb/Wa) in the input plane. For input ratios far larger than this ratio the neuron provides a certain constant output current, for input ratios far smaller it has zero output current. In the intermediate region around this ratio it provides an output current that is approximately linearly changing. By placing several lines in the plane of input values, bounded by $amin<a<amax$ and $0<bmin<b<bmax$,thus using several neurons with different ratios, and adding up their output currents, the ratio can be approximated for all possible input values. Figure 4 illustrates the principle of operation, showing a sweep through the input plane and the corresponding outputs of the single neurons and the whole circuit.

THE DESIGN OF A SINGLE NEURON

The basic building-block for a perceptron network is the neuron. The neuron designed here (fig. 2) has the following characteristics:

- the output is a current since the outputs of the neurons have to be added to obtain the final output value.
- the input is a current, too, to avoid the need for external voltage-to-current conversions.
- the gain of the neuron is changing with the input current, high gain for low currents, where the lines of constant ratio are closely spaced, lower gain for higher input currents. This leads to good interpolation characteristics.
- the weights Wa and Wb, are realized by the geometrical parameters (W/L)n and (W/L)p. The signs of the weights are fixed by the design.

The neuron consists of a current comparator with voltage output, which has a gain that is inversely proportional to the current. The output voltage of the comparator is used to control the output current by a transistor switch in the following way:
For high output voltages the switching transistor is in the ohmic region and the current is limited by the reference source. For low output voltage the switching transistor is turned off and thus the output current is zero. In the intermediate region the reference source is in the ohmic region, the switching transistor is in saturation and the output current changes with the output voltage of the comparator. The reference is chosen such that the reference source reaches saturation for the comparator reaching Vdd-Vt. The 'switch off' occurs at Vss+Vt, where Vdd and Vss are the power supply voltages and Vt is the threshold voltage of the MOSFETs. The input currents are chosen such that the comparator has a linear region between Vdd-Vt and Vss+Vt. This results in a relatively simple nonlinearity. A drawback is, that the gain of the complete circuit is difficult to design analytically. Here a circuit simulator was used to adjust parameters.

DESIGN OF THE WHOLE NETWORK

Since a high regularity reduces the efforts in the design, especially when the design has to be done in a "try-and-error" manner, the input plane was divided into similar segments so that the neurons differ only in geometric parameters representing different ratios. These can be determined easily from the corresponding ratio -Wb/Wa by

$$-Wb/Wa = (W/L)p \;/\; (W/L)n$$

where the subscripts p and n stand for PMOS and NMOS transistors respectively. The gain, which corresponds to the width of the region of interpolation in the input plane, is the same for all neurons. Only in the regions where interpolation is more difficult, near the maximum and near the minimum ratio, the lines were spaced twice as close, corresponding to a higher gain of the

neurons in this region.
The resulting (W/L)p / (W/L)n ratios for the comparator and the W/L ratio for the output devices are shown in figure 3. To obtain a similar channel-length modulation for all components of the comparators the device length L has to be the same for all devices.
The W/L ratio of the output devices for each neuron is proprotional to the difference between the input current ratios at the 'switch off' of the next neuron and the 'switch off' of the considered neuron. The fourth neuron for example covers the range from 0.58 to 1.02 and has a current output proprotional to 1.02 - 0.58 = 0.44.

SIMULATION-RESULTS

The circuit was simulated using the circuit simulator PSPICE.
Figure 1 shows the location of the DC-sweep in the input plane.
The principle of operation of the circuit is illustrated by figure 4. It shows, how the different neurons do the interpolation in their range and give constant output outside this region. Figure 5 shows the absolute error (normalized to the maximum output) and the relative error of the output current. The maximum relative error is approximately 7% disregarding the region near a/b=0.
The maximum absolute error is 3.5% of the maximum output, the error near a/b=0 is about 1% of the maximum output.
Other simulations of the single neuron were performed to analyze the influence of different error sources on the output current.
The most critical parameters are the geometrical values and relative variations in the threshold voltage. The influence of finite output resistance of the reference sources is significant, too, but it could be reduced by using cascoded current sources instead. Variations of the channel-length modulation did not have significant influence on the accuracy of the output.

CONCLUSIONS

The perceptron network seems to be a promising design for an analog divider. The abilities of a single layer perceptron network without feedback and the requirements for a divider are very similar. A simple neuron circuit with good interpolation characteristics can be used. The result is a simple structure consisting of only 30 transistors leading to a relative error of about 7%. The error could be further reduced by increasing the number of neurons. The maximum accuracy will depend on the accuracy of geometrical parameters in the process.

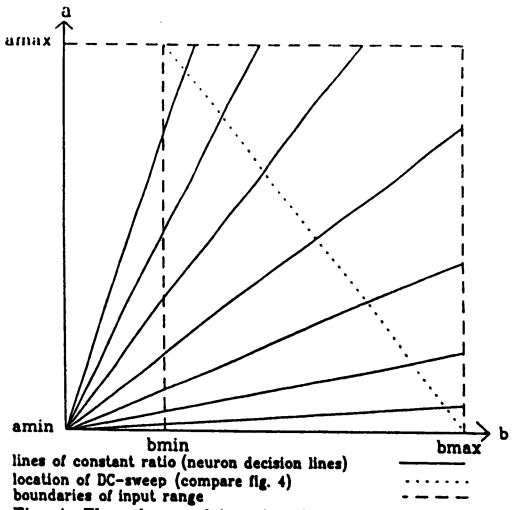


Fig. 1: The plane of input values

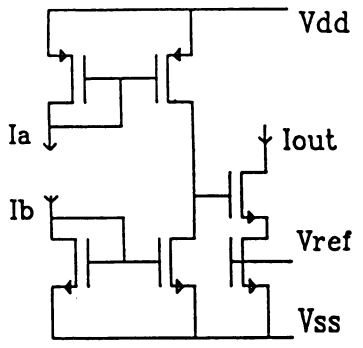


Fig. 2: The single neuron

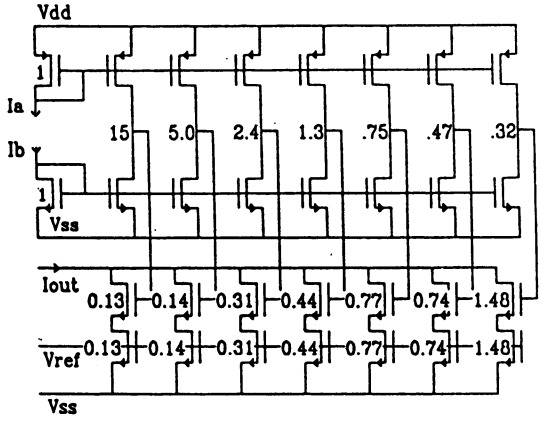


Fig. 3: Circuit diagram of the complete divider.
Numbers indicate the (W/L)p / (W/L)n ratio and the W/L of the output devices

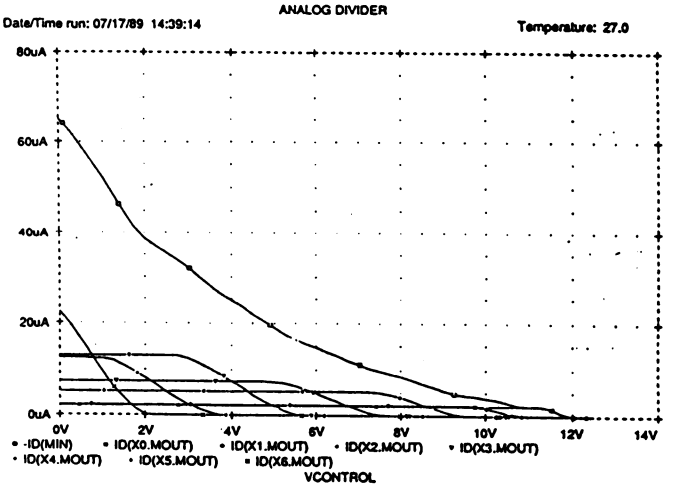


Fig. 4: output current of the single neurons and the whole circuit (VCONTROL is used to control input currents:
Ia = 10uA - 0.8uA * VCONTROL
Ib = 2.5uA + 0.6uA * VCONTROL)

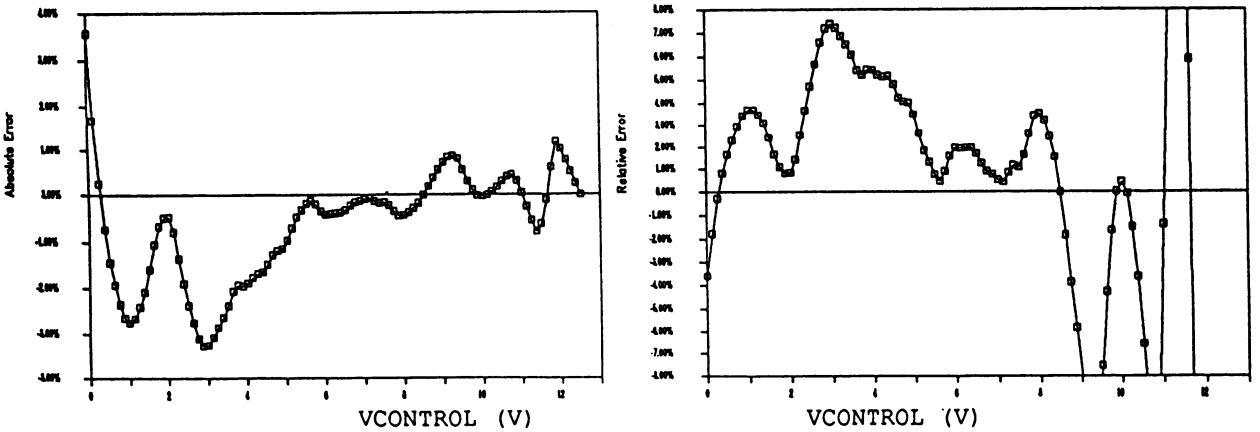


Fig. 5: (a) absolute error of the output current
(b) relative error of the output current
(normalized to the maximum output)

Experiments with the Spatio-Temporal Pattern Recognition Approach and the Dynamic Time Warping Approach to Word Recognition

M. Daniel Tom
M. Fernando Tenorio

School of Electrical Engineering
Purdue University
West Lafayette, IN 47907

(mdtom@ei.ecn.purdue.edu)
(tenorio@ee.ecn.purdue.edu)

July, 1989

Abstract

This paper compares the performance of a Spatio-Temporal Pattern Recognition neural network word recognizer with a traditional Dynamic Time Warping (DTW) word recognizer. Each recognizer is evaluated over a set of 4 words spoken by 6 different speakers/talkers (24 tokens total.) On each run, one speaker's utterances are used as training input for the neural network, or as templates for the DTW word matcher. The utterances from the remaining 5 speakers are reserved for testing. Using a different speaker's data as the training/template set each time, a total of six runs are performed. Averaging the results over 6 runs yields a 97.5% accuracy rate for the neural network, as compared with an 87.5% rate for DTW with a slope constraint of 0, or 85% for DTW with a slope constraint of 1.

Introduction

The Spatio-Temporal Pattern Recognition Approach to word recognition has been introduced in IJCNN-89 [6]. We have built a limited vocabulary, speaker independent, isolated word recognition system based on the above approach. The system has been trained with 4 words spoken by one single speaker (talker.) The trained system has recognized the 4 words in the vocabulary spoken by each of 5 other speakers.

Here we extend our experiments with the system by varying the training set. To test for speaker independence, we use only one speaker's utterances as training input. The trained system is tested on other speakers' utterances. We vary the training input, using one speaker's data each time. The results are presented. Training and testing times are recorded.

To compare the Spatio-Temporal Pattern Recognition Approach to a traditional technique, we implement an isolated word recognizer based on the Dynamic Time Warping (DTW) approach [1]. We evaluate the DTW word recognizer by using a single speaker's utterances as templates, and matching the utterances voiced by other speakers. The results based on different speaker's template utterances are presented. Computation time is recorded.

The Spatio-Temporal Pattern Recognition Approach

The Spatio-Temporal Pattern Recognition Approach is a static pattern recognition approach applied to recognize a spatio-temporal pattern of time varying speech features. A system has been built with this approach and recognizes digitized isolated words without performing segmentation, phoneme identification, or time alignment. The preprocessing only includes endpoints identification, preceding and trailing silence removal, and word length determination. A

fourth-order Linear Predictive Coding (LPC) analysis [2] is performed on each of 32 equally spaced frames. The 4 LPC coefficients plus 4 other features from each frame are input to a multilayer feedforward neural network.

Preprocessing is performed before LPC analysis. The endpoints of the word are determined [3]. Preceding and trailing silence are stripped off. Dividing the length of the word by 32 gives the spacing between the starting points of the 256-point LPC analysis frames. The LPC analysis frames may overlapped if the length of the preprocessed utterance is less than 32x256 points (819.2 ms at 10 kHz sampling rate.)

For each frame, we obtain 8 representative features that are input to the neural network recognizer. We subtract the mean value of the digitized signal, and apply Hamming window weighting. We then apply a 4-th order LPC analysis, and obtain 4 LPC coefficients plus the normalized prediction error. In addition, we record the peak amplitude, the number of zero crossings, and the root-mean-squared energy of the signal in the frame. We apply a nonlinear normalization exp(-x) to keep the last three feature values within the [0,1] interval. We thus obtain 8 features per analysis frame.

The neural network pattern recognizer is a multilayer feedforward network trained with the backpropagation learning algorithm [4]. The neural network has 4 output layer units, where each represents a word in the lexicon. We use 50 hidden layer units. The network takes 256 inputs, which are the 32 frames with 8 features each. We use a learning rate of 0.1, and a momentum constant of 0.9. We do not clear the momentum term during training. Initial weights are small random values uniformly distributed between -0.3 and 0.3. We represent the desired output with 1.0 at the unit representing the corresponding word, and 0.0 at other units. The desired values are not achieved with linear graded units which use the sigmoidal nonlinearity. Thus we choose the word corresponding to output layer unit having the highest output value as the recognized word.

Evaluating the Spatio-Temporal Pattern Recognition Approach

In this section we present the results with our extended experiments with the neural network recognizer. We have presented the result of training the network with only one speaker's data, and perfect recognition of 5 other speaker's utterances [6]. Here we vary the training set, always using only one speaker's data set as training input. We test the trained network on the remaining data sets. We also record the computation time to train the network with one data set, and the time to recognize the remaining data sets.

The data sets consist of utterances of 4 different words (colt, star, drink, and taste) spoken by 6 female native speaker of English. The speakers are identified by their initials: C.L.C., J.J.H., L.L.S., S.E.K., J.L.O., and L.H.J. The utterances are recorded in a quiet room and are digitized at 10 kHz. The A/D conversion has 12-bit resolution. These utterances vary in length, and have a variable duration of preceding and trailing silence.

We first train the network with 100 epochs on the 4-word set spoken by C.L.C., and then test the network on the same 4 words spoken by 5 other speakers. The perfect recognition in the above experiment has been presented in IJCNN-89. We repeat the above experiment 5 times, training with the utterances of J.J.H., L.L.S., S.E.K., J.L.O., and L.H.J. respectively. Each training run starts with initial weights, and testing is done on the remaining 5 data sets.

The results are summarized in the Table I. The neural network only fails on 3 out of the total 120 testing words. This gives an average of 97.5% accuracy. The neural network fails to recognize the words "colt" and "taste", both spoken by L.H.J., when trained with the words spoken by J.J.H. The third failure occurs on the input word "taste" spoken by L.H.J. when trained with the 4 words spoken by S.E.K.

We perform our experiments on the Gould NP-1 computer with vector processor/arithmetic acceleractor. Training 100 passes of the 4-word training set take 11.1 cpu seconds (rounded to the nearest tenth of a second) on the sequential version, and only 1.0 second on the vectorized version. Testing all 24 words in the training set and testing sets take 0.3 cpu second on the sequential version, and 0.1 second on the vectorized version.

The Dynamic Time Warping Approach

The traditional dynamic programming approach serves as a comparison for the Spatio-Temporal Pattern Recognition Approach. We build a DTW word recognizer based on the minimum prediction residual principle (Itakura, 1975.) The training and testing utterances are the ones mentioned in the previous section. The evaluation procedure is also similar. The DTW word recognizer is run with two different slope constraints on the warp path [5]. Recognition accuracy and computation time are recorded.

Using the same utterances as we use for the neural network, we can use the same endpoints determined for each utterance. Preceding and trailing silence are stripped off. Here we apply 14-th order LPC analysis on successive 128-point frames. The 14 LPC coefficients give a better characterization of the frame than the 4 coefficients from an LPC-4 analysis described in the previous section. Each utterance is processed into a variable number of frames (due to varying word length) with 14 LPC coefficients per frame.

Testing of the DTW word recognizer is similar to that of the neural network. Speaker independence is stressed. Only one speaker's utterances are used as templates for the DTW word recognizer. The utterances voiced by the remaining 5 speakers are reserved for matching. Six runs are performed, with each run using a different speaker's data as the template set. Besides using Itakura's slope constraint of 0 [1], the slope constraint of 1 introduced by Sakoe & Chiba [5] is applied in a second run. Sakoe & Chiba reports higher accuracy rate with the slope constraint of 1.

The results are summarized in the last two columns of Table I. The DTW word recognizer with slope constraint 0 fails on 15 out of the total 120 input utterances. This gives an average of 87.5% accuracy. The DTW word recognizer with slope constraint 1 fails on 18 out of the total 120 input utterances, which translates to 85% accuracy.

As the DTW word recognizer needs no training, we only report the testing time. Taking the 4 template words for reference and performing matching on a single input word take 2.9 seconds and 15.9 seconds with a slope constraint of zero and one respectively. Testing all 24 words in the training set and testing sets would take 69.6/381.6 seconds with a slope constraint of zero/one.

Conclusion

Evaluating over the 6-speaker, 4-word vocabulary, the Spatio-Temporal Pattern Recognition neural network isolated word recognizer has a significantly higher accuracy rate than that of the Dynamic Time Warping word recognizer. The vectorized version of the neural network recognizer is about 2 orders of magnitude faster than the DTW word recognizer. The output layer unit activations of the neural network also shows more discriminatory power than the prediction residuals computed by the DTW approach, as can be contrasted in Figures 1, 2, and 3.

References

[1] F. Itakura, "Minimum Prediction Residual Principle Applied to Speech Recognition," IEEE Transaction on Acoustic, Speech, and Signal Processing, Vol. 23, No. 1, February 1975.

[2] J. D. Markel, and A. H. Gray Jr, "Linear Prediction of Speech," Springer-Verlag, 1976.

[3] L. R. Rabiner and M. R. Sambur, "An algorithm for determining the endpoints of isolated utterances," Bell System Technical Journal, vol. 54, no. 2, pp. 297-315, 1975.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," in D. E. Rumelhart and J. L. McCelland (Eds.), "Parallel Distributed Processing: Explorations in the Microstructure of Cognition Vol. 1," MIT Press, Cambridge, 1986.

[5] H. Sakoe and S. Chiba, "Dynamic Programming Algorithm Optimization for Spoken Word Recognition," IEEE Transaction on Acoustic, Speech, and Signal Processing, Vol. 26, No. 1, February 1978.

[6] M. D. Tom and M. F. Tenorio, "A Spatio-Temporal Pattern Recognition Approach to Word Recognition," Proceedings of the International Joint Conference on Neural Networks, Vol. 1, pp. 351-355, Washington, D.C., 1989.

Table I
Summary of recognition results with the Spatio-Temporal Pattern Recognition neural network (NN) and the Dynamic Time Warping word recognizer with slope constarints zero (DTW-0) and one (DTW-1)

Training Speaker	Recognized / New utterances		
	NN	DTW-0	DTW-1
C.L.C.	20/20	18/20	18/20
J.J.H.	18/20	16/20	14/20
L.L.S.	20/20	16/20	16/20
S.E.K.	19/20	18/20	19/20
J.L.O.	20/20	18/20	15/20
L.H.J	20/20	19/20	20/20
Total	117/120	105/120	102/120
Average	97.5%	87.5%	85.0%

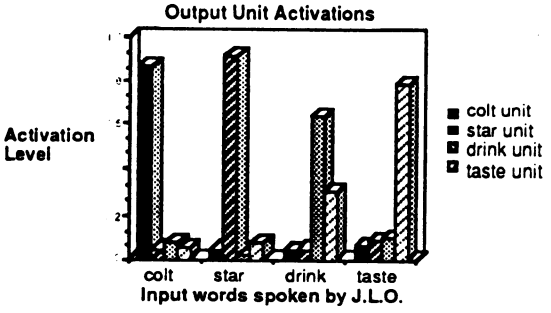


Fig. 1. Output unit responses to different input words spoken by J.L.O. when trained with words spoken by C.L.C.

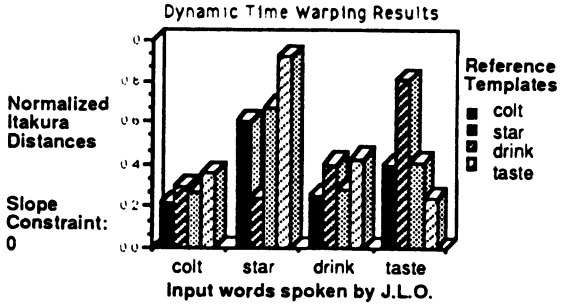


Fig. 2. Normalized Itakura distances on warping different input words spoken by J.L.O. with reference words by C.L.C. The word "drink" is incorrectly recognized as "colt." Slope constraint 0 is used.

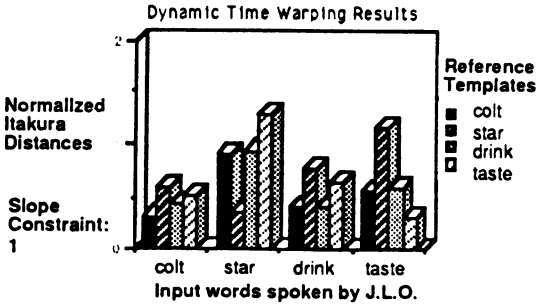


Fig. 3. Normalized Itakura distances on warping different input words spoken by J.L.O. with reference words by C.L.C. The word "drink" is incorrectly recognized as "colt." Slope constraint 1 is used.

Point Pattern Matching using a Hopfield-type Neural Network

Darrin R. Uecker* and Hiroshi Sakou
Central Research Laboratory, Hitachi Ltd., Kokubunji, Tokyo 185, Japan

Abstract: In this paper we present a method for point pattern matching[1] using a Hopfield-type neural network. We give a formulation with energy functions which converge to the optimal solution in a least squares sense. The input points are assumed transformed from the template points by affine transformation with additive positional noise. We show that the number of iterations required for the network to converge is linear with the number of points in the template. Results are given with example matchings and future work and extensions are discussed.

Introduction

A network for solving difficult optimization problems using neural like elements was developed by Hopfield and Tank [2,3]. They were able to solve the Traveling Salesman Problem (TSP) in certain cases and showed that their network had the potential to solve many types of optimization problems. Currently much research is being done in using the Hopfield network to solve other optimization problems by proper choice of the energy function. Typical examples of this are graph matching [6] and the Nonthreatening Queens Problem [7].

We have used a Hopfield-type neural network to solve the point pattern matching problem, which is especially important in model-based computer vision. In our system points may undergo affine transformation, i.e., translation, rotation, scaling and/or stretching, and some random additive positional noise. The Hopfield network's parallel computation offers the speed that has been a major drawback of many previous point pattern matching methods and in general our method compares favorably with other techniques[8,9] which use heuristics in error estimation.

Hopfield Neural Network Model

Hopfield introduced his model in 1982 [4] as a highly interconnected network based on neurophysiological evidence of the interaction between neurons. Neurons are modeled as amplifiers and each output is connected to all other neurons and receives input from all other neurons. The input/output characteristics of the neurons are modeled as a sigmoid function. The dynamic equations for this network are given as,

$$\frac{du_{ij}}{dt} = -\frac{u_{ij}}{\tau} - \frac{\partial E}{\partial V_{ij}} \tag{1}$$

where,

$$V_{ij} = \frac{1}{1 + e^{-u_{ij}}} \tag{2}$$

These equations are based on classical neurodynamics where u_{ij} represents the effective input potentials, V_{ij} represents the output or firing frequencies of the neurons, and τ is the time constant associated with the input capacitance of the cell membranes and the transmembrane resistance.

In equations 1 and 2, E is known as the computational energy of the network. With initial u_{ij}'s selected the network will evolve in time, through the state space given by the V_{ij}'s, and eventually converge to some minimum energy location. By properly formulating the computational energy, Hopfield and Tank were able to decode the solution to the TSP from the final states of the neurons.

Point Pattern Matching Formulation

For a typical point pattern matching problem a set of input points need to be optimally matched to a set of model template points by minimizing a cost function. The template coordinates, which are characteristic points to be detected in the input object image, will be represented as two vectors $x = [x_1 \ldots x_n]^T$ and $y = [y_1 \ldots y_n]^T$ and the input vectors as $X = [X_1 \ldots X_n]^T$ and $Y = [Y_1 \ldots Y_n]^T$. The input image is assumed to be transformed by an affine transform with additive positional noise, i.e.,

$$X = ax + by + c + e, \quad Y = Ax + By + C + E \tag{3}$$

where e and E are additive positional noise vectors and a, b, c and A, B, C are all scalar values.

To formulate the point pattern matching problem in terms of a Hopfield network we use a representation which will allow us to decode the output of the final states of the neurons into the appropriate matched pairs. Therefore, each template coordinate has n neurons where one of the neurons will be on indicating the number of the input pair that it should be matched with.

---

* Visiting Researcher until 12/12/89 from University of California at Santa Barbara, Santa Barabara, CA 93106.

Then, for n points there will be an n-square matrix we will denote as V. Where an on-neuron V_{ij} denotes a matching of (x_i, y_i) with (X_j, Y_j) and an off-neuron denotes a no-match between the points. Figure 1 shows a physical representation of this. The formulation is a very natural one because the final matrix can only have one neuron on in each row and column and therefore is a permutation matrix which maps X to x and Y to y.

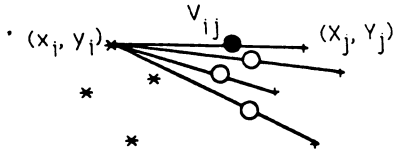


Figure 1. Template-input relationship.

We use two different energies in this formulation. The first energy, E_1, is a constraint which is used to insure that there is only one neuron with a one value per row and column. It is given as

$$E_1 = \frac{K_1}{2}\left[\sum_i\sum_j\sum_{k\neq j} V_{ij}V_{ik} + \sum_j\sum_i\sum_{k\neq i} V_{ij}V_{kj}\right] + \frac{K_2}{2}\left[\sum_i\sum_j \frac{V_{ij}}{n} - 1.5\right]^2 \tag{4}$$

This equation insures that the V matrix converges to a permutation matrix while staying away from the origin.

The second energy, or cost function, insures that the matched solution has a minimum error, or cost, in a least squares sense. To get the optimal solution for a known template-input coordinate pair the well known least squares method can be used. In our case we reformulate it as

$$\begin{bmatrix} \langle x,x\rangle & \langle x,y\rangle & \langle x,z\rangle & 0 & 0 & 0 \\ \langle y,x\rangle & \langle y,y\rangle & \langle y,z\rangle & 0 & 0 & 0 \\ \langle x,z\rangle & \langle y,z\rangle & n & 0 & 0 & 0 \\ 0 & 0 & 0 & \langle x,x\rangle & \langle x,y\rangle & \langle x,z\rangle \\ 0 & 0 & 0 & \langle y,x\rangle & \langle y,y\rangle & \langle y,z\rangle \\ 0 & 0 & 0 & \langle x,z\rangle & \langle y,z\rangle & n \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ A \\ B \\ C \end{bmatrix} = \begin{bmatrix} \langle VX,x\rangle \\ \langle VX,y\rangle \\ \langle VX,z\rangle \\ \langle VY,x\rangle \\ \langle VY,y\rangle \\ \langle VY,z\rangle \end{bmatrix} \tag{5}$$

where $\langle .,. \rangle$ represents the inner product in R^n and z is a column vector of size n filled with ones. When V converges to the correct permutation matrix we can calculate the optimal coefficients. Using this formulation, we define E_2 as

$$E_2 = \frac{K_3}{2}\sum_i\sum_j V_{ij}^2\left\{[X_j - (ax_i+by_i+c)]^2 + [Y_j - (Ax_i+By_i+C)]^2\right\} \tag{6}$$

Thus, at every iteration of the dynamics, the coefficients are recalculated using equation 5. When the system converges we have the permutation matrix that matches the appropriate pairs and naturally the optimal coefficients of the affine transform.

Hopfield has shown[5] that if the synaptic interconnections are symmetric and there is no cross talk within neurons then the energy function will always be decreasing as a function of time. In our formulation, however, these conditions are not met. We therefore modify equation 1 by eliminating the u_{ij}/τ term which makes it is easy to see that the time derivative of the energy will be less than zero regardless of the energy function chosen. Uesaka[10] has shown that eliminating this term does not greatly affect the energy curve associated with this network. Incidentally, our system can get good results with the original equations but with more iterations. Thus, the modified equation is used because it is not clear that with the original equation the energy is always decreasing.

The energy differentials required by the dynamics which are derived from equations 5 and 6 are thus,

$$\frac{\partial E_1}{\partial V_{ij}} = K_1\left(\sum_{k\neq j} V_{ik} + \sum_{k\neq i} V_{kj}\right) + \frac{K_2}{n}\left(\sum_i\sum_j \frac{V_{ij}}{n} - 1.5\right) \tag{7}$$

$$\frac{\partial E_2}{\partial V_{ij}} = K_3 V_{ij}\left\{[X_j - (ax_i+by_i+c)]^2 + [Y_j - (Ax_i+By_i+C)]^2\right\} + K_3\sum_m\sum_n V_{mn}^2\left\{\begin{array}{l}[X_n - (ax_m+by_m+c)]\left[\left\{\frac{\partial a}{\partial V_{ij}}x_m + \frac{\partial b}{\partial V_{ij}}y_m + \frac{\partial c}{\partial V_{ij}}\right\}\right] \\ +[Y_n - (Ax_m+By_m+C)]\left[\left\{\frac{\partial A}{\partial V_{ij}}x_m + \frac{\partial B}{\partial V_{ij}}y_m + \frac{\partial C}{\partial V_{ij}}\right\}\right]\end{array}\right\} \tag{8}$$

The second part of equation 8 is much smaller in magnitude than the first part and can therefore be neglected during simulations. To solve these equations we used the Euler-Cauchy method.

In the point pattern matching case the parameters represent a trade-off between a possible solution, K_1 and K_2 in E_1, and one that is weighted heavily in favor of minimum squared error, or K_3 in E_2. Clearly every permutation matrix of V is a local minimum in the energy space involved. A good set of parameters which prevent our system from entering the local minimum are

$$K_1 = 5.0 \quad K_2 = 10n \quad K_3 = 1.0 \quad \text{for } \Delta t = 0.1.$$

Although a different set of parameters might work for different cases these parameters worked for all cases tested.

One very important initial parameter is the initial neural input potentials, or u_{ij}'s. There is no *a prior* knowledge of the matched pairs used thus, all the u_{ij} must be set to zero which means we are not favoring any matching. With this initial set-up the only initial change in V_{ij} will come from a change in E_2. This is due to the fact that the initial neural input potentials are equal and E_1 is only dependent on V_{ij}. To break the symmetry the following a, b, c and A, B, C are used for the first iteration,

$$a = \cos\theta,\ b = \sin\theta,\ c = -a\bar{x} - b\bar{y} + \bar{X},\ A = -\sin\theta,\ B = \cos\theta,\ C = -A\bar{x} - B\bar{y} + \bar{Y},$$

where theta is the rotation angle which is obtained by the estimation of the distribution of points between the input and the template points, and $\bar{x}$ is the average value of the x_i's. This is equivalent to biasing the initial neural input potentials because these initial coefficients transfer this information directly to the u_{ij}'s and thus the V matrix.

Our results show that if a good estimate of the rotation angle, θ, between the template and input is made then the system will always converge adjusting for any error in the estimate as well as shift, stretching, shrinking and additive noise. Therefore, we can by checking at different angles, find the optimal matching. This will be discussed in more detail in the next section.

There is also preprocessing which is done to normalize the coordinate values and center the images on an axis. The normalization is simply

$$x_i = \frac{x_i}{\max\{(x_{max} - x_{min}), (y_{max} - y_{min})\}},\ y_i = \frac{y_i}{\max\{(x_{max} - x_{min}), (y_{max} - y_{min})\}} \tag{9}$$

The same is also done for the input image. This normalizes the individual images so that the K_3 parameter can remain constant independent of image size or field size.

Experiments

Figure 2 shows an optimal matching with characteristic points of a star and a transformation of that star matched using this system. Figure 3 shows a typical energy curve which is generated during matching. As expected the energy curve clearly decreases with the iteration number, slowly at the beginning as the system tries to figure out the best match and then rapidly towards convergence at the end.

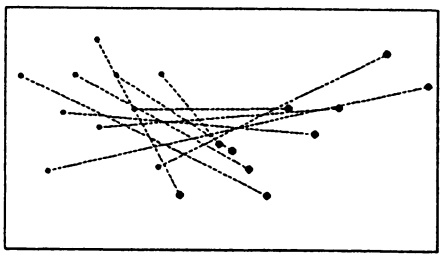


Figure 2. Star matching.

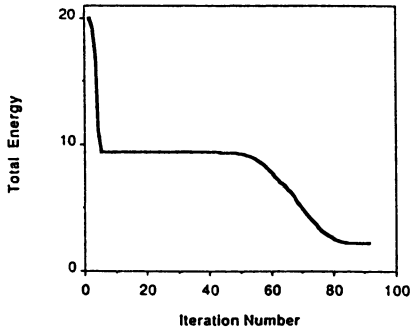


Figure 3. Typical energy curve.

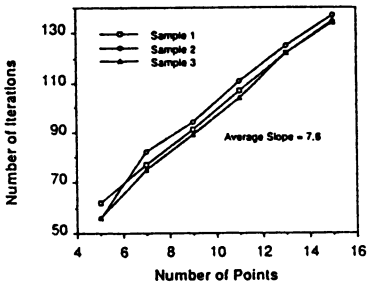


Figure 4. Linearity of iteration number.

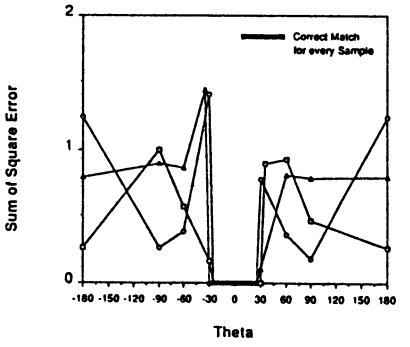


Figure 5. Theta estimate error.

Figure 4 shows a plot of the relationship between the number of points and the iteration number for three different samples of randomly generated inputs. The samples are a string of random coordinates where we used 5-15 points in a 250 by 250 region for testing. The figure shows a definite linearity between the number of points and the iterations required to solve the problem. The increase in iteration with the number of points is

due to the increase in density of points in the image plane. In other words, because the difference in energy from E_2 is smaller between points, it takes longer for E_1 to be able to take over and make the system converge.

As mentioned in the previous section, if the angle of rotation can be estimated correctly, our system can insure the optimal solution. With this is mind we tested a number of randomly generated petterns to see the amount of error that could be tolerated in the estimate and still get the correct solution. Figure 5 shows some of these results in which we can see that the system is able to overcome the difference, within +/- 25 degrees, and still find the optimal solution. It is important to note that at all other angles there is a larger error because this suggests that we can attempt matchings at different angle intervals, in other words, at several initial u_{ij}, and choose the smallest error as the correct match. This will add computation to the overall matching but it will not affect the linearity of the iteration versus number of points. Figure 6 shows another matching result.

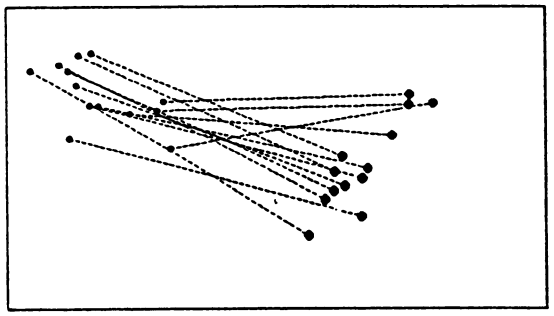


Figure 6. Another matching result.

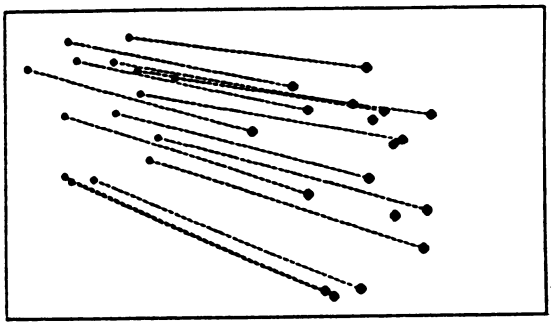


Figure 7. Additional points matching.

Conclusion and Discussion

In this paper we proposed a method for point pattern matching using a Hopfield type neural network. The method uses a formulation in which the optimal matching is obtained in a least squares sense and the matching is decoded directly from the outputs of the neurons after the system converges. Our system also demonstrates an ability to converge to the optimal solution linearly with the number of points in the pattern regardless of deformations, where the deformations are brought on by affine transform and additive positional noise. Finally, because this network is general in the sense of least squares it is possible that it could be applied to many other correspondence problems.

These results have been all on patterns with equal number of template-input points. The current system has the ability to solve a problem with additional input points in certain situations, as shown in figure 7, but not in all cases.The applications of this system would be much greater if we could insure the optimal solution in this case. Our current work is aimed at solving this problem.

Acknowledgements

The authors would like to thank Mr. Kazuaki Iwamura for his many helpful discussions and Mr. Hitoshi Matsushima and Dr. Yoshito Tsunoda for their encouragement and support throughout this research. They are all members of Central Research Laboratory, Hitachi Ltd.

References

[1] H. Sakou and D.R. Uecker. "Point Pattern Matching using a Hopfield-type Neural Network," Proc. of the 39th Nat'l Con. of Information Processing Society of Japan, to be published Oct. 1989. (in Japanese).

[2] J.J. Hopfield and D.W. Tank. "'Neural' Computation of Decisions in Optimization Problems," Biological Cybernetics, 52, 141-152, (1985).

[3] J.J. Hopfield and D.W. Tank. "Computing With Neural Circuits: A Model," Science, Vol 233, 625-633, (1986).

[4] J.J. Hopfield. "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," Proc. Natl. Acad. Sci. USA, Vol. 79, pp. 2554 - 2558, (1982).

[5] J.J. Hopfield. "Neurons with Graded Response have Collective Computational Properties Like Those of Two-state Neurons," Proc. Natl. Acad. Sci. USA, Vol. 81, pp. 3088 - 3092, (1984).

[6] W. Li and N.M. Nasrabadi. "Object Recognition Based on Graph Matching Implemented by a Hopfield-style Neural Network," IJCNN, Vol. 2, pp. 287-290. (1989).

[7] G.A. Tagliarini and E.W. Page. "Solving Constraint Satisfaction Problems with Neural Networks," Proc. IEEE First Int'l Conf. on Neural Networks, Vol. 3, pp. 741-747, (1987).

[8] A. Wong and R. Salay. "An Algorithm For Constellation Matching," 8th Inter. Conf. on Pattern Recognition, Vol 1. 546-554, (1987).

[9] S. Ranade and A. Rosenfeld. "Point Pattern Matching By Relaxation," Pattern Recognition, Vol. 12, 269-275, (1980).

[10] Y. Uesaka. "On The Stability of Neural Network with the Energy Function Induced from a Real-Valued Function of Binary Values," IEICE Technical Report PRU 88-6, 7-14,(1988). (in Japanese)

An Application of Neural Networks to Impulse Radar Waveforms from Asphalt-Covered Bridge Decks

G. Vrckovnik, T. Chung, C.R. Carter

Communications Research Laboratory
McMaster University
Hamilton, Ontario
Canada L8S 4K1
(416) 525-9140 x7268

Abstract

A multi-layer neural network is used to classify impulse radar waveforms from asphalt-covered bridge decks. The back-propagation algorithm is used to train a 3 layer network to identify whether a waterproofing membrane is present, and to identify the specific structure from which the waveform originated. On average, the neural network performed better than the nearest neighbour classifier. Very encouraging results were obtained with correct classification percentages ranging between 78.96% and 95.52%.

Introduction

Impulse radar is a special type of radar which is particularly useful in the remote sensing of objects which are buried, encased in concrete or other material, or hidden behind walls or other structures. It has been used in a wide variety of applications such as locating geological structures, detecting underground pipes and cables, and detecting cadavers. Testing has also demonstrated the ability of impulse radar to detect deterioration in concrete bridge deck slabs that are covered with bituminous surfacing [1]. It is possible to use the radar signal to estimate the thickness of the asphalt and the concrete cover over the reinforcement. These quantities are important in determining the amount of asphalt and concrete to be removed when repairing a bridge deck. Both the asphalt thickness and the concrete cover over reinforcement may vary over a bridge deck, and no simple non-destructive evaluation method has previously existed to measure these variations.

A large percentage of bridge decks have bituminous surfacing. Deterioration occurs mainly at or beneath the asphalt-concrete boundary and is of three types. The first problem is debonding which results when the asphalt layer separates from the concrete surface, usually producing an extra boundary. The second type of damage is scaling, which is induced by the freeze-thaw process causing the disintegration of the concrete into a gravelly matrix. The third fault is delamination which occurs as a result of corrosion of the embedded reinforcement. In all three cases, the impulse radar signal reflects from the damaged area producing changes in the reflected waveform which differ from undamaged areas.

The *Ministry of Transportation of Ontario (MTO)* and the *Communications Research Laboratory (CRL)* at *McMaster University* have become actively involved in the use of impulse radar as a tool to detect various forms of deterioration of asphalt-covered bridge decks through a program called *Deck Assessment by Radar and Thermography (DART) [2].* Computer algorithms for assessing bridge deck conditions have been developed for DART and tested using data sets obtained from different bridge decks. One of the main problems in processing the radar reflections, however, is in distinguishing between the different asphalt structures which are found to exist in different bridge decks. These include normal two-lift asphalt over concrete over rebars (*2ACR*), normal two-lift asphalt over waterproofing membrane over concrete over rebars (*2AMCR*), and three-lift asphalt over concrete over rebars (*3ACR*). Previous studies have shown that radar waveforms may be interpreted incorrectly if there is insufficient information about the bridge deck structure [3]. The error is usually due to improperly identified reflections from the concrete

surface and rebars. Consequently, both the measurements for asphalt thickness and concrete cover over reinforcement will be incorrect.

The current operation for DART includes a policy of taking at least one core sample on every bridge deck being surveyed. The core sample thickness and structure is incorporated into the signal processing algorithm in order to maximize the information content. However, this procedure deviates from the concept of adopting non-destructive evaluation of the bridge decks. For the project to be more effective, it is believed that existing procedures for data processing can be enhanced by using neural networks to provide information about the bridge deck structure.

Radar Waveforms

The electromagnetic waves transmitted by the impulse radar penetrate and propagate through the bridge deck surface until they intercept a boundary. A boundary in this case, is any discontinuity (such as a crack or separation) or regions of differing dielectric (such as air to asphalt, or asphalt to concrete). At the boundary, a portion of the incident energy will be reflected and a portion will be transmitted through the boundary.

In Fig. 1a, a waveform from a normal ***2AMCR*** structure is shown. The peaks labelled A,M,C and D indicate the boundaries of the asphalt surface, membrane surface, concrete surface and the rebars respectively. Note that there is a distinct characteristic W between the region from peaks C and D. In Fig. 1b, a waveform from a damaged region of the same bridge is shown. The W shaped reflection is not well defined between peaks C and D. This waveform indicates delamination.

A waveform from a normal ***2ACR*** structure is shown in Fig. 2a. Peak A is the reflection from the air-asphalt boundary; peak C is the reflection from the asphalt-concrete boundary; peak D is the reflection from the rebars. The characteristic W between peaks C and D indicates that the concrete surface is undamaged. When damage is present, the W shape becomes distorted as shown in Fig. 2b.

Fig. 3a illustrates the waveform from a ***3ACR*** structure. Peaks 1 and 2 are the reflections from the top-middle and middle-bottom layers of asphalt, whereas peak 3 (ie. C) is the reflection from the concrete surface, and peak 4 (ie. D) is from the rebars. Again we see a well defined W between peaks 3 and 4. In Fig. 3b a waveform from a damaged region of this bridge is shown.

In all of the illustrated waveforms, the peak from the air-asphalt boundary (peak A) is well defined. This portion of the waveform does not give us any useful information as to the structure which lies beneath the asphalt surface. Only the portion of the waveform after the first negative peak after peak A provides information about the deck's structure. Hence, we truncated the 'front' portion of the waveform up to the first negative peak after peak A. The remaining waveform contained 140 samples and this was used as the input to the neural networks. The waveforms are collected in lines along a bridge deck and 225 waveforms from 4 lines of each of the three decks were obtained for use in this project. Hence a database of 2700 waveforms was available.

Detection of the Waterproofing Membrane

The first network was created with the purpose of identifying the presence, or absence of a waterproofing membrane between the asphalt and concrete boundaries. A fully connected 3 layer neural network was created. It was trained by using the back-propagation algorithm [4]. Training was performed on the WARP systolic array computer using software provided by Carnegie Mellon University. The input layer contained 140 neurons (1 for each sample of the waveforms). The hidden layer contained 25 neurons and 1 output neuron was used. The desired response of the output neuron was a 1 if the waveform's bridge deck had a membrane, and 0 if no membrane existed.

Two different training sets were created. The first set (*SET A*) contained 30 waveforms from 2 of the 4 lines per bridge deck. Hence *SET A* contained a total of 180 waveforms. The second set (*SET B*) also contained 180 waveforms but there were 15 waves from each of the 4 lines per bridge deck. Training the network with *SET A* was complete after 3000 epochs and 97.78% of the training patterns were learned correctly. Using *SET B* to train the network only

required 2000 epochs of training, and 100% of the training patterns were learned correctly. Once training was completed, all 2700 waveforms were presented to the network for classification. The percentage of correct classifications, for the net trained with *SET A* and the net trained with *SET B* are listed in Table 1. The results of the nearest neighbour classifier are also indicated in Table 1.

TABLE 1: Identification of Waterproofing Membrane

	SET A used for Training		*SET B* used for Training	
	NEURAL NET.	NEAREST NGBR.	NEURAL NET.	NEAREST NGBR.
2AMCR	86.11%	82.78%	94.33%	95.36%
3ACR	86.33%	86.22%	94.89%	91.56%
2ACR	92.22%	95.78%	97.33%	96.89%
avg.	**88.22%**	**88.25%**	**95.52%**	**94.00%**

Identifying Bridge Deck Structure

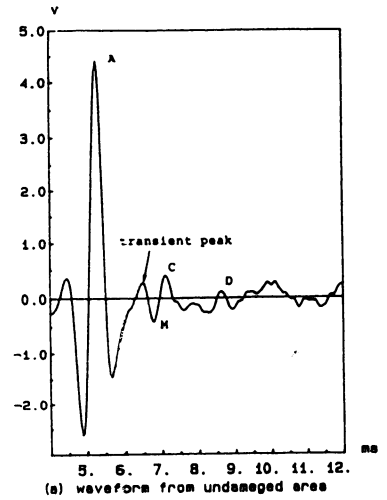
This network was created to identify the structure of the bridge deck from which the waveform presented originated. Once again, back-propagation trained on the WARP computer was employed. The network contained 140 input neurons, 25 hidden neurons in 1 layer, and 2 output neurons. The desired outputs were: 00 if the waveform came from a ***2AMCR*** structure; 01 if the waveform came from a ***3ACR*** structure; 11 if the waveform came from a ***2ACR*** structure. The same two training sets as above (*SET A* and *SET B*) were used to train two different networks. Training with *SET A* was completed after 4000 epochs and 97.78% of the training patterns were learned correctly. Training with *SET B* was also performed for 4000 epochs and this time 99.44% of the patterns were learned correctly. Table 2 shows the percentage correct classification of all 2700 waveforms by the networks trained with *SET A* and *SET B*. Nearest neighbour classification results are also given.

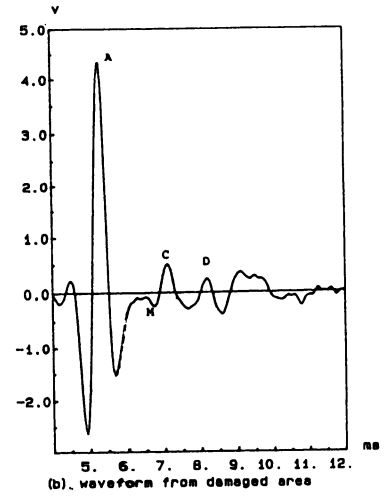
TABLE 2: Identification of Bridge Deck Structure

	SET A used for Training		*SET B* used for Training	
	NEURAL NET.	NEAREST NGBR.	NEURAL NET.	NEAREST NGBR.
2AMCR	88.77%	82.78%	94.56%	93.56%
3ACR	63.77%	52.56%	90.67%	79.78%
2ACR	84.33%	92.44%	84.56%	92.89%
avg.	**78.96%**	**75.92%**	**89.92%**	**88.74%**

Conclusions

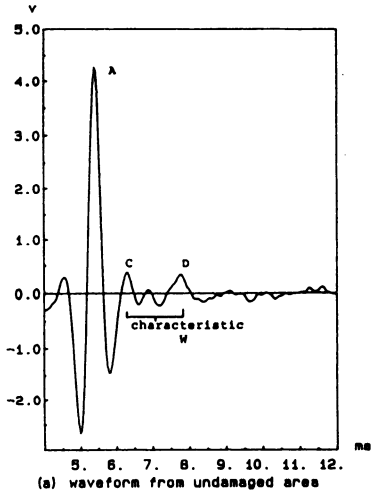
A neural network using the back-propagation algorithm was successfully trained to detect the presence of a waterproofing membrane in an asphalt covered bridge deck. A second neural network was able to distinguish between three different types of bridge decks with a high success rate. Although the nearest neighbour classifier also worked well, the neural network is much more efficient and flexible. The performance of the networks improves if the training data contains sample waveforms from the entire bridge deck surface. Clearly neural networks can be used to extract information about a bridge deck's structure when waveforms from the deck are presented to it.

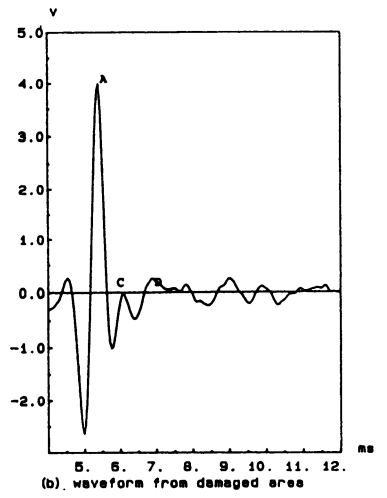




Waveforms from the (a) undamaged area and (b) damaged area of a bridge deck having normal two-lift asphalt over membrane over concrete structure.

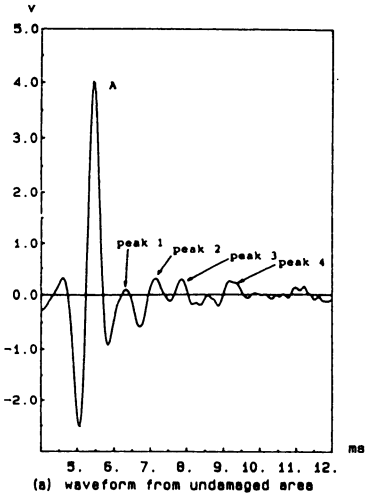
Fig. 1

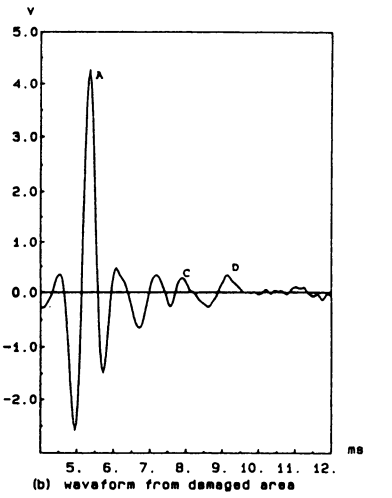




Waveforms from the (a) undamaged area and (b) damaged area of a bridge deck having normal two-lift asphalt over concrete structure.

Fig. 2





Waveforms from the (a) undamaged area and (b) damaged area of a bridge deck having three-layer asphalt over concrete structure.

Fig. 3

References

[1] Manning, D.G., and Holt, F.B. (1983)
Detecting Deterioration in Asphalt-Covered Bridge Decks.
Transport. Research Record 899, Washington D.C., Jan. 1983, 10-20.

[2] Carter, C.R., Chung, T., Holt, F.B., and Manning, D.G. (1986)
An Automated Signal Processing System for the Signature Analysis of Radar Waveforms from Bridge Decks. *Canadian Electrical Engineering Journal, Vol. 11, 3* (1986), 128-137.

[3] Carter, C.R., Chung, T., Masliwec, A., and Manning, D.G. (1988)
Analysis of Radar Reflections from Asphalt-Covered Bridge Deck Structures.
Presented at the Workshop on Ground Probing Radar, Ottawa, Ontario, May 24-26, 1988.

[4] Rumelhart, D.E., and McClelland, J.L. (1988)
Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundation. Cambridge, MA: M.I.T. Press, 1988.

Feature Detector and Application to Handwritten Character Recognition

Xiao-yan ZHU, Kouichiro YAMAUCHI, Takashi JIMBO and Masayoshi UMENO

Department of Electrical & Computer Engineering, Nagoya Institute of Technology, Showa-ku, Nagoya, 466, JAPAN

ABSTRACT

A pattern recognition model using feature detectors is described, which can be approximated by a multilayered main network and an additional network. Some feature detectors are used in these networks to extract features from a given image as preprocessing. Feature extraction ability of the feature detector, the effect of the additional network and the discrimination ability of the system are discussed. Experiment were carried out on handwritten digit character recognition. The recognition rate of 93% is obtained by using only main network and 95% by supplemented with the additional network for handwritten digit characters.

1. Introduction

Many pattern recognition researches have studied neural network[1][2]. Perceptron learning rules is one of the earliest simple models of learning, but was proved to have serious disadvantages[3]. One limitation of perceptron is that they can only associate patterns those are completely uncorrelated. Some of the other models such as Neocognitron and error propagation models, do not have this problem, but are computationally expensive[1][4].

Thinking of the simplicity of the perceptron and the discrimination ability of the backpropagation learning rule, we propose a pattern recognition model that uses feature detector trained by a learning rule like as perceptron to extract a priori features, and a two-layer network trained by backpropagation learning rule to discriminate according to the features. Using special feature detectors and independent learning method make the architecture and the training of the system simple. In order to compensate errors in practical application, an additional network is employed. After the learning process of main network, the recognition can also be helped by the additional network, if there are some error that should be corrected. The advantages in this model are that the feature detector can be trained for any shape which is thought to be a feature in an image and that the additional network can be used to improve the recognition rate without training the whole system again.

2. Feature Detector

The feature extraction process is made by feature detectors called f-cell, of which one type responds to only a special shape, such as an end of line, cross point, etc.. The f-cell takes a weighted linear sum of the input signals with which it connects, and compares with a threshold to determine the output level (0 or 1). The output of f-cell U can be expressed as

$$U=F(\Sigma WjXj-\theta), \qquad (1)$$

$$F(x)=\begin{cases} 1 & x>0 \\ 0 & x\leqq 0, \end{cases}$$

where, Xj is j-th input signal, Wj is the connecting weight between the j-th input signal and the f-cell, and θ is threshold.

The f-cell is trained with a set of sample patterns. For the sample patterns which the f-cell is expected to respond to, there is a teacher signal being equal to 1, for the others the teacher signal is equal to 0. An example of sample pattern set with their teacher signal T is shown in Fig. 1. The f-cell learns by changing in the connecting weights. Initially the weights are randomized. While presentation of the sample pattern, the weights are changed as following.

When $U>0$ and $T=0$ or $U\leqq 0$ and $T=1$,

$$W_j(t)=W_j(t-1)-\varepsilon U(t)X_j,$$
$$\theta(t)=\theta(t-1)+\varepsilon U(t), \qquad (2)$$

otherwise,

no change in weights and threshold,

where ε is a constant which determines the rate of learning. Six types of f-cell are trained to respond to the end of line, the cross point, oblique line "／" and "＼", horizontal line "—" and vertical line "|", respectively. Four types are used in the main network and the others in the additional network, as described in followings.

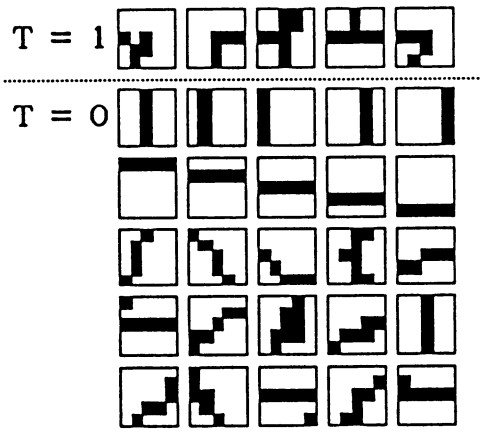


Fig. 1 Example of sample patterns use to training f-cell detecting a cross point.

3. Architecture

The system consists of a main network and an additional network. The main network can be divided into two parts, preprocessing and recognition. The task of the preprocessing part is the transformation of data from the input pattern space into a new space. Patterns transformed to this space are then classified by the recognition part.

3.1 Main Network

There are two layers in preprocessing part, layer C1 and C2. Four types of f-cell are employed in layer C1 as units, and each type of cell forms a plane which respond to the end of line, the cross point and the oblique lines, respectively. The group of input signals with which the unit connects, is referred as the receptive field. Fig. 2 shows the unit plane in layer C1, the receptive fields of each layer, and the interconnection among the input pattern, layer C1 and C2. The outputs of f-cells determined by Eq. 1 express the local features of the input pattern. Here, the receptive field of one f-cell consists of 5*5 pixels in the input pattern, and that of adjacent f-cell is the same size pixel area but shifted by one pixel. The input pattern is completely covered with these receptive fields of one plane. There are four such planes corresponding to the type of f-cell.

Each plane in layer C1 is divided into 3*3 regions with same size and each region is a receptive field of one unit in layer C2. The units in layer C2 connect with layer C1 locally and collect the local features contained in its receptive field to produce a representative feature as its output. The local features are finally compressed into a feature vector (36 dimensions), which is the input of the following network. The output of the units in layer C2 can also be expressed by Eq. 1, where all connecting weights are equal to 1.

The recognition part is a simple multilayered neural network consisting of a hidden layer C3 and an output layer C4, and receives input signals from layer C2. Each unit in output layer expresses one class of recognized patterns. The system classifies the input pattern according to the unit with maximum activity. The connecting weights of this part are determined by using backpropagation learning rule.

Using main network is enough to accomplish the pattern recognition. In practice, however, sometimes the pattern which we want to recognize is slightly different with the training pattern so that it may be misclassified and the recognition rate becomes lower. Although, training the system again would make better, but it cost a lot of time and is impossible for a hardware system of which the value of connecting weights are fixed. Here, an additional network is proposed such that helps the discrimination

3.2 Additional Network

The additional network have two layers, called layer C1′ and C2′ of which the structure is the same as that of the preprocessing part of the main network. There are two types of f-cell that respond to additional features: horizontal and vertical line, respectively. The units of output layer C4 take a weighted linear sum of the outputs of layer C3 and C2′ , before putting it through a sigmoidal activation function F^4. The outputs of them O^4i can be given by

$$O^4i=F^4(\Sigma Wji O^3j+\Sigma Wsi' O^2s' -\theta), \quad (5)$$

where, O^3j and O^2s' are the outputs of j-th units in layer C3 and s-th unit in layer C2′ , Wij and Wij ′ are weights that connect i-th unit in output layer C4 with j-th unit in later C3 and s-th unit in layer C2′ , respectively, and θ is the threshold. The additional network affects the discrimination through the connecting weights Wij′ , which are set to zero initially.

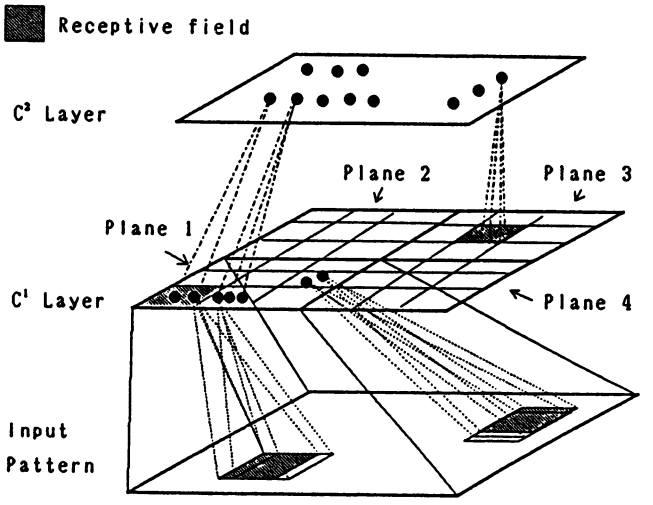


Fig. 2 Interconnection among input pattern, layer C1 and C2

For training the additional network, a set of standard patterns (P1, P2, ... Pk) is prepared , and the additional features expressed as outputs of layer C2′ are used. Pattern Pk is a model pattern of k-th class of the patterns to be recognized. Assume an input pattern Pα of s-th class is misclassified as be m-th class. Let Oj(i) is the output of j-th unit in layer C2′ while input pattern is Pi, then vector O(O1(α), O2(α), ...On(α)) is the additional feature vector for pattern Pα and O(O1(s), O2(s), ..., On(s)) for the corresponded standard pattern Ps. Only the weights connected with m-th and s-th units in output layer are changed as

$$\begin{aligned} Wmi' (t)&=Wmi' (t-1)-cOi(s)(1-Oi(\alpha)) \\ Wsi' (t)&=Wsi' (t-1)+cOi(s)(1-Oi(\alpha)), \\ c&=(Om+\gamma)/(2\Sigma Oi(s)(1-Oi(\alpha))), \end{aligned} \quad (6)$$

where, γ is a constant and t is the training time. The Eq. 6 says that, the weights connecting the additional features, which exist in model pattern but not in misclassified one, with the ideal output unit are increased, and decreased for which connect with the mistaken unit. After training, through the weights the additional features become excitatory signal for the s-th unit and inhibitory signal for m-th unit in layer C4, and do not affects other units. That is why the additional network can supplement the main network for correcting misclassification. The results of the experiments show that the discrimination of the patterns of other classes is affected slightly, after learning the additional network for some patterns. But the effect is very little comparing with the increment of the recognition rate of corrected pattern class.

4. Simulation and Results

A database used in experiments has 1500 digit characters (24*24 pixel) written by 150 people. A training set of 500 characters was generated by selecting 50 characters of each digit from this database, and the others were used for recognizing experiment. There are 22*22*4, 3*3*4, 30 and 10 units in layer C1, C2, C3 and C4, and 22*22*2 and 3*3*2 units in layer C1′ and C2′ , respectively.

Feature extraction ability of the model was tested using 1500 character patterns. The experiments was performed to make feature vectors (36 dimensions) made by four types of f-cell and (45 dimensions) by six types. It is ascertained that the pattern of different classes makes different vectors in both cases. This means that preprocessing model can transform the input pattern from pattern space (24*24=576 dimensions) to feature vector space (36 or 45 dimensions), correctly.

Learning by backpropagation learning rule, the system achieves 100% recognition rate for training pattern at a training time about 100 times. The recognition rate was examined in two cases. First is the case where weight Wij′ are set to zero and then the additional network does not contribute to discrimination. The average recognition rate was 93%. The other case is where the additional network has been trained for 8, 5, 3 and 2 whose recognition rates were lower in former case. The recognition rates of them increased by from 6% to 1%. The recognizing experiment for trained pattern were performed again and the recognition rate were 100%, i.e. no change for using the additional network. Fig. 3 shows the recognition rates in each case.

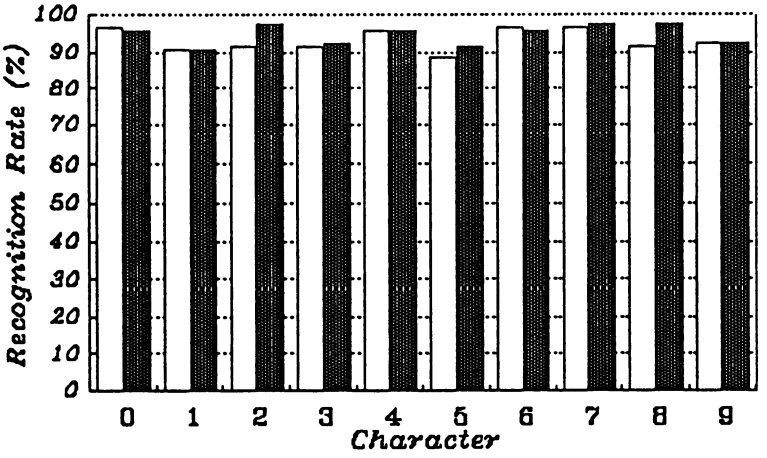


Fig. 3 Recognition rates for digit characters by main network without (white bar) and with (black bar) additional network

5. Conclusion

A model using feature detector was discussed and proved to be useful for handwritten digit character recognition. In addition to the main network, we proposed an additional network which supplements main network for some pattern easily misclassified. This can be applied in the pattern recognizing hardware system to supplement the system for device error.

References

1. T.F.Pawlicki, D.Lee, J.J.Hall and S.N.Seihari: Neural Network Model and their Application to Handwritten digit Recognition, Proc. IEEE Second Int. Conf. on Neural Network, San Diego, p. II-63 (1988).
2. K.Yamada, H.Kami, J.Tsukumo and T.Temma: Handwritten Numeral recognition by Multi-layered Neural Network with Improved Learning Algorithm, Proc. IJCNN Int. Joint Conf. on Neural Network, Wash. D.C. p. II-259 (1989).
3. Block, H.D.: The Perceptron, a model for brain function I, Rev. of Modern Physics, **34**, p.123 (1962).
4. K.Fukushima: Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, Biol. Cybernetics, **36**, p.193 (1980).

Expert Networks and Other Real-World Applications

Integrating Neural Networks and Knowledge-Based Systems in a Commercial Environment

Joseph P. Bigus
Application Business Systems
IBM Corporation
Highway 52 and 37th St. NW
Rochester, MN 55901

Keith Goolsbey
Dept. of Computer Science
The University of Texas
Austin, Texas 78712

Abstract

Neural networks have been shown to be useful for a variety of tasks ranging from pattern matching and associative memory to optimization. However, they have usually been applied to small problems in tightly constrained environments. This paper presents an example of how neural nets can be used in a commercial office business environment. The problem studied is that of dispatching delivery trucks under weight and volume constraints to minimize the number of trucks required and the total miles travelled. This difficult real-world problem is solved by a hybrid system including IBM AS/400™ data base facilities, AS/400 KnowledgeTool™ (a rule-based expert system), and self-organizing neural networks.

Introduction

While neural networks have displayed impressive performance for a variety of tasks, they have yet to achieve any real level of acceptance in the commercial business environment. This is not due to any defect of neural network technology *per se*, but due to the large scope of any "real" problem faced in such an environment. In many ways this parallels the trials and tribulations that knowledge-based (expert) systems have faced in being accepted into commercial computing environments. Perhaps the most crucial feature of any new technology is, "How can it integrate with existing techniques for problem solving?" Unless a new technology claims to solve all of the pieces of a problem, integration is a necessity.

In the neural net research community, most development and delivery systems are high-powered workstations. However, most companies run their businesses on small and mid-range systems which are optimized for transaction processing throughput and for data base operations. In order to evaluate the usefulness of neural nets in this environment, we used a combination of IBM AS/400 data base functions, IBM AS/400 KnowledgeTool rule-based knowledge processing functions, and self-organizing neural networks.

The IBM AS/400 has over 25,000 systems installed worldwide and is used heavily in commercial office business environments. Thus, it serves as a good test bed for evaluating how neural networks perform in this environment. The AS/400 integrated data base services were used to store customer information such as name, location, order information related to the day of delivery, and the size and weight of the shipments.

AS/400 KnowledgeTool is a rule-based forward chaining inference engine with development and runtime environments on AS/400 systems. It is an OPS5-like tool with procedural extensions which allow it to be integrated with COBOL and RPG III applications. Both KnowledgeTool inferencing and neural network routines are callable from RPG III or COBOL programs.

The Truck Dispatching Problem

Our problem consists of creating a day's distribution route for each of a number of trucks which deliver products from a store to customers. This problem contains three interrelated sub-problems:

(1) Determining the number of trucks to use.
(2) Determining which deliveries to place on each truck.
(3) Determining the route each truck should travel through the city.

Our goal is to minimize the number of trucks needed and to minimize the distance each truck must travel.

The city is a square with coordinates ranging from 0 to 1 West to East and from 0 to 1 South to North, with the store centered at (0.5,0.5). Distributed throughout the city are customers who have ordered deliveries from the store. There are three different types of deliveries which a customer can order; these are mnemonically named Heavy, Big, and Little. The weight and volume of each delivery is tabled below.

DELIVERY	Weight (Units)	Volume (Units)
Heavy	50	10
Big	10	50
Little	1	1

Each day consists of from 30 to 70 deliveries, with 48 on average. Customers order, on average, Heavies and Bigs equally often, but order Littles fives times as often as either Heavies or Bigs. The trucks have capacity constraints of 200 Weight Units and 200 Volume Units, and are only allowed to make 20 deliveries in one day. Clearly, carrying a load of only all Heavies, all Bigs, or all Littles wastes resources. A truck that carries a mixture of deliveries is more productive, and more efficient loading allows fewer trucks to be used. In defining this problem, we also made some simplifying assumptions:

(1) All trucks are considered identical.
(2) A truck performs only one delivery circuit per day.
(3) Each customer receives at most one delivery per day.
(4) There is no limit on the distance any truck can drive.

A Hybrid Solution

Our solution proceeds in four steps. First, we determine the minimum number of trucks that will be needed. Second, we make an initial assignment of deliveries to trucks. This assignment is valid, but not necessarily good. In this step, we also determine if more trucks are needed. Next, we improve the assignments by swapping deliveries between trucks to reduce the distances the trucks must travel. Finally, the travelling salesman problem is solved for each truck to produce the actual route to drive. These steps will now be described in detail.

Number of Trucks Required

After reading the problem data from a customer and delivery database on the AS/400, the bare minimum number of trucks required is first calculated. This step is performed by procedural code upon entering KnowledgeTool. Extra trucks are only added to this calculated minimum if they are deemed necessary later in the initial assignment step. The minimum number of trucks depends simply on the capacities of a single truck and the total number, weight, and volume of all the deliveries for the day. From these totals, we calculate three load factors, LFW, LFV, and LFN, which indicate the number of trucks needed only by weight, volume, and number, respectively. For example, consider a day's total deliveries of 10 Heavies, 14 Bigs, and 42 Littles:

day's totals	single truck capacities	resulting load factors	associated delivery type
total weight = 682	200	LFW = 682/200 = 3.41	Heavy
total volume = 842	200	LFV = 842/200 = 4.21	Big
total number = 66	20	LFN = 66/20 = 3.30	Little

The minimum number of trucks is then the maximum load factor, rounded up. In the above case, LFV implies five trucks at least are needed.

Initial Assignment

The initial assignment of deliveries to trucks is performed next using KnowledgeTool rules. Each truck's centroid, the average position of the delivery locations assigned to the truck, roughly indicates the section of the city travelled by the truck. Since each truck must start at the store, all the centroids are initially (0.5,0.5). The deliveries are then assigned to the trucks by type, according to the load factors. Each load factor has associated with it the delivery type which most influences its value. Clearly, Heavies influence LFW the most, Bigs influence LFV the most, and Littles influence LFN the most. The deliveries are assigned from highest load factor to lowest. Intuitively, this is because it will be harder to assign the deliveries with higher load factors, so they are done first. Within each type, deliveries are assigned from locations farthest away from the store to those closest. This allows the truck centroids to more rapidly separate and produces better initial assignments. Each delivery is assigned to the "best" truck which still has room for the order. The "best" truck has the minimum Weighted Distance from the delivery location, which is defined as follows:

Weighted Distance = alpha* Distance + (1 - alpha)*(%resource used)

Distance is the distance of the delivery location from the truck's centroid. **Alpha** is a parameter which is initially 1 and ranges from 1 to 0. It allows the initial assignment to parametrically favor the truck with the closest centroid, or the "emptiest" truck, or some mixture in between. The **%resource used** varies according to the delivery type (weight for Heavies, etc.), so that "emptiest" indicates the lightest truck when assigning Heavies and the truck with the fewest deliveries when assigning Littles. After a delivery is assigned to a truck, its centroid is updated.

Two types of backtracking are implemented. First, a high value of **alpha** may result in inefficient loading and leave no room on any truck for a particular delivery. When this occurs, **alpha** is lowered to favor assignments to the "emptier" trucks, and the process is repeated. If **alpha** is already low, an inability to assign a delivery indicates that not enough trucks are being used. A new truck is added, alpha is set back to 1, and the process repeats. The initial assignment is concluded when all deliveries have been assigned to trucks.

Mopping Up

The "mopping up" step is also performed using KnowledgeTool rules. By this time, a valid assignment of all the deliveries has been made, although it may not be very good. The purpose of this step is to improve the routes by swapping deliveries between trucks so that all delivery locations are as close to their truck's centroid as possible. There are three types of swaps used:

(1) Two trucks swap deliveries of the same type.
(2) Two trucks swap deliveries of different types.
(3) One truck gives a delivery to another.

These swaps are performed if the distance from the delivery locations to their respective truck centroids would decrease if the assignment was made. Clearly, swap #2 and swap #3 can only be performed if the resulting swap does not exceed either truck's capacity. These three swaps are sufficient to eliminate overlap of truck routes in all but the most exceptional cases. In order to reduce the search space for the swaps, certain deliveries are "locked" and not considered in swapping. These deliveries are those for which all other truck centroids are more than twice as far away as its own. Intuitively, it is very unlikely that such a delivery would get reassigned to another truck so far away. After "mopping up", the trucks are relatively balanced with little overlap between their areas of coverage.

Planning a Truck's Route

The problem of finding each truck's delivery route is solved using neural networks. Finding the shortest path between points using neural networks has been studied in detail. Hopfield (1985) has proposed a method which finds good solutions but does not scale well. Durbin and Willshaw (1987) proposed a method using a variation of Kohonen's feature maps called elastic nets. We implemented a variation of the method proposed by Angeniol et. al. (1988) which is also based on feature maps.

This method dynamically allocates and deletes nodes as it trains. It is based on a self-organizing map, with output nodes competing for each delivery location. If any particular node wins twice in any epoch, a twin is created with the same input weights. An exponential gain function is used to progressively decrease the range and size of weight modifications of nodes near the winning unit. This implements the neighborhood function of Kohonen (1984). A single parameter, **alpha**, is used to set the number of steps (i.e. how fast the gain exponentially decays) which affects the quality of the answer. A smaller **alpha** for a given initial and final gain would result in more steps being taken and would give a better answer.

The solution returned by the network is dependent on the order of training data. Five random orderings of the delivery locations for each truck were presented to the network. From this, the shortest delivery route was selected.

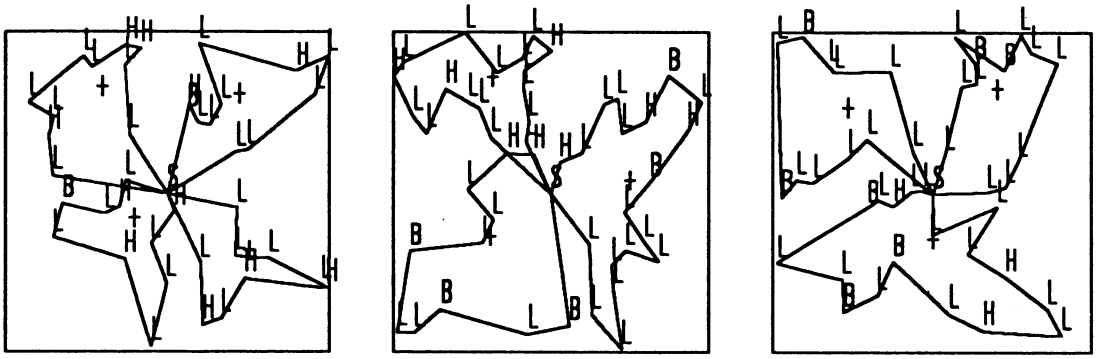
Further Work

In our original formulation of the solution to this problem, we used the rules to perform the initial assignment of items to trucks and to perform the "mopping up" operations which fine tune the delivery assignments. However, we feel that it would be possible to use neural networks for both of these functions.

We are currently testing modified network models which perform the initial assignments in a single training pass. This is expected to provide a substantial performance improvement. We are also studying the possibility of combining the assignment, weight and size constraints, and routing calculations into one network formulation.

Sample Results

Below are three sample runs showing the final assignments and routes for each truck. The H's, B's, and L's indicate Heavy, Big and Little deliveries, the + 's indicate the truck centroids, and S shows the store's location at the center of the map.



Conclusions

In answer to the question, "Can neural networks be useful in a commercial office business environment?", the answer is a qualified yes. Neural networks have been used in several transaction processing applications such as credit scoring and risk analysis. Moreover, the data base facilities of most data processing systems can be used to provide training data for the network.

However, neural networks can only do small parts of large business problems, so they must be integrated with the other software development tools on the host system. Careful selection of neural models is also required, since training some networks is very compute intensive. This can degrade system response times, which is unacceptable if a business depends on order entry or transaction processing applications. However, this can be circumvented by training the network during off-peak hours.

The combination of strong data base services, rule-based expert systems, and neural networks have been shown to be synergistic. When thoughtfully applied to problems in the commercial business environment, neural networks can play a significant part in providing a total solution.

References

IBM AS/400 KnowledgeTool User's Guide and Reference, SC21-9849

Durbin R., & Willshaw, D. (1987), *An analogue approach to the travelling salesman problem using an elastic net method.* Nature 326, 689-691.

Kohonen, T. (1984), *Self-Organization and associative memory.* Berlin: Springer-Verlag.

Angeniol, et. al.(1988), *Self-Organizing Feature Maps and the Travelling Salesman Problem*, Neural Networks, Vol. 1, Number 4, 289-293.

Hopfield,J.J.,& Tank,D. (1985), *"Neural" computation of decisions in optimization problems.*, Biological Cybernetics, 5., 141-152.

A Connectionist Network for Color Selection

James R. Chen
chen%cs@ucsd.edu
Richard K. Belew
rik%cs@ucsd.edu
Cognitive Computer Science Research Group
Computer Science & Engr. Dept. (C-014)
Univ. California at San Diego
La Jolla, CA 92093
Gitta B. Salomon
gitta@apple.com
Human Interface Group
Apple Computer, Inc.
Cupertino, CA 95014

Abstract

A connectionist network was developed to assign appropriate colors to screen elements of the Apple Macintosh II interface. The network was trained with a small set of colorings designed by experts, yet could handle a vast number of possible color combinations. We used a multi-layer network with recurrent links to model an asymmetric auto-associative memory. A gradient descent relaxation algorithm was derived to ensure convergence, and was compared to the standard recurrent relaxation. The color network also learned nonlinear continuous mapping among well matched colors, and was capable of generalizing such mapping to colors never used in the training set.

1 Introduction

The Macintosh II system gives users the ability to apply color to the Finder desktop interface. However, users often find it difficult to select appropriate colors for various interface elements, from screen background to a window scroll bar [6]. A good design must take into consideration not only interface element characteristics such as shape, size and location, but also how well colors of these elements match with one another. An auto-associative connectionist network was implemented to aid users in the design of their color screen desktop. Color patterns with ten interface elements were collected from design experts to train the color selection network. A user can then specify colors for any subset of the ten interface elements and the network will "relax" into a state that combines the user's preference with the experts' aesthetics (Figure 1). We expected a limited set of expert designs, which could employ only a small subset of all 2^{48} different colors supported by the system. Thus our goal was not just to implement an associative memory, but to have a network that can be trained to "design" a color screen, namely, to generalize to new color patterns when a user selects colors or combinations of colors never used in the training set. Such requirements necessitated a non-standard network architecture.

2 Color Net

A two-layer feedforward network with recurrent links is used to model an asymmetric continuous auto-associative memory (Figure 2). The colors of ten interface elements are represented by ten sets of input units as well as ten corresponding sets of output units. Three units are used in each set for the RGB (red, green, blue) encoding of a color. The input units of each element are connected to the output units of the other nine elements, but not to their own corresponding output units. This ensures that the network

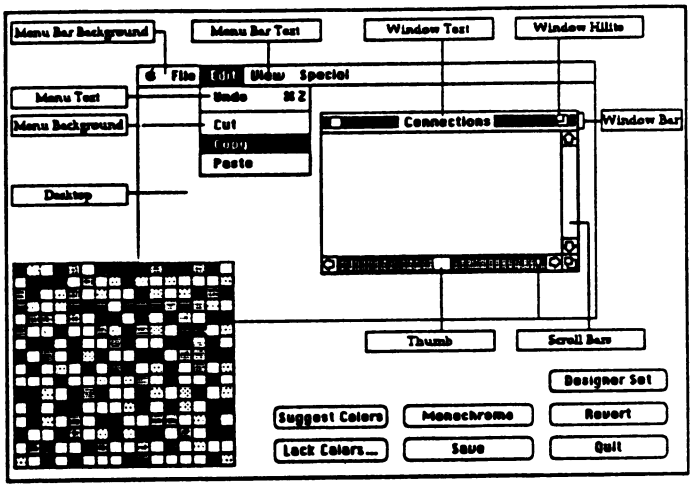


Figure 1: Color Net Interface

cannot learn to simply copy an input pattern onto the output directly, nor can it learn to compress an input pattern at the hidden layer and restore it at the output layer [2]. The set of connections between each pair of interface elements consists of complete (nine) connections from the input RGB units of one element to three hidden units, and nine connections from these hidden units to the output RGB units of the other element. Recurrent links are used to propagate the activation value of each output unit back to its corresponding input unit. Iterations of such recurrent activations until the network reaches a stable state are often referred to as relaxation. The configuration of the color net can be thought of as a ten-node Hopfield net [3] with complete asymmetric connections, each modified with local hidden units to facilitate nonlinear mapping between two sets of RGB values.

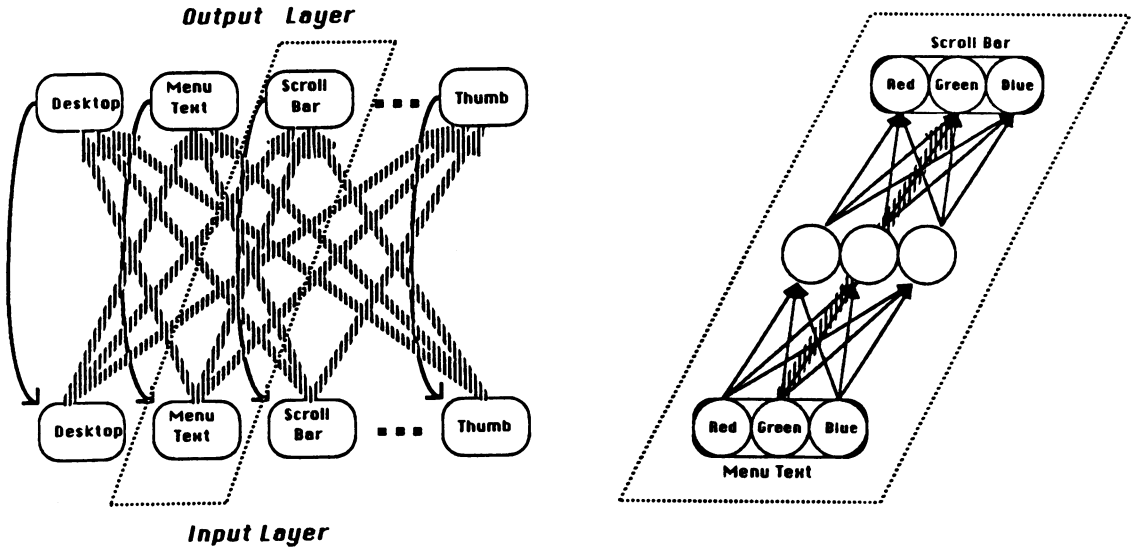


Figure 2: Color Net Architecture

Back propagation [5] was used to train the network with thirty-six designer patterns. Each pattern was presented as both input and target output to the network, ie. error of an output unit was defined as the difference between its activation and the value of its corresponding input unit. The recurrent links were not involved during training. The total squared-error of all output units were taken as a measure of "color incoherence". The network learned the incoherence measure and created basins of attraction in the

relaxation space. The simple non-recurrent learning enforced smooth landscapes within and across basins and thus facilitated generalization of relations among colors of interface elements, without specific designation of units to represent clamped versus unclamped colors.

3 Gradient Descent Relaxation

The standard recurrent relaxation, ie. the process of repeatedly feeding output activations back to their corresponding input units, is not guaranteed to converge asymptotically to a stable state due to asymmetric connections [4]. A new algorithm that followed steepest gradient descent along the relaxation space was derived to minimize color incoherence and to ensure convergence. In addition to recurrently updating each input unit according to the activation of its corresponding output unit, this relaxation algorithm made further adjustments according to errors back propagated from output units of other interface elements.

Define the global error

$$E = \sum_i E_i = \sum_i \frac{1}{2}(i_i - o_i)^2$$

where i_i and o_i denote the value of an input unit i and the activation of its corresponding output unit.

To follow steepest gradient descent, we need to take the partial derivative of the global error with respect to the value of each input unit.

$$\frac{\partial E}{\partial i_i} = \sum_{j \neq i} \frac{\partial E_j}{\partial i_i} + \frac{\partial E_i}{\partial i_i}$$

Since the input color units of an interface element are not connected to its own corresponding output units,

$$\frac{\partial E_i}{\partial i_i} = i_i - o_i$$

and

$$\sum_{j \neq i} \frac{\partial E_j}{\partial i_i} = \sum_k \frac{\partial \sum_j E_j}{\partial net_k} \frac{\partial net_k}{\partial i_i} = \sum_k \frac{\partial \sum_j E_j}{\partial net_k} w_{ik}$$

where net_k denotes the net input (sum of weighted incoming activations) of the kth hidden unit connected from unit i, and w_{ik} denotes the weight associated with the connection between input unit i and hidden unit k.

$\frac{\partial \sum_j E_j}{\partial net_k}$ can be obtained through error back propagation as in the well known back propagation learning algorithm.

Thus each input unit i adjusts its value by

$$\Delta i_i = -\eta \frac{\partial E}{\partial i_i}$$

at each time step until the network reaches a stable activation state asymptotically. For this application, the relaxation rate η was set in proportion to the logarithm of the global error to speed up convergence.

4 Result

The trained color net was able to generate quite pleasing color combinations. The network clearly detected the topographical relations among the interface elements from the training set, as the color of an element affected most strongly the colors of adjacent areas. Also, the network consistently preserved sharp contrast between texts and backgrounds. As an associative memory, the network performed pattern completion well when more than half of the ten colors were clamped by the user. With few colors clamped, the network still did well on noise filtering, but often settled into spurious memories if the initial colors of the unclamped

units were totally random. The color net also generalized well upon colors never used in the training set. When one or more of the colors of a stored pattern were changed and clamped by the user, the network settled into color designs where the unclamped colors underwent similar changes. This is likely the result of interpolation on the numerical mapping between the RGB values of colors. When dissimilar colors were used and clamped by the user, the suggested color design often combined fragments of different stored patterns.

The gradient descent relaxation algorithm compared favorably with the standard recurrent relaxation method. The standard method updated unclamped units according to the activations suggested by other units, but ignored errors on the clamped units, whereas gradient descent relaxation minimized total error and distributed error among all units, clamped or unclamped. Such difference was not essential during memory recall, when the values of the clamped units were part of a training pattern. However, when novel colors were selected and clamped, i.e., when the network was expected to generalize, the standard method could not minimize the total error and often relaxed into a less coherent state. In theory, the standard recurrent relaxation may not converge due to asymmetric connections, but that never happened in our experience. Nevertheless, since such relaxation did not necessarily descend along the color incoherence landscape, the system momentarily jumped from one basin to another before convergence. The gradient descent relaxation, on the other hand, exhibited much better local stability of stored point attractors when few units were clamped.

5 Summary

While only a handful of training patterns were available, the color selection network was expected to handle an exceedingly large number of possible initial color sets, with any number of colors clamped. Bearing this in mind, we used a feedforward network with back propagation learning, and gradient descent relaxation, to model an asymmetric auto-associative memory. With limited storage capacity, the network was designed to learn nonlinear relationships among well matched colors, and to generalize such relationships. We think such approach can be applied as well to other situations, where human expertise is best captured by a connectionist network.

References

[1] T. N. Cornsweet. *Inside Visual Perception.* Academic Press, New York, 1970.

[2] G. Cottrel, P. Munro, and D. Zipser. Inside image compression by back propagation. Technical report, Institute of Cognitive Science, UCSD, 1987.

[3] J. J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proc. Natl. Acad. Sci. USA*, 81:3088–3092, May 1984.

[4] F. J. Pineda. Dynamics and architecture for neural computation. *Journal of Complexity*, 4:216–245, 1988.

[5] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group. *Parallel Distributed Processing*, volume 1. The MIT Press, 1986.

[6] A. Wagner and G. Salomon. Color finder survey results: Implications for changes to system software. Proprietary technical report, Human Interface Group, Apple Computer, 1988.

Deductive and Inductive Learning in a Connectionist Deterministic Parser

Kanaan A. Faisal & Stan C. Kwasny

Center for Intelligent Computer Systems[1]
Department of Computer Science
Washington University, St. Louis, MO 63130

INTRODUCTION

Deterministic Parsing of Natural Language, as performed by PARSIFAL (Marcus, 1980), has shown that such a task can be conducted under the somewhat severe restriction of the determinism hypothesis using a rule-based approach. Others have independently extended PARSIFAL to the task of parsing ungrammatical sentences in PARAGRAM (Charniak, 1983), resolving lexical ambiguities in ROBIE (Milne, 1986), and acquiring syntax in LPARSIFAL (Berwick, 1985).

Why have these researchers chosen to focus on extensions with these rather narrow goals? The answer, in part, lies in the general difficulty of the task and the limitations of conventional, symbolic means. We have found it beneficial to combine these tasks into one implementation which is partly symbolic and partly sub-symbolic. The results of our experiments hold implications for rule-based expert systems.

A Connectionist Deterministic Parser (CDP) is under development (Kwasny & Faisal, 1989). CDP combines the concepts and ideas from deterministic parsing together with the generalization and robustness of connectionist, adaptive (neural) networks. A backpropagation neural network simulator, which features a logistic function that computes values in the range of -1 to $+1$, is being used in this work. The ultimate goal is to produce a parser that has some reasonable facility with language and does not fail on inputs that are only slightly different from the inputs it is designed to process.

There are important advantages to constructing rule-based systems using neural networks (Gallant, 1988). Our focus is on building a connectionist parser, but with more general issues in mind. How successfully can a connectionist parser be constructed and what are the advantages? Success clearly hinges on the careful selection of training sequences. Our experiments have examined two different approaches and compared them.

The "deductive" strategy uses rule "templates" derived from the rules of a deterministic grammar. It is deductive in the sense that it is based on general knowledge in the form of rules although the resultant network is required to process specific sentences. The "inductive" strategy derives its training sequence from coded examples of sentence processing. It is inductive in the sense that it is based on traces of the processing of specific sentences but is required to generalize to a wider range of examples. The goal of deductive learning initially is to produce a network which is capable of mimicking the rules on which training is based. The goal of inductive learning initially is to produce a network which is capable of processing the sentences on which its training is based. Once that learning has been accomplished, simulation experiments are done to examine certain generalization capabilities of the resulting networks.

Deductive training generally performs well on all generalization tasks and outperforms inductive training by scoring generally higher on all experiments. Reasons for this include the specificity of the inductive training data as well as the lack of a large amount of training data in the inductive case required to provide sufficient variety.

LEARNING A RULE-BASED GRAMMAR

A deterministic parser applies rules to a stack and buffer of constituents to generate and perform actions on those structures. One of its primary features is that it does not backtrack, but proceeds forward in its processing never building structures which are later discarded.

Training of CDP proceeds by presenting patterns to the network and teaching it to respond with an appropriate action using backpropagation (Rumelhart, et al, 1986). The input patterns represent encodings of the buffer positions and the top of the stack from the deterministic parser. The output of the network contains a series of units representing actions to be performed during processing and judged in a winner-take-all fashion. Network convergence is observed once the network can achieve a perfect score on the training patterns themselves and the error measure has decreased to an acceptable level (set as a parameter).

[1] The sponsors of the Center are McDonnell Douglas Corporation and Southwestern Bell Telephone Company.

Once the network is trained, the weights are stored in a file so that various experiments can be performed. Each sentence receives a score representing the overall average strength of responses during processing. The score for each processing step is computed as the reciprocal of the error for that step. The error is computed as the Euclidean distance between the actual output and an idealized output consisting of a **-1** value for every output unit except the winning unit which has a **+1** value. The errors for each step are summed and averaged over the number of steps. The average strength is the reciprocal of the average error per step.

Deductive Learning. Each grammar rule is coded as a training template which is a list of feature values, but templates are not grouped into rule packets. In general, each constituent is represented by an ordered feature vector in which one or more values is ON(**+1**) for features of the form and all other values are either OFF(**-1**) or DO NOT CARE (**?**). A rule template is instantiated by randomly changing **?** to **+1** or **-1**. Thus, each template represents many training patterns and each training epoch is slightly different. During training, the network learns the inputs upon which it can rely in constructing its answers and which inputs it can ignore.

The probability of a **?** becoming a **+1** or **-1** is equal and set at 0.5. Each rule template containing n **?**'s can generate up to 2^n training cases. Some rule templates have over 30 **?**'s which means they represent approximately 10^9 unique training cases. It is obviously impossible to test the performance of all these cases, so a zero is substituted for each **?** to provide testing patterns. Zero is used since it represents the mean of the range of values seen during training.

A slightly modified version of the grammar from appendix C of Marcus (1980) was used as a basis for all deductive training experiments in this paper. This appendix includes the rules specifically discussed by Marcus in building his case for deterministic parsing and can be taken as representative of the mechanisms involved. To assure good performance by the network, training has ranged from 50,000 to 200,000 presentations cycling through training cases generated from the rule templates.

Inductive Learning. Inductive learning begins with training data derived as "sentence traces" of deterministic parsing steps. Training proceeds by presenting patterns of these steps to the network and teaching it to respond with an appropriate action. A small set of positive sentence examples were traced which resulted in 64 unique training patterns. These were used for all inductive experiments in this paper.

This approach can be compared with LPARSIFAL which attempts to learn PARSIFAL rules from examples of *positive evidence* (i.e., grammatical sentences). LPARSIFAL starts with a small set of rules and gradually builds up new rules. In effect, the system is inductively learning the grammar rules from sentence examples. The target for learning in LPARSIFAL is the PARSIFAL grammar. LPARSIFAL requires several hundred sentences to acquire approximately 70% of the parsing rules originally hand-written for the Marcus parser. In our experiments, the network exhibited better than 70% coverage of our rule-based grammar after training on a small number of traces.

NETWORK ARCHITECTURE

Patterns consist of a list of syntactic features, divided into four groups to match the three buffer positions and the top of the stack. These are represented in a localist manner in the network with each syntactic feature being represented by a unit. The choice of a localist representation allows the grammar to be represented in a very straightforward manner and permits experimentation with sentence processing in a direct way.

In the set of experiments described here, the network has a three-layer architecture with 35 input units, 20 hidden units, and 20 output units. Each input pattern consists of three feature vectors from the buffer items and one stack vector. Each vector activates 14 input units in a pattern vector representing a word or constituent of the sentence. The stack vector activates seven units representing the current node on the stack. In our simplified version of the grammar, only two items are coded from the buffer and thus 35 input units are sufficient. One hidden layer has proven sufficient in all of our experiments. The output layer represents the 20 possible actions that can be performed on each iteration of processing. All weights in the network are initialized to random values between –0.3 and +0.3.

During sentence processing, the network is presented with encodings of the buffer and the top of the stack. What the model actually sees as input is not the raw sentence but a canonical representation of each word in the sentence in a form that could be produced by a simple lexicon, although such a lexicon is not part of the model in its present form. The network produces the action to be taken which is then performed. If the action creates a vacancy in the buffer and if more of the sentence is left to be processed then the next sentence component is moved into the buffer. The process then repeats until a stop action is performed, usually when the buffer becomes empty. Iteration over the input stream is achieved in this fashion.

PERFORMANCE COMPARISON

CDP is capable of processing a variety of simple sentence forms such as simple declarative, simple passive, and imperative sentences as well as yes-no questions. For test and comparison purposes, several sentences were coded that would parse correctly by the rules of the deterministic parser. Also, several mildly ungrammatical and lexically ambiguous sentences were coded to determine if the network would generalize in any useful way. The objective was to test if the syntactic context could aid in resolving such problems.

TABLE 1: Grammatical Sentences Used in Testing

	Sentence Form	Deductive Average Strength	Inductive Average Strength
(1)	John should have scheduled the meeting.	283.3	84.7
(2)	John has scheduled the meeting for Monday.	179.3	84.2
(3)	Has John scheduled the meeting?	132.2	64.4
(4)	John is scheduling the meeting.	294.4	83.5
(5)	The boy did hit Jack.	298.2	76.2
(6)	Schedule the meeting.	236.2	67.8
(7)	Mary is kissed.	276.1	84.9
(8)	Tom hit(v) Mary.	485.0	80.3
(9)	Tom will(aux) hit(v) Mary.	547.5	78.7
(10)	They can(v) fish(np).	485.0	80.3
(11)	They can(aux) fish(v).	598.2	76.8

Parsing Grammatical Sentences. Grammatical sentences, by our definition, are those which parse correctly in the rule-based grammar from which we derived the training set. Table 1 shows several examples of grammatical sentences which are parsed successfully along with their response strengths in both deductive and inductive learning. Each example shows a relatively high average strength value, indicating that the training data has been learned. Also, the deductive average strength value is higher than the corresponding inductive average strength.

TABLE 2: Ungrammatical Sentences Used in Testing

	Sentence Form	Deductive Average Strength	Inductive Average Strength
(12)	*John have should scheduled the meeting.	25.1	6.6
(13)	*Has John schedule the meeting?	38.1	18.2
(14)	*John is schedule the meeting.	4.7	4.9†
(15)	*The boy did hitting Jack.	26.6	7.5‡

Parsing Ungrammatical Sentences. An important test of the generalization capabilities of CDP is its response to ungrammatical sentences. Such capabilities are strictly dependent upon the experiences of the network during training since in deductive training no relaxation rules were added to the original grammar to handle ungrammatical cases and in inductive training no ungrammatical sentences were used.

In this set of experiments a few ungrammatical sentences were tested that were similar to the training data and within the scope of our encoding. Table 2 contains examples that have produced reasonable structures when presented to our system along with their response strengths. Note that overall average strength is lower for ungrammatical sentences when compared to similar grammatical ones.

In sentence (12), the structure produced was identical to that produced while parsing sentence (1), but with lower strength in the inductive case. The only difference is that the two auxiliary verbs, *have* and *should*, were reversed. Sentence (13) contains a disagreement between the auxiliary *has* and the main verb *schedule* and yet the comparable grammatical sentence (3) parsed identically in both approaches, but with lower strength again in the inductive approach.

Sentence (14) can be compared with sentence (4). In the deductive case, a structure similar to that built for sentence (4) is indeed constructed. However, in the inductive case (marked with †), the network attempts to process 'is' as if it were indicating the passive tense. Although this is incorrect for this sentence, it is not an unreasonable choice. Sentence (15) can be compared with sentence (5), but there is not one clear choice in how the sentence should appear if grammatical. The deductive-trained network processes sentence (15) as sentence (5), while the inductive result (marked with ‡) shows the sentence processed as if it were progressive tense ('The boy is hitting Jack'). In PARAGRAM, a nonsensical parse structure is produced for sentence (15), as reported by Charniak (p. 137).

TABLE 3: Lexically Ambiguous Sentences Used in Testing

	Sentence Form (Words in <> are presented ambiguously)	Deductive Average Strength	Inductive Average Strength
(16)	<Will> he go?	83.6	14.3
(17)	Tom <will> hit Mary.	118.7	19.9
(18)	Tom <hit> Mary.	39.0	2.5
(19)	They <can> fish.	4.5	2.6
(20)	They can <fish>.	172.2	4.9

Lexical Ambiguity. In a final set of experiments, the parser was tested for its ability to aid in the resolution of lexical ambiguity. Grammatical sentences were presented, except that selected words were coded ambiguously to represent an ambiguously stored word from the lexicon. These examples are shown in Table 3. Several of these examples come from ROBIE.

Sentence (17) contains the word *will* coded ambiguously as an NP and an auxiliary, modal verb. In the context of the sentence, it is clearly being used as a modal auxiliary and the parser treats it that way. A similar result was obtained for sentence (18). In sentence (19), *hit* is coded to be ambiguous between an NP (as in playing cards) and a verb. The network correctly identifies it as the main verb of the sentence. Sentence (20) presents *can* ambiguously as an auxiliary, modal, and main verb, while *fish* is presented uniquely as an NP. *Can* is processed as the main verb of the sentence. Compare this example with sentence (10) of Table 1. Here, each word is presented unambiguously with *can* coded as a verb and *fish* coded as an NP. The same structure results in each case, with the average strength level much higher in the unambiguous case. By coding *fish* ambiguously as a verb/NP and coding *can* uniquely as an auxiliary, the result obtained is as shown for sentence (21), which is comparable to sentence (12).

In the cases shown, the lexically ambiguous words were disambiguated and reasonable structures resulted. Note that the overall average strengths were lower than comparable grammatical sentences discussed, as expected. Also, the deductive average strength value is higher than inductive average strength.

DISCUSSION

While deductive training exhibits better performance than inductive training for all tasks, there are tradeoffs in the two approaches. Deductive training requires rules as the basis for rule templates while inductive training requires a large amount of data to be successful. Fortunately there is a middle ground which allows mixtures of the two training strategies. Training can be performed using rule templates as well as patterns based on sentence traces. In a recent set of experiments in which the two types of training data were combined, the network was capable of generalizing in ways similar to deductive learning, but also showed particularly good performance on the specific cases reflected in the inductive data.

What does this mean for expert systems? Where knowledge naturally exists in rule form and such rules can be reliably stated, rule templates can be formed which generate appropriate training cases. However, where knowledge only exists in the form of anecdotal cases, it can be expressed in the form of inductive training patterns. As new cases are discovered for which the rules do not apply, inductive data can be easily constructed and the network retrained. Contrast this with the typical rule-based expert system in which each new rule may require re-thinking an entire set of existing rules.

Our work has shown the trade-offs between deductive and inductive learning. Both have a place in the construction of a neural network designed to perform a complex rule-based task such as parsing.

REFERENCES

Berwick, Robert C. (1985). *The Acquisition of Syntactic Knowledge*, Cambridge: MIT Press.

Charniak, E. (1983). A Parser with Something for Everyone. In King (Ed.), *Parsing Natural Language*, New York: Academic Press.

Gallant, Stephen I. (1988). Connectionist Expert Systems. *Communications of the ACM 31*, 2, 152-169.

Kwasny, S.C., and Faisal, K.A. (1989). Competition and Learning in a Connectionist Deterministic Parser. *Proceedings of the 11th Annual Conference of Cognitive Science Society*, Ann Arbor, Michigan.

Marcus, M. P. (1980). *A Theory of Syntactic Recognition for Natural Language*. Cambridge: MIT Press.

McClelland, J. L., & Rumelhart, D. E. (1988). *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*, Cambridge: MIT Press.

Milne, R. (1986). Resolving Lexical Ambiguity in a Deterministic Parser. *Computational Linguistics 12*, 1-12.

Rumelhart, D. E., Hinton, G., & Williams, R.J. (1986). Learning Internal Representations by Error Propagation. In Rumelhart & McClelland *Parallel Distributed Processing*. Cambridge: MIT Press.

COMPILING HIGH-LEVEL SPECIFICATIONS INTO NEURAL NETWORKS
(Extended Abstract)†

P. Myllymäki H. Tirri P. Floréen P. Orponen

Department of Computer Science
University of Helsinki
Teollisuuskatu 23
SF-00510 Helsinki FINLAND

1. INTRODUCTION

Currently most of the neural network research concentrates on the adaptive properties of neural networks [RuMc 86]. However, as recent research in machine learning theory clearly indicates, general learning, in particular learning of higher concepts from raw data, is extremely difficult and time-consuming (see for example [PiVa 88]). Hence it seems that the most promising approach would be to develop hybrid methods, combining both symbolic and neural techniques. The symbolic component can then be used for complicated inference (for which current neural techniques are completely inadequate), while the neural component provides a basic robustness to knowledge representation even in the presence of noisy or incomplete data. Some recent attempts towards this goal are reported in [Dert 88], [Died 88], [DoSm 88], [Dyer 88] and [Hend 89].

Inspired by this general idea, we will introduce a knowledge representation scheme for robust storing of concepts. This scheme can eventually be embedded in a Prolog programming environment. Our work is in the same tradition as Shastri's [Shas 88], but with several important differences. Firstly, our approach is based on a distributed representation of concepts (see e.g. [HiMR 86]), as opposed to the local representation adopted in [Shas 88]. Secondly our computational process is an (efficient) relaxation, whereas Shastri computes activation values by an ordered sequence of sweeps over the network. As we will show, these differences provide our system with a greater flexibility to use all the relevant information during the retrieval process. *Our main point is to demonstrate the ability of this particular representation scheme to perform limited inference in the presence of partial and even inconsistent information.* As this "inference by similarity" is a property of our representation structure and the computational mechanism used, it is completely transparent to the user.

2. COMPILING A HIGH LEVEL LANGUAGE TO A NEURAL REPRESENTATION

The term "neural network programming" is usually taken to mean very low-level operations to set interconnection weights and activation levels. Our approach is a radical departure from this tradition. *We argue that neural networks can be programmed for concept property inference by giving a structure with similar notational convenience as in e.g. logic programming.* On the other hand, since a neural computational structure is used, the language is not restricted to describe only necessary properties, as in logic based concept specification languages, which have to be augmented with awkward additional constructs for inference with incomplete information – witness the well-known problems of handling uncertainty in rule-based expert systems.

NEULA (NEUral LAnguage) is designed for robust representation of concepts with a possible associated inheritance structure. Complex inferences are intended to be performed by a more traditional inference engine, which can retrieve the information stored in the knowledge base described with *NEULA* by simple queries. However, *part of the inference process is transferred from the explicit inference engine to the structure of the knowledge representation.*

The concept structures specified in the language are *labelled collections of (property,value) pairs.* The values of attributes are also concepts, and thus concepts may be arbitrarily complex. For instance, a concept labelled Hobbit may be specified as follows (this example is a part of a larger specification):

† This research is supported by Technology Development Center (TEKES) in Software Technology Programme (FINSOFT), and by the Academy of Finland.

```
CONCEPT HOBBIT IS MIDDLE_EARTH_INHABITANT.
  ~ NATURE                GOOD.
  ~ HEIGHT                SHORT.
  ~ NOT IS_FOND_OF        SWIMMING(MANY).
  ~ NOT IS_FOND_OF        FIGHTING.
  ~ IS_FOND_OF            BIRTHDAY_PARTIES,ROUND_THINGS.
  ~ LANGUAGE              COMMON_SPEECH.
  ~ NOT LANGUAGE          SECRET_LANGUAGE.
  ~ LIFE                  MORTAL.
  ~ STRENGTH              DURABLE.
  ~ HAVE_BUILDINGS        LOW_HOUSES.
  ~ NOT HAS_ENEMY         DRAGON.
```

The concept structures specified in the language can be thought of as *labelled collections of (property,value) pairs*. Hence one of the properties of a concept Hobbit is 'nature' which has value 'good'. These (property,value) pairs can be interpreted as features typical to the concept in question, i.e. Hobbit as a concept is associated to the (property,value) pairs represented by the above list. *NEULA* allows the use of graded values to describe the degree of certainty of a particular feature. For instance,

```
  ~ NOT IS_FOND_OF        SWIMMING(MANY).
```

is to be understood that not being fond of swimming is typical to most Hobbits but not necessarily to all (the default value is 'ALL'). In addition, some simple logical relations between the concepts can also be expressed. These language elements will be converted to a connectionist network by the *NEULA* compiler.

In the implementation of *NEULA* we assign one unit in our neural network to each of the statements in a specification to be compiled. Hence a concept C is realized as a distributed collection of *property units*, $C = \{(P_1,V_1),(P_2,V_2),...,(P_n,V_n)\}$. To bind together all the different property units we add to this set a special *label unit (label,C)*. From the label unit we then draw directed arcs to all the other property units of concept C. For each positive statement "C ~ P V [W]" we create an arc with the weight determined by the corresponding certainty factor W. For the negative statements "C ~ NOT P V [W]" we create similar arcs, but with equally sized negative weights.

The compilation scheme above can be understood as a method for creating connections from a concept C to values V_i through properties P_i. As the values V_i are also concepts defined in the network, the links drawn can be interpreted as implementing associations from concept C to concepts V_i. To allow bidirectional associations, we similarly associate concepts V_i with concept C. An obvious choice for the property through which these connections are created is the inverse relation of P, denoted with P^{-1}. According to this idea, the compiler adds implicitly for every compiled sentence "C ~ P V [W]" the code corresponding to the sentence "V ~ P^{-1} C [W]".

If the activation level of a label unit is high, we interpret that the network represents the corresponding concept. Similarly, if the activation level of a property unit is high, then the network is understood to represent some concept having this specific property. During the computation, the activation is propagated along the arcs between different units in the network, as described by the activation propagation formula discussed in the sequel. The arcs, having either a positive (excitatory) or a negative (inhibitory) weight, impose an inference order in the network. However, due to the iterative nature of the activation propagation process, the arcweight from unit i to unit j cannot be interpreted simply as the probability of the statement "if (P_i,V_i) then (P_j,V_j)". Nevertheless, an arcweight indicates the approximate effect on the occurring inference. To be able to carry out similar inference for concept recognition, i.e. to recognize an object from its properties, arcs from the property units to the label units are also required. This is accomplished by creating an *echo arc* for every existing arc. The direction of an echo arc is opposite to the original one, but with a proportional weight smaller than that of the original arc. In our experiments, we have simply divided the original weight with a small constant, usually 5.

Implementing the hierarchical relation "CONCEPT C IS C' " is somewhat different from the method to implement properties. As concept C may appear as a value of several different properties $R_1,...,R_n$, to make the hierarchy influence each of these properties, we have to draw an arc from all the units (R_i,C) to the units (R_i,C'), respectively. If only one of these units exists, the other one is automatically created. In particular, as this connection is also created between the corresponding label units (R_k='label', for some k), we have created a path from the property units of C to those of C' (see Figure 1). In other words, we have linked an instance of a class to the class itself, thus implementing the hierarchy of concepts.

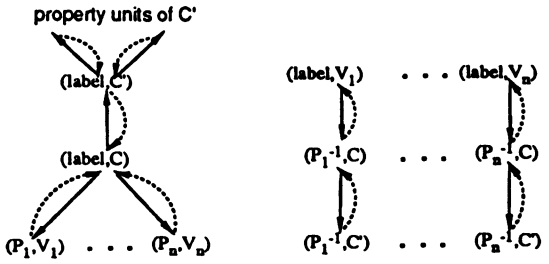


Figure 1. The connectionist network implementing the specification of concept C, given that C IS C'. Notice that the property unit sets for C and C' do not necessarily have to be disjoint.

A property unit (P_i,V_i) is generally connected to several different label units. As the concepts corresponding to these label units may be hierarchically related to each other, the compilation scheme offers a natural way to implement exceptions in the inheritance of properties.

The inference in the network is carried out by *activation propagation*. The idea is that initially a user query is compiled in such a way that some corresponding nodes are turned on (initiated with activity level 1) and some are turned off (initiated with activity level -1). The other nodes have activity level 0. All the nodes then synchronously change their activity level according to the *activation propagation formula (1, below)*. The resulting activity level depends on the activity level of the nodes from which there are incoming arcs, and on the weights on those arcs. The activation propagates through the network, and spreads from the initialized nodes to other nodes. This process is continual, thus each node will change its activity level iteratively. The arcweights remain unchanged during the computation. For computing the change in the node activity level, we use a formula of the general form

$$a_i^{k+1} = a_i^k + (1 - |a_i^k|)\ sigm(\sum_j w_{ij} a_j^k)\ . \qquad (1)$$

(by a_i^k we denote the activity level of node i after the k^{th} update, and by w_{ij} the weight on the connection from node j to node i). In the formula the sum states that each node j incident to node i affects the activation of node i in proportion to the weight on the arc from j to i. If the connection w_{ij} is highly positive, node i tends to get an activation similar to that of node j, and if w_{ij} is negative, the activity level of node i tends to be opposite to that of node j.

If we use the formula above, the activity level of the nodes with initial activity level 1 or -1 will remain unchanged during the whole computation.

Some kind of *termination conditions* are also needed. One solution is to terminate the computation when the activity level of any of the nodes representing possible answers to the query is within distance ε from 1 or -1, where ε is a limit given in advance. In our experiments with about 300 nodes and $\varepsilon = 0.1$, the computation has always terminated fast, in less than about 10 synchronous steps.

3. AN EXAMPLE

To test the validity of the ideas discussed in the previous Sections, we have simulated a larger example (over 100 statements compiled into a 300 node network). Our implementation is built on the floating-point version of the Rochester Connectionist Simulator 4.1 (RCS) [GoLM 88]. The software runs on a SUN 3/60 workstation under Unix SUNOS 4.0. The *NEULA* declarations are compiled into an intermediate code, which consists of some high-level functions. These functions then call the built-in functions of the RCS to construct the corresponding network. The nodes in the network apply the activation propagation formula (1) in a synchronous order, hence the results are fully deterministic and can be easily reproduced.† In the sequel we present an illustrative example of the actual queries performed against a large declaration. More detailed description of the compilation scheme and its implementation can be found in [FMOT 89].

† Those interested in reproducing the experiments may obtain the source C-code (excluding the RCS-software) by sending a request to tirri@cs.helsinki.fi (Internet).

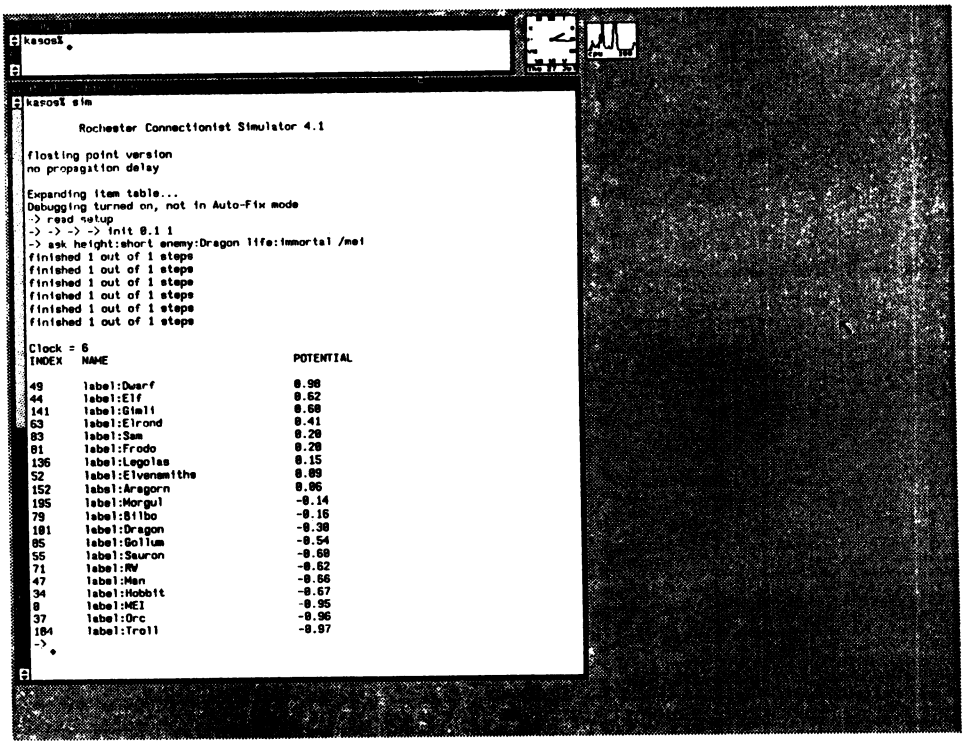


Figure 2. What Middle Earth inhabitant is short, has Dragons as an enemy, and is immortal? The first two properties strongly suggest Dwarfs as an answer, but the immortality would fit Elves better. In spite of this contradiction, both Dwarfs and Elves are suggested (immediately followed by individuals in these classes), but the activity level for the parent class Middle_Earth_Inhabitant is negative.

REFERENCES

[Dert 88] M.Derthick, Mundane Reasoning by Parallel Constraint Satisfaction. Ph.D. Thesis, TR CMU-CS-88-182, Carnegie–Mellon Univ., September 1988.

[Died 88] J.Diederich, Knowledge representation and learning in a structured neural network. Proc. Workshop Konnektionismus, TR 329, GMD St. Augustin, April 1988, 47–62.

[DoSm 88] C.P.Dolan, P.Smolensky, Implementing a Connectionist Production System Using Tensor Products. TR UCLA-AI-88-15, Univ. of California, Los Angeles, October 1988.

[Dyer 88] M.G.Dyer, Symbolic NeuroEngineering for Natural Language Processing: A Multilevel Research Approach. TR UCLA-AI-88-14. Univ. of California, Los Angeles, August 1988.

[FMOT 89] P.Floréen, P.Myllymäki, P.Orponen, H.Tirri, Neural representation of concepts for robust inference. To appear in F.Gardin, M.G.Filippini (eds.), Proceedings of the International Symposium on Computational Intelligence '89. Elsevier, 1989.

[GoLM 88] N.H.Goddard, K.J.Lynne, T.Mintz, Rochester Connectionist Simulator. TR 233, Univ. of Rochester, Rochester, March 1988.

[Hend 89] J.A.Hendler, Marker-passing over microfeatures: towards a hybrid symbolic / connectionist model. Cognitive Science 13 (1989), 79-106.

[HiMR 86] G.E.Hinton, J.L.McClelland, D.E.Rumelhart, Distributed representations, 77–109 in [RuMc 86].

[PiVa 88] L.Pitt, L.G.Valiant, Computational limitations on learning from examples. J. Assoc. Comp. Mach. 35 (1988), 965–984.

[RuMc 86] D.E.Rumelhart, J.L.McClelland (eds.), *Parallel Distributed Processing: Explorations in the Microstructures of Cognition. Vol 1,2.* The MIT Press, 1986.

[Shas 88] L.Shastri, *Semantic Networks: An Evidential Formalization and its Connectionist Realization.* Morgan Kaufmann Publishers, 1988.

Risk Assessment of Mortgage Applications with a Neural Network System: An Update as the Test Portfolio Ages

by
Douglas L. Reilly, Edward Collins, Christopher Scofield, Sushmito Ghosh
Nestor, Inc., One Richmond Square, Providence, RI 02906

Nestor's multiple neural network technology has been applied to many problems; among them, applications in signal processing for character recognition [1], in medicine [2,3], in vision [4], industrial inspection [5,6], diagnostics [7], speech recognition [8]. Neural net applications have also been developed in the financial services arena. One particular problem domain that has been under investigation is that of automated decision-making and risk assessment for mortgage insurance underwriting. The application of Nestor's neural net technology to this problem has been previously reported [9,10,11]. This paper presents an update to this work, returning to the risk assessment portion of the problem to analyze how well the risk prediction has fared for the mortgage portfolio under study, now that the portfolio has aged.

Mortgage risk assessment begins with mortgage origination. A mortgage originator filters the general population of potential property owners according to a set of simple guidelines on acceptable ranges for such risk measures as the proposed loan-to-value ratio, the ratio of proposed obligations to income, etc. Fannie Mae, and others in the secondary mortgage market, publish guidelines that serve to qualify and segregate the home loan applicant pool into risk categories. Some of the higher risk loans are referred for private mortgage insurance. The process of determining whether or not a loan applicant should be accepted for mortgage insurance involves in effect a second underwriting. If the applicant exhibits an acceptable level of risk, he will be sold insurance. Mortgage insurance applicants are by nature a higher risk group than the general population of mortgage applicants. These applicants have already been underwritten by the mortgage originator and assessed as less secure cases. Thus, this second order underwriting performed by the mortgage insurer is bound to be more difficult, and prone to greater uncertainty.

The mortgage insurance problem can be divided in two parts. The first problem is that of automating the decision-making process of the underwriters. This can be served by constructing a system that can learn to emulate the decisions that underwriters make on mortgage applications. A system that correctly mimics the judgement of human underwriters on some subset of the loan applications presented to it can have economic benefit as a result of introducing consistency in underwriter judgements and in allowing an underwriting agency to better handle peak work loads. The strategy of applying a neural network approach to this phase of the problem is to capture data that represents loan application information together with the corresponding judgements that an underwriter has made on each application. The pool of data represents judgements from a number of underwriters. The neural net system trains to emulate the decision-making of this collection of experts on the problem.

A different aspect of the problem arises not from the use of the network in automating the human decision-making, but rather from the use of the network to improve upon the quality of the decisions through its ability to learn to estimate some measure of the risk of a loan applicant's defaulting on his mortgage payments. Underwriters for loan originators and private mortgage insurance companies do not perform this task flawlessly. Although these insurance underwriters typically decline approximately 20% of the applications they review, of the remaining accepted group, some 20% will go delinquent during the course of the loan. Approximately 6% will eventually lead to losses as a result of claims. The peak in claims rates occurs some three to five years after the loan is granted. Because any feedback from an incorrect decision occurs some number of years after the decision is

made, and because of the high turnover rate in underwriter staffing, there is little opportunity to improve on the underwriters' judgements from observations of historical outcomes.

Although the economic payoff of improving the quality of the underwriting decisions can be substantially greater than that of simply automating and replicating their current decision-making trends and practices, the acceptance of the former can require a higher level of commitment and reliance upon the technology. It is relatively easy to immediately verify whether or not the system is deciding as the underwriter would decide. From the simple perspective of "trusting the machine", it requires more commitment to accept that the machine, when disagreeing with the underwriter, is actually making the *better* decision about something that may happen three to five years from now.

The neural network that was used for this application is a derivative of an RCE network that has been reported on previously [12]. The RCE network is a three layer network that has been described extensively elsewhere [13,14,15]. Essentially, the network trains by committing cells on its single internal layer that represent prototypical "exemplars" of the pattern classes that it sees in the training set. It automatically selects from the training set the exemplars that are to be stored in memory, storing in its weights the values that define the prototype features. Associated with each prototype cell is a cell threshold, a number that captures the extent to which this prototype's exemplar will participate in the classification of incoming new patterns. This cell threshold represents a region of influence around the prototype in the pattern space.

Data used for the risk assessment study was taken from a collection of some 111,080 home mortgage loans from the period July 1984 to December 1986. The status of the loans was noted as of December, 1987 and this served as the classification of "Good" and "Bad" loans. For the initial study, "bad" was defined as any loan which had gone delinquent at least once in the period from origination through the end of 1987. A total of 758 good applications and 844 bad applications was used for the risk assessment study.

The results of the initial study are shown in Figure 1. For a certain percentage of the applications (10%) it is possible to predict with 95% accuracy the loans which, if granted, would go delinquent in payment. (Delinquency is not the same as default, but it is a necessary precursor.) If the system's decisions are accepted at this throughput (10% of applications) then some number of good applications will also be called "bad". Rejecting such loan applicants would amount to turning away good business. Since the cost of replacing this lost business is far less than the cost associated with underwriting loans that go to claim, this 10% throughput represents an operating point with a viable economic benefit.

As noted earlier, these results were reported on a portfolio of mortgages whose delinquency/default status was dated as of December 1987. Since the portfolio represented originations between July 1984 and December 1987, the age of the applications at the time of the study ranged from 1 to 3 1/2 years. Updates on the status of this portfolio were provided as of September 1988. Since claims begin to peak in the 3-5 year period after origination, we would expect to see significantly more claims in the updated portfolio. We will present results and analysis of the effectiveness of the neural network risk assessment predictions with regard to the updated status of the portfolio under study. Special attention will be paid to those loans that, at the time of the initial study had been labelled good, but have since gone bad.

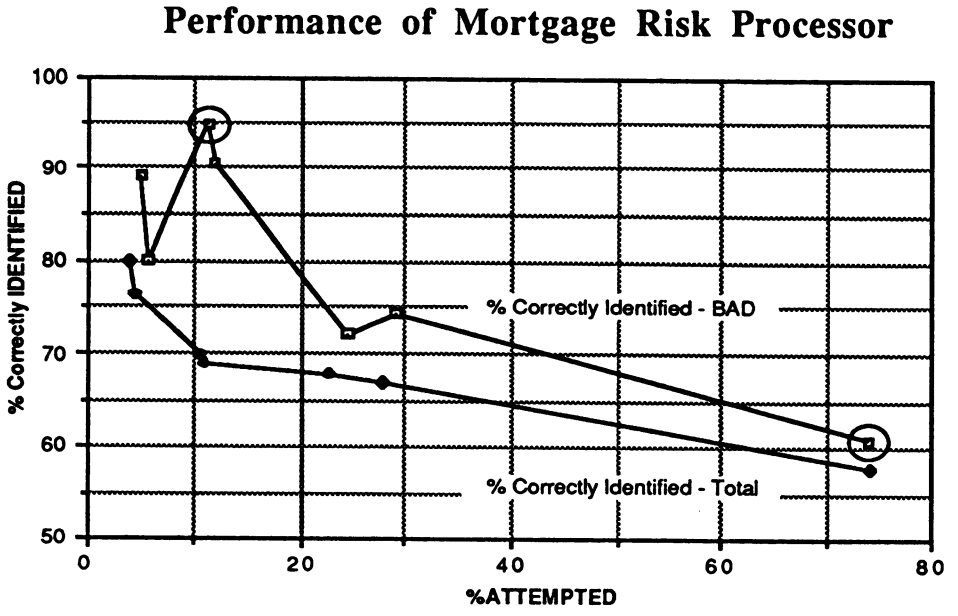


Figure 1

Additionally, we will present experimental evidence on the sensitivity of the model to the size of the training set. The collection of a portfolio for training purposes can often have some cost associated with it, if all the information that is typically available to underwriters is not available in electronic file format for the the neural network to process. Consequently, it can be important to establish some measure of the additional marginal benefit in risk assessment as a function of additional loan examples made available for training. Figure 2 shows a learning curve for the neural net risk assessment system as a function of the percentage of the data set trained on.

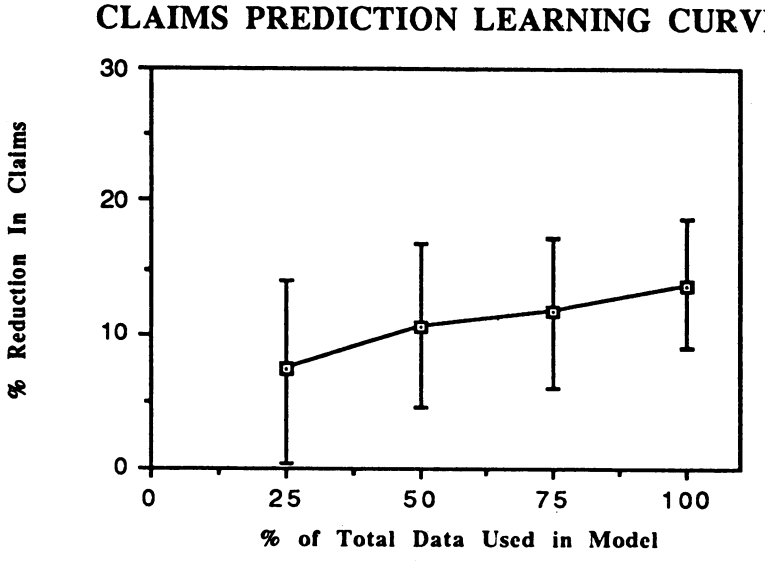


Figure 2

References

[1] D. Ward, C. L. Scofield, D. L. Reilly, " An application of a multiple neural network with modifiable network topology (GENSEP) to on-line character recognition," Abstracts of INNS, Boston, 1988.

[2] T. O. Carroll, H. Ved, D. L. Reilly, "A Neural Network ECG Analysis", Proceedings IJCNN, II-575, June, 1989.

[3] P. Sherman, J. DaPonte, "Using neural networks and expert systems as techniques to computerize ultrasound image recognition," to be published

[4] R. Rimey, P. Gouin, C. Scofield, D. L. Reilly, "Real-time 3-D object classification using a learning system," Proceedings SPIE Cambridge Symposium on Intelligent Robots and Computer Vision, October 26-31 1986.

[5] D. L. Reilly, C. Scofield, P. R. Gouin, R. Rimey, E. A. Collins, S. Ghosh, "An application of a multiple neural network learning system to industrial part inspection," Proceedings of ISA, October ,1988.

[6] R. O. Fox, F. Czerniewjewski, F. Fluet, E. Mitchell, "Neural network machine vision", Abstracts of INNS 1988, vol. 1, p. 438.

[7]. K. Marko, J. James, J. Dosdall, J, Murphy, "Automotive control system diagnostics using neural nets for rapid pattern classification of large data sets", Proceedings IJCNN, vol II, pg13-16, June, 1989.

[8] P. Zemany, W. Hogan, E. Real, D. P. Morgan, L. Riek, D. L. Reilly, C. L. Scofield, P. Gouin, F. Hull, "Experiments in discrete utterance recognition using neural networks", Proceedings of Second Biennial Acoustics Speech and Signal Processing Conference, May 1989,

[9] E. Collins, S. Ghosh, C. L. Scofield, "An application of a multiple neural network learning system to emulation of mortgage underwriting judgements," IEEE Int'l. Conf. on Neural Networks II, 459-466 (1988)

[10] C. Scofield, E. A. Collins, S. Ghosh, "Prediction of mortgage loan performance with a multiple neural network learning system," Abstracts of INNS 1988, vol 1, p. 439.

[11] DARPA Neural Network Survey Study, AFCEA, pp. 429-443, Nov. 1988.

[12] C.L. Scofield, D.L. Reilly, C. Elbaum, L.N. Cooper, "Pattern Class Degeneracy in an Unrestricted Storage Density Memory," Neural Information Processing Systems ed. D. Z. Anderson, 674-682, New York (1985)

[13] D.L.Reilly, L.N. Cooper, C. Elbaum, "A Neural Model for Category Learning," Biol. Cybern. 45, 35-41 (1982)

[14] L.N. Cooper, C. Elbaum, D.L. Reilly, "Self Organizing General Pattern Class Separator and Identifier," U.S. Patent No. 4,326,259. Awarded Apr. 20, 1982.

[15] L. N. Cooper, E. Elbaum, D. L. Reilly, C. L. Scofield, "Parallel, Multi-Unit, Adaptive Nonlinear Pat.tern Class Separator and Identifier," U. S. Patent No. 4,760,604. Awarded July 26, 1988.

[16] D.L. Reilly, C. Scofield, C. Elbaum, L.N. Cooper, "Learning System Architectures Composed of Multiple Learning Modules," IEEE First Int'l Conf. on Neural Networks II, 495-503 (1987)

FUZZNET: TOWARDS A FUZZY CONNECTIONIST EXPERT SYSTEM DEVELOPMENT TOOL

Steve G. Romaniuk and Lawrence O. Hall
Department of Computer Science and Engineering
University of South Florida, Tampa, FL. 33620

Abstract

In this paper we present a raw expert system development tool, based on a connectionist architecture for representing knowledge. Our work is centered around rule based systems as a basis for a connectionist expert system, which can be expanded, and updated through learning of sample domain specific cases [1]. A cell recruitment learning algorithm [4,5] capable of forgetting previously learned facts by learning new ones is incorporated. Using this learning mechanism, we let the system learn the knowledge bases of GEMS (Gem stone classification) and FEVERS (Fevers diagnosis system), examples of which were extracted prior to this, from the corresponding rule knowledge bases. The learned knowledge bases compared favorably with the existing rule based expert systems.

1. FUZZNET -- A Fuzzy Connectionist Expert System

A fuzzy connectionist expert system development tool dubbed FUZZNET [4,5] has been developed. FUZZNET uses fuzzy logic for the implementation of reasoning under uncertainty and takes advantage of a connectionist like architecture for knowledge representation. It has a shell like structure, allowing it to construct expert systems, either by using rules or making use of its learning capabilities. In FUZZNET a variable number of expressions can be placed on the left and right hand side of rules. Information can be deleted, thereby allowing knowledge to be updated. Expressions can be negated, and the ability to activate partial patterns has been included. Furthermore, variables have been included (Attribute, value pairs are supported), which we believe are a necessity in most classification systems, but have not been included in most approaches to constructing connectionist expert systems [1,6]. With the inclusion of variables, we have also addressed the question of how to achieve variable binding, which has now become an integrated part of the tool. By allowing fuzzy and relational comparators, the systems potential has been further increased. The user can construct aggregating quantifiers, and use them within rules, which is a step towards integrating more natural language like constructs. Finally, the system can be used for the generation of stand alone expert systems, for any knowledge base it acquired, either through rule formulation, learning, or a combination of the two.

2. Fuzzy Logic and the Connectionist Model

A connectionist model is a network, which in its simplest format has no feedback loops. It consists of three types of cells (input, output, and hidden cells). Every cell has a bias associated with it, which lies on the real number scale. Cells are connected through links which have weights associated with them. In the FUZZNET model of a connectionist network, each cell can take on an activation value within the range [0..1] (Corresponding to the fuzzy range of membership).

In fuzzy set theory, sets are combined by performing the union, intersection, or set difference. In fuzzy logic we define disjunction (fOR) as the maximum operation, conjunction (fAND) as the minimum operation and complement (fNOT) as strong negation [2]. Since fOR and fAND are defined as maximum and minimum operations, we let certain cells act as *max* and *min* functions, in order to provide for the above operators. In order to be able to distinguish cells as modeling the *min* (fAND) or the *max* (fOR) function we use the sign of the bias of a cell to determine which of the two functions is to be modeled. Furthermore, we denote a bias value of zero to indicate when a cell should operate as an inverter (fNOT). It is important to note, that these complex cells can be reduced to networks of linear threshold cells, which are commonly found in connectionist models.

3. Representing Knowledge in FUZZNET

The FUZZNET tool allows knowledge to be represented either in the form of rules or by learning of domain specific knowledge, via a *cell recruitment learning* algorithm [4,5]. Rules consist of a *premise* and an *action* part. The *premise* contains a list of parameters, which are combined through conjunction and disjunction connectors. The nesting of these connectors can be of arbitrary depth. The *action* part is a list of conclusions, which in case of a positive evaluation of the *premise,* are activated. With every conclusion a fuzzy membership value (We refer to it as the *rule-range*) is associated indicating an upper threshold for the activated conclusion (An example follows next).

if and(or(s1,s2),s3) **then** d1 (0.8);

Only if the premise evaluates to 1.0 (completely true) will the conclusion d1 be derived with a belief of 0.8. In all other cases the final conclusion will be less then 0.8. We will next look at the FUZZNET network model.

We can think of every cell in a network accommodating n inputs I_n with associated weights CW_n. Every cell contains a bias value, which indicates what type of fuzzy function a cell models, and its absolute value represents the previously mentioned *rule-range.*

Every cell C_i with a cell activation of CA_i (except for input cells) computes its new cell activation CA_i' according to the formula given below. If cell C_i (with CA_i) and cell C_j (with CA_j) are connected then the weight of the connecting link is given as $CW_{i,j}$, otherwise $CW_{i,j}=0$.

CA_i -- cell activation for cell C_i, CA_i in [0..1].
$CW_{i,j}$ -- weight for connection between cell C_i and C_j, $CW_{i,j}$ in R.
CB_i -- cell bias for cell C_i, CB_i in [-1,..+1].

$$CA'_i = \begin{cases} \min\limits_{j=0,..,i-1,i+1,..n} \{CA_j * CW_{i,j}\} * |CB_i|, CB_i < 0 & \text{or} \\ \max\limits_{j=0,..i-1,i+1,..n} \{CA_j * CW_{i,j}\} * |CB_i|, CB_i > 0 & \text{or} \\ 1 - CA_j, CB_i = 0 \wedge CW_{i,j} \neq 0 \end{cases}$$

By successively translating rules into sub-networks, and by combining these, we are capable of constructing complete networks that model our rules. In the final process an extra layer of two cells (denoted as the *positive* and the *negative* cell) is placed before every output cell. These two cells will be collecting information for (*positive* cell) and against the presence of a conclusion (*negative* cell). Both cells are connected to every output cell, and every concluding intermediate cell (these are cells defined by the user in the FUZZNET program specification). The final cell activation for the concluding cell is given as:

$$CA_{output} = CA_{positive\_cell} + CA_{negative\_cell} - 0.5$$

Whenever, there is a contradiction in the derivation of a conclusion, this fact will be represented in a final cell activation close to 0.5. For example, if $CA_{positive\_cell} = 0.9$ and $CA_{negative\_cell} = 0.1$, then $CA_{output} = 0.5$, which means it is unknown. If either $CA_{positive\_cell}$ or $CA_{negative\_cell}$ is equal to 0.5, then CA_{output} will be equal to the others cell activation. The FUZZNET system also allows for the construction of quantifying operators. It is for example possible to construct an operator like *almost*, which can be defined as:

almost := **they** *almostall* **are** *completely true*;

and then use it within a rule like the aggregating operators *and* and *or*.

Besides allowing for rules as a knowledge representation scheme, the system allows learning of information. Learning is achieved by a recruitment of cells algorithm [4,5]. It recruits cells from a conceptual pool of fully connected cells, whenever they are needed to represent new knowledge. The learning algorithm can be used in conjunction with an existing rule knowledge base or alone. Learned knowledge can also be extracted in form of rules from the network model. This is extremely important within the explanation phase of the system. Here the user can ask the system how certain conclusions were derived, or why certain questions where asked (when entering the consultation phase).

One of the major reasons for having learning capabilities in a system is to overcome the knowledge acquisition problem and be able to change knowledge whenever this becomes necessary. It also supports the synthesis of different experts, who may not even know of one another, but are all modifying and contributing to the knowledge base of the expert system. In conventional expert system this can represent a serious problem, and special care has to be taken to avoid introducing contradictions and inconsistencies in the knowledge base. We will next describe the learning algorithm used in FUZZNET using the following input vector <s1=1, s2=.8, s3=.5, s4=.5, s5=.2, s6=1, d1=.1, d2=.9> (range for cell values is [0..1]). Here S_i is an input cell and d_i is an output cell.

Let us assume that the activation for an input cell is $S_i < .5$ (like s5), then the input cell activation has to be sent through an inverter cell. This will ensure, that negative evidence will be converted to positive evidence, which is necessary for evaluating the premise of a rule and finally firing the rule. The fact that a piece of information is negative evidence, is not retained in the cell activation of the appropriate cell, but rather in its connection to the *negative* cell, which is used to collect negative evidence. This conversion is shown in Fig. 1a. For the input cell s5 we would obtain the specific network presented in Fig. 1b. If the input x for cell s5<=.2, then a fuzzy value of 1.0 is propagated by that cell. Otherwise, the activation falls within the range [0..1). Let us assume x=0.3, then according to the formula for calculating the cell activation of an inverter cell (CA=1-x) we obtain as an cell activation for the inverter cell in Fig. 1b (CA=1-.3=.7). The value CA is next multiplied with the associated weight 1/.8 from the learn stage (the weight is calculated by the formula 1/(1-Si). In our example Si=.2, therefore we obtain as a weight 1/(1-.2) = 1/0.8). The final output of the cell in Fig. 1b is given as .7/.8 = .875.

One can view the value given to s5 (.2) as being a lower threshold. Only a membership value of .2 or less is necessary to propagate a final value of 1. For the case of S_i being greater than .5 we use a similar argument. Again a lower threshold is used for the input cells. Take s2=.8 for example. If the activation of cell s2 is greater or equal to .8 a value of 1 is passed to the next level of the network. We represent this by a simple arrow up with an associated weight (Fig. 1c).

We next show the cell structure needed for combining all the inputs into one final fuzzy value. We again take separate looks at the combined cells, depending on whether d_i is above or below .5. We present the case of d_i<=.5 first (Fig. 2a).

For the case that d_i>0.5 is given in Fig. 2c. The links labeled with output in our two examples will be fed into the *negative* and *positive* cell of the output layer of the network. If d_i<.5 then the cell is connected to the *negative* cell, and if d_i>.5, it is combined with the *positive* cell.

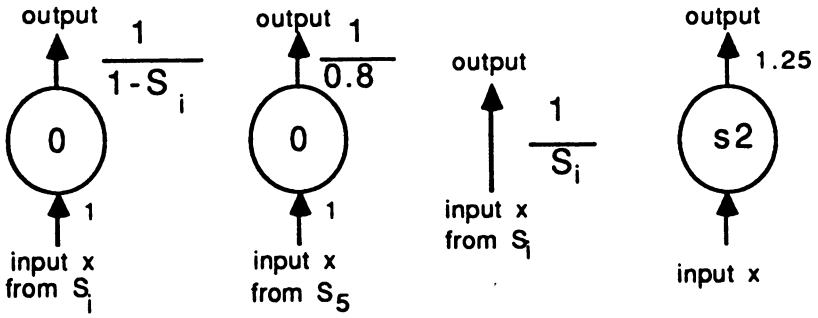


FIGURE 1. a) general inverted input b) inverted input for s_5 c) general non-inverted input d) non-inverted input for s_2.

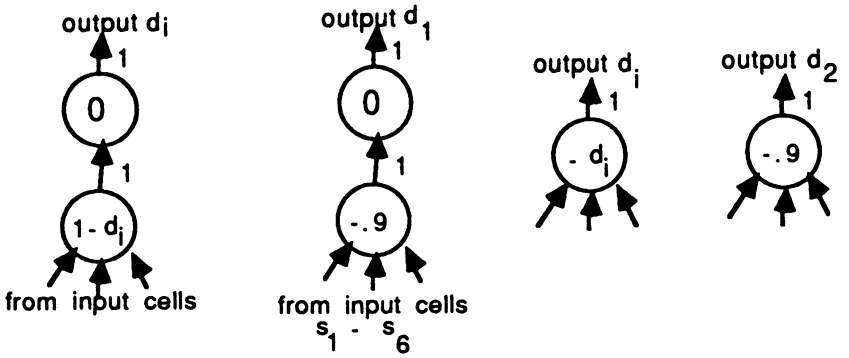


FIGURE 2. a) general inverted accumulate cell b) inverted accumulate cell for d_1 c) general non-inverted accumulate cell d) non-inverted accumulate cell for d_2.

4. Summary and Results

After allowing the system to learn the knowledge bases of GEMS and FEVERS, comparisons where made with a EMYCIN [3] version of these two knowledge bases running under MultiLisp [3]. In all cases the results where almost identical and only differed by a few percent in the certainty factor of the final conclusion. From our tests we can conclude, that FUZZNET is capable of learning knowledge bases from scratch and can achieve similar overall performances as conventional symbolic expert systems. Other features such as consultation and explanation facilities have been added and tested favorably. By allowing variables, fuzzy and relational comparators, and the ability to construct quantifying operators, the systems capabilities have been taken beyond existing connectionist expert systems, and conventional (pure symbolic) expert system shells. Because of its hybrid structure, the system incorporates both the advantages of symbolic and purely connectionist models, were no symbolic information is contained.

References

1. Gallant, S. I. (1988). "Connectionist Expert Systems". Communications of the ACM, Vol. 31, Number 2.
2. Hall, Lawrence O. (1986). "Designing fuzzy expert systems".Verlag TUV Rheinland.
3. Krall, Edward J., McGehearty, Patrick F. (February 1986). "A Case Study of Parallel Execution of a Rule-Based Expert System". Vol. 15, No. 1, International Journal of Parallel Programming.
4. Romaniuk, S. G., Hall, L. O. (1989) "FUZZNET, A Fuzzy Connectionist Expert System." Technical Report CSE-89-07, Dept. of Computer Science and Engineering, University of South Florida, FL.
5. Romaniuk, S. G., Hall, L. O. (1989) "Parallel Connectionist Expert Systems." IASTED International Conference EXPERT SYSTEMS, Theory and Applications, June, 89, Zurich.
6. Samad, Tariq. (1988). "Towards Connectionist Rule-Based Systems". Vol. II, International Conference on Neural Networks.

Interfacing a Neural Network with a Rule-Based Reasoner for Diagnosing Mastitis

Jos F. Schreinemakers*
Department of Information Sciences
Erasmus University of Rotterdam
The Netherlands

David S.Touretzky†
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

1. Introduction.

Production systems have proven to be a powerful tool for exploiting knowledge that can be stated explicitly, via rules. Neural networks offer a complementary ability: they can acquire and generalize from implicit knowledge extracted from training examples. In this paper we present an elegant approach to merging neural networks with production systems. Our solution allows an OPS5-based expert system to selectively invoke neural net inference components for making certain types of decisions which would be difficult to describe with symbolic rules. We also introduce a rule-based knowledge manager that allows an expert system to incrementally improve the competence of its neural network components by acquiring new training examples when appropriate.

2. Overview of ELSIE.

ELSIE is a Common Lisp system that consists of a production system module, a neural network simulation module, and a knowledge acquisition module. The production system used is OPS5 [2]. It has served as the implementation language for a number of classical expert systems, the most famous of which is probably McDermott's XCON/R1 [4], the first commercially successful expert system. The neural network simulator is YANNS, an extensible Common Lisp simulator developed by CMU's connectionist research group under the direction of the second author. The knowledge manager was developed by the first author.

All three modules communicate via OPS5 working memory elements, or WMEs (prounounced "woom-ies"), which are flexible record structures. The fields of a WME may be accessed either by name or by numeric subscript. We interface a neural net to the production system by specifying a mapping between WME fields and the net's input and output layers.

3. An example from a real application.

To illustrate the merger of a rule-based system with a neural net, we briefly describe a veterinary diagnosis task we have been researching as part of a larger agricultural management expert system. The task is to detect clinical and subclinical mastitis (udder infection) in dairy cows [3,6], given a series of measurements of milk production and leucocyte counts (white blood cells per volume of milk).

Most of the work in the sytem is done by a set of production rules programmed in OPS5, plus some auxiliary Common Lisp routines. But the actual diagnosis decision is performed by a 36-6-3 neural net. The first step in interfacing this net to the production system is to give it a name by which it can be referenced in the OPS5 program. The DEFNETWORK construct below accomplishes this. It also records the location of the network descriptor and saved weights files, which must be loaded before the network can be invoked.

---

*This work was supported in part by D.L.O., Department of Agriculture, Wageningen, the Netherlands.

†This work was supported in part by National Science Foundation grant EET-8716324, and by the Office of Naval Research under contract number N00014-86-K-0678.

```
(defnetwork mastitis-net
   :network-file "mastitis.lisp"
   :weights-file "mastitis.save"
   :wme-name 'cow-descriptor
   :fields (cow-number
            calving-date
            (kgs-milk :input 18)
            (cell-counts :input 18)
            (healthy :output)
            (subclinical :output)
            (clinical :output)))
```

The second step in the interfacing process is to declare the format of the WME that will be used to exchange data with the neural net. Each WME has a name (corresponding to a record type in Pascal or C) and some number of fields. The WME-NAME and FIELDS arguments to the DEFNETWORK macro supply this information. As far as the neural net is concerned, there are three types of WME fields. Ordinary fields, such as COW-IDENTIFIER and CALVING-DATE, are ignored by the net. Input fields, such as KGS-MILK or CELL-COUNTS, hold data which is to be copied from the WME into the input units of the neural net when the net is invoked. The notation (CELL-COUNTS :INPUT 18) indicates that the CELL-COUNTS field holds a list of 18 numbers to be copied into the next 18 input units; the layout of the fields in a DEFNETWORK expression must match the layout of the neural net's input and output fields. Output fields, such as the three diagnosis fields HEALTHY, SUBCLINICAL, and CLINICAL, will have their values filled in from the corresponding output layer units at the completion of the forward pass.

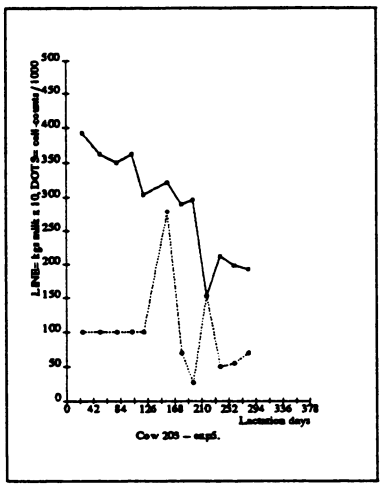


Figure 1: Normal cow.

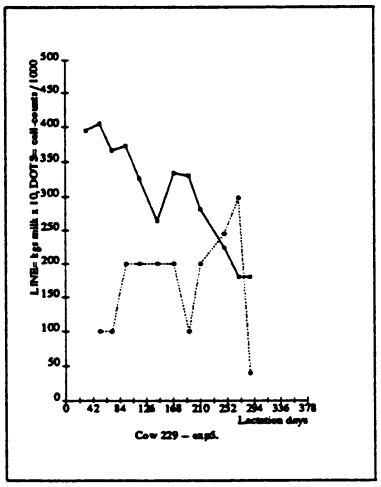


Figure 2: Subclinical.

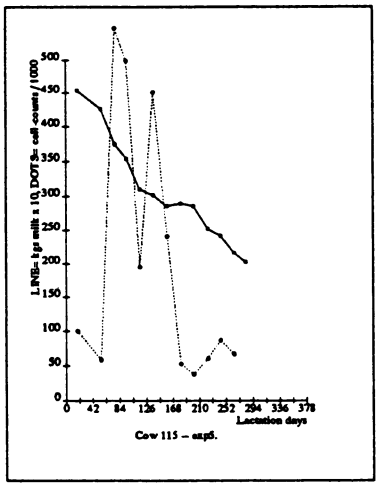


Figure 3: Clinical.

Figure 1 shows a plot of the cell count and milk production levels of a healthy cow during a lactation period. Figure 2 shows the plot of a subclinical mastitis cow, and Figure 3 shows a clinical mastitis case. The solid lines represent milk production and the dotted lines cell counts.

The following OPS5 rule identifies an undiagnosed cow (denoted by NILs in the WME's output fields) and sets up a diagnosis subgoal:

```
(p RECOGNIZE-UNDIAGNOSED
   (cow-descriptor ^healthy nil ^cow-number <n>)
  -->
   (goal ^type diagnose-cow ^number <n>))
```

The next step is to feed the cow data to the neural network. This is accomplished by the rule FEED-COW-TO-MASTITIS-NETWORK.

```
(p FEED-COW-TO-MASTITIS-NETWORK
   {<subgoal> (goal ^type diagnose-cow ^number <n>)}
   {<cow-wme> (cow-descriptor ^number <n>)}
  -->
   (call invoke-network mastitis-net <cow-wme>))
   (remove <subgoal>))
```

The INVOKE-NETWORK function appearing on the right hand side of the rule maps the contents of the WME's input fields into the actual neural net inputs. Next, the forward pass of the mastitis network generates the output unit values which represent the diagnosis of the cow. The network's outputs are copied into the corresponding output fields of the WME, where they replace the NIL values. Finally, INVOKE-NETWORK returns control to the knowledge manager, which deletes the diagnosis subgoal from working memory. It will then examine the result of the diagnosis.

4. The Knowledge Manager.

We have implemented a knowledge manager in ELSIE. The task of the knowledge manager is to operate as an intelligent rule-based dispatcher within the system. One of the subtasks of the knowledge manager is to exploit the occurrence of improperly-classified cows to extend the neural net's training set. Such cows are detected by a set of OPS5 rules that examine the net's diagnosis decisions, contained in the output fields of the cow WME. The rules look like the ALL-LOW-DIAGNOSED rule below, which would fire if the network refused to draw any conclusion about a cow.

```
(p ALL-LOW-DIAGNOSED
   (cow-descriptor ^healthy low ^subclinical low
    ^clinical low ^cow-number <n>)
  -->
   (goal ^type obtain-feedback ^number <n>))
```

For every improper classification, the rule ACQUIRE-RECLASSIFICATION generates a plot of the cow's milk production and cell count history. An expert veterinarian is asked to reclassify the cow based on this graph. A new training example is then generated by taking the input values from the cow descriptor WME and the output values from the vet's response. The expanded training set is used by the knowledge manager for retraining of the neural net, resulting in an incremental expansion of the knowledge captured in the network.

5. Experimental results.

The WARP systolic array supercomputer was used for training and retraining of the mastitis network on a dataset of 116 cows [5]. The data were selected from a cow database maintained by the Department of Herd Health and Reproduction, Utrecht University, the Netherlands. The data had been collected on a controlled farm at 21-day intervals. All cows were standardized on a 378 day lactation period, resulting in 36 input values for the neural network: 18 cell count values and 18 milk production values. For lactation curves spanning a period shorter than 378 days, the missing data values were filled in from an idealized cow model. Missing production data were substituted either by interpolation, or by calculating expected lactation values based on a normal dropoff rate [1]. Missing cell count values were replaced with a standard value of 50,000, representing a typical cell count in a mastitis-free animal. The output units in the training data were set to either 0 or 1 depending upon the expert's classification of the cow. Each output vector had exactly one of the three units turned on, representing either a healthy, a subclinically sick, or a clinically sick cow.

During initial training on the entire dataset, the network classified 87% of the cows identically to our expert veterinarian informant. Analysis of the errors showed that the mastitis net had detected inconsistencies in the

training data that were the result of classification errors by the veterinarian. This was confirmed by showing our informant some of the data again, without telling him why. In several cases he revised his opinion to agree with the net. Training the mastitis net on the corrected dataset resulted in 98% correct performance.

Unfortunately, our original database was too small to provide separate training and test sets large enough to demonstrate good generalization. Mastitis can occur any time during the lactation period; the net needs to see examples at each point in the period in order to recognize novel cases correctly. We are currently expanding the database with additional cows to remedy this problem.

The two curves we are using are only a small part of the data human experts rely on to make accurate diagnoses. Veterinarians typically employ such additional factors as milk temperature and electrophoretic conductivity of the milk. We intend to expand our network to selectively make use of these additional factors. Furthermore, we wish to apply multiple networks to perform other classification tasks on the data, under the control of the knowledge manager. One example is predicting estrus from temperature curves, which is useful for scheduling inseminations. The software we have developed makes it easy to combine multiple networks into one rule-based system.

6. Discussion.

We have presented a novel approach to merging production systems and neural networks. Our solution allows multiple neural networks to serve as inference components of rule-based systems. We have also described a knowledge manager that performs knowledge acquisition by incrementally expanding the training sets of the neural networks under its dominion. This approach is general and can be applied to other systems that need to make implicitly-defined classifications under explicit rule-based control.

The practical uses of our present system are as an aid for herd culling and selection decisions, and for the education of non-expert veterinarians. With further work, we hope to extend the present retrospective analysis to the on-line classification of cows at the moment of milking. Our long-term goal is to develop an inexpensive system for on-line detection of both mastitis and estrus for everyday use on dairy farms.

Acknowledgements.

We thank Arie Brand and David Brée for discussions that contributed to this work. Mirjam Nielen organized and assisted in getting the veterinary reclassifications. Jon Fincham and Gillette Elvgren III helped with the simulations.

References

[1] van Arendonk, J., and Koops, W. (1987) *Applied quantitative analysis in dairy farming*. Course readings published by the Agricultural University of Wageningen, the Netherlands (in Dutch).

[2] Cooper, T., and Wogrin, N. (1988) *Rule-based Programming with OPS5*. San Mateo, CA: Morgan Kaufmann Publishers.

[3] Grommers, F. J. (1988) Host resistance machanisms of the bovine mammary gland: An analysis and discussion. *Neth. Milk Dairy J.*, **42**:43-56.

[4] McDermott, J. (1982) R1: A rule-based configurer of computer systems. *Artificial Intelligence*, **19**:39-88.

[5] Pomerleau, D. A., Gusciora, G. L., Touretzky, D. S., and Kung, H. T. (1988) Simulating neural networks at Warp speed: How we got 17 million connections per second. *Proceedings of the IEEE International Conference on Neural Networks*, volume 2, pp. 143-150. San Diego, CA.

[6] Schukken, Y. N., Erb, H. N., Sears, P. M., and Smith, R. D. (1988) Ecologic study of risk factors for environmental mastitis. *Am. J. Vet. Res.*, pp. 766-169.

NEURAL NETWORKS AS FORECASTING EXPERTS: AN EMPIRICAL TEST

Ramesh Sharda
College of Business Administration
Oklahoma State University
Stillwater, Oklahoma 74078
email: MGMTRSH@OSUCC.BITNET

Rajendra B. Patil
Dept. of Computer Science
Oklahoma State University
Stillwater, Oklahoma 74078
email: patil@a.cs.okstate.edu

ABSTRACT

Among the various potential applications of neural networks, forecasting is considered to be a major application. Several researchers have reported their experiences with the use of neural networks in forecasting, and the evidence is inclusive. This paper presents the results of a forecasting competition between a neural network model and a Box-Jenkins forecasting expert system. Seventy five series, a subset of data series which have been used for comparison of various forecasting techniques, were analyzed using the Box-Jenkins approach and a neural network implementation. The result show a dead heat between the two approaches. The simple neural net model tested on this set of time series could forecast about as well as the Box-Jenkins forecasting system.

I. INTRODUCTION

Forecasting has been mentioned as one of the most promising applications of artificial neural networks. The autoassociative memory of certain neural network models can be tapped in prediction problems. Smolensky(1986) specifies a dynamic feed-forward network in the following way:

$$u_i(t+1) = F[\Sigma_k W_{ki} G(u_k(t))]$$

where $u_i(t)$ is the activation of unit i at time t, F is a nonlinear sigmoid transfer function, G is a nonlinear threshold function and W_{ki} is the connection strength or weight from unit k to unit i. This relationship can, in principle at least, be used for predicting future values of variables.

Several authors have attempted to apply this idea for forecasting a time series. Werbos(1988) states that he laid the foundations for use of backpropagation in forecasting in his doctoral dissertation, Werbos(1974). We have not yet seen the 1974 reference. In his 1988 article, he describes an application of backpropagation to locate sources of forecast uncertainty in a recurrent gas market model.

Lapedes and Farber(1987) used a multilayered perceptron to predict values of a nonlinear dynamic system with chaotic behavior. They reported that neural network gave superior prediction for their dynamical system. Fozzard et al(1989) discuss a neural nets based expert system for solar flare forecasting, and claim that its performance is superior to human experts.

However, the experiences with neural networks in forecasting are not all positive. Fishwick(1989), for example, reports that the forecasting ability of neural networks was inferior to simple linear regression and surface response methods. There are some trade magazine articles about use of neural networks in stock price forecasting, but no concrete writeup can be found (perhaps for confidentiality reasons). Even when the use of neural networks in forecasting has been shown to be positive, it is usually based on test data sets from a particular problem domain.

This paper reports results of a forecasting competition between a neural network model and a traditional forecasting technique, Box-Jenkins forecasting. Several data series from a comprehensive forecasting competition were analyzed using a neural network model and the Box-Jenkins time series forecasting techniques. The data series came from a variety of sources. The results show that the performance of neural networks was on par with the Box-Jenkins modeling technique.

II. DATA, MODELS AND METHODS

DATA

The time series were selected from the famous M-Competition (Makridakis et al 1982) to compare the performance of various forecasting techniques. Out of 1001 series collected, only 111 series were analyzed in M-Competition using Box-Jenkins methodology. This was done because the Box-Jenkins approach requires an analyst's intervention and is thus quite time consuming. Pack and Downing (1983) examined this 111 series subset and concluded that several series were not appropriate for forecasting using the Box-Jenkins technique. We took our sample of 75 series from this 111 series set after considering Pack and Downing's recommendations. Our

test set contains 8 annual, 18 quarterly and 49 monthly series. In Table 1, series numbers ≤ 112 are annual, series numbers ≤ 382 and > 112 are quarterly, and the rest are monthly.

MODELS

Box-Jenkins Method:

This approach to time series forecasting is well known(Box & Jenkins 1976) and has been applied in practice. It is considered to be a 'sophisticated' approach to forecasting, but is quite complex to use. Essentially the analyst examines both the auto and partial autocorrelations and identifies models of the form

$$\phi(B)\Phi(B^s)\nabla^d\nabla^D_s\ (Z_t-c) = \theta(B)\Theta(B^s)a_t$$

where B is the back-shift operator (i.e $Bx_t = x_{t-1}$)
∇ = I-B; s=seasonality, a_t=white noise;
$\phi(B)$ and $\Phi(B^s)$ are nonseasonal and seasonal autoregressive polynomials respectively;
$\theta(B)$ and $\Theta(B^s)$ are nonseasonal and seasonal moving average polynomials respectively;
Z_t = series (transformed if necessary) to be modeled.

After identification of several candidate models, the analyst can iterate through the process of estimation and diagnostic-checking. Once the final model has been selected, the forecasting process can begin.

The process of model identification, estimation and diagnostics-checking has been automated and is available in the form of a forecasting expert system. The performance of such an automatic 'expert' system has been reported to be comparable to real experts (Sharda &Ireland 1987). For our tests, we used such an automatic Box-Jenkins modeling expert system, AUTOBOX (AFS Inc, 1988). This program can take a dataset and iterate through the model identification, estimation and diagnostics process to develop the best model.

Neural Network Model:

A simplest form of backpropagation rule was used to train a three layer perceptron network (with one hidden layer). A commercially available neural network simulator, BrainMaker was used. It uses a version of delta rule used by Sejnowski and Rosenberg (1987) in their NETtalk application which is:

$$\Delta_p W_{ij} = \varepsilon\ ((1-\mu)\ \delta_{pi}\ O_{pj} + \mu\Delta_{p-1}W_{ij}\ )$$

where $\Delta_p W_{ij}$ is change in weight W_{ij} due to pattern p, T_{pi} and O_{pi} are target and output neuron values, TF is a transfer function and A_{pi} is activation level of neuron i, p is one of the N facts used for training, ε is the learning rate, μ is the smoothing factor:

$$\delta_{pi} = -(\delta E_{pi}/\delta O_{pi})\ TF'(A_{pi})$$

and $E_{pi} \equiv \frac{1}{2}(T_{pi} - O_{pi})^2$ is the squared error in the actual and desired output of the network.

$$E = \frac{1}{2}\ \Sigma_p \Sigma_i\ (T_{pi} - O_{pi})^2$$

Throughout the analysis we had only one neuron in the output layer as only single step ahead forecasting is tested. The algorithm used is not a true gradient descent in the error E where the weights are changed after each pattern presentation (Rumelhart, Hinton and Williams 1986).

METHOD

Seventy five data sets were analyzed using the following approach. For each data set, *n-k* observations were used to build the forecast model (to train the network), and then the model (the trained neural network) was used to forecast the future *k* values, where *k*=6, 8, 18 for annual, quarterly and monthly series respectively. These values are well established for such comparisons in the forecasting literature. The forecast were generated only one step ahead. That is, at the end of period *n-k*, *n-k+1* value was forecast; at the end of *n-k+1* period, *n-k+2* value was forecast and so on. The generated forecast were compared with the actual values for the k periods, and mean absolute percent error (MAPE) was computed for each series as:

$$MAPE_i = \left(\sum_{j=1}^{k} |A_{ij} - P_{ij}|*100\ \right) / k$$

The series were run in AUTOBOX using its default setting with no intervention detection. For the neural network model, the training set was constructed in the following way. Each record consisted of two years of history and the target (to be forecast) value. Thus the annual series model consisted of 2 input neurons, quarterly series were trained using 8 input neurons and monthly using 24 input neurons. The number of neurons in the hidden layer was the same as in the input layer. The output neuron was always one, since only one step ahead forecast were desired in this experiment. The training tolerance was set to 0.08 and learning rate was set to 1.

III. RESULTS AND CONCLUSIONS

Table 1 exhibits the MAPE's of AUTOBOX and the neural network approach. It shows that the simple (training algorithm) neural network approach performed as well as a forecast expert system.

The mean of the MAPE's for neural nets model is slightly less than that for the Box-Jenkins modeling system. However, due to a large standard deviation, the difference is insignificant. A pairwise means test also indicates the same result. Forecasts using AUTOBOX resulted in lower MAPE's for 36 series, and thus the neural network model was able to do better in the other 39 series.

When the series are grouped on the basis of periodicity, the MAPE's are still insignificantly different between the two approaches. This suggests that the periodicity of the series being modeled does not affect a technique's performance. It was quite interesting, atleast for us, that the neural network model was able to incorporate seasonality automatically, just as AUTOBOX is able to do.

Chart 1 shows a graphical comparison of MAPE's for two approaches, this chart indicates that the pattern of MAPE's for both approaches is quite similar.

These results are quite encouraging for the proponents of neural networks as a forecasting tool. Obviously this work needs to be replicated to assess the full potentiality of neural networks for forecasting. Possible use of other neural network architectures and more sophisticated training algorithm may improve the results. Repeating the same experiment with n-step ahead forecast remains as a future work. We also need to examine the connection weights to compare the underlying models developed by AUTOBOX and neural network models.

IV. REFERENCES

AUTOBOX, *Software User Manual,* Automatic Forecasting Systems, Hatboro, PA, 1988.

Box, G. E. P. and G. M. Jenkins, "Time Series Analysis: Forecasting and Control", Holden-Day, San Francisco, CA. 1976.

Brainmaker, *Users Guide and Reference Manual,* California Scientific Software, Sierra Mandre, CA 1988.

Fishwick, P., "Neural Network Models in Simulation: A Comparison with Traditional Modeling Approaches", *Working Paper,* Department of Computer and Information Science, University of Florida, Gainesville, FL, 1989.

Fozzard, R., Bradshaw, G. & Ceci, L., "A Connectionist Expert System for Solar Flare Forecasting", *in Advances in neural information processing system I* / Edited by David S. Touretzky, Morgan Kaufmann Publishers, Inc. pp. 264-271, 1989.

Lapedes, A. & Farber, R., " Nonlinear Signal Processing Using Neural Networks: Prediction and System Modeling", *Los Alamos National Lab Technical Report* LA-UR-87-2261, July 1987.

Makridakis, S., et al, " The Accuracy of Extrapolation (time series) Methods: Results of a Forecasting competition", *Journal of forecasting,* Vol. 1, pp. 111-153, 1982.

Pack, D. J. and D. J. Dowining, "Why Didn't Box-Jenkins Win (Again)?", *Proceedings, Third International Symposium on Forecasting,* Philadelphia, 1983.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J., "Learning Internal Representation by Error Propagation", *Parallel Distributed Processing,* Vol. 1, pp. 318-62. Cambridge, MA: MIT Press, 1986.

Sejnowski, T. J., and Rosenberg, C. R., "Parallel networks, that learn to pronounce english text", *Complex Systems* 1:145-68. 1987.

Sharda, R. and T. Ireland., " An Empirical Test of Automatic Forecasting Systems", *ORSA/TIMS Meeting,* New Orleans, May 1987.

Smolensky, P., "Neural and Conceptual Interpretation of PDP Models", *in Parallel Distributed Processing,* Vol. 2, J. L. McClelland and D. L. Rumelhart and the PDP Research Group, Editors, MIT Press, Cambridge, MA, pp 397, 1986.

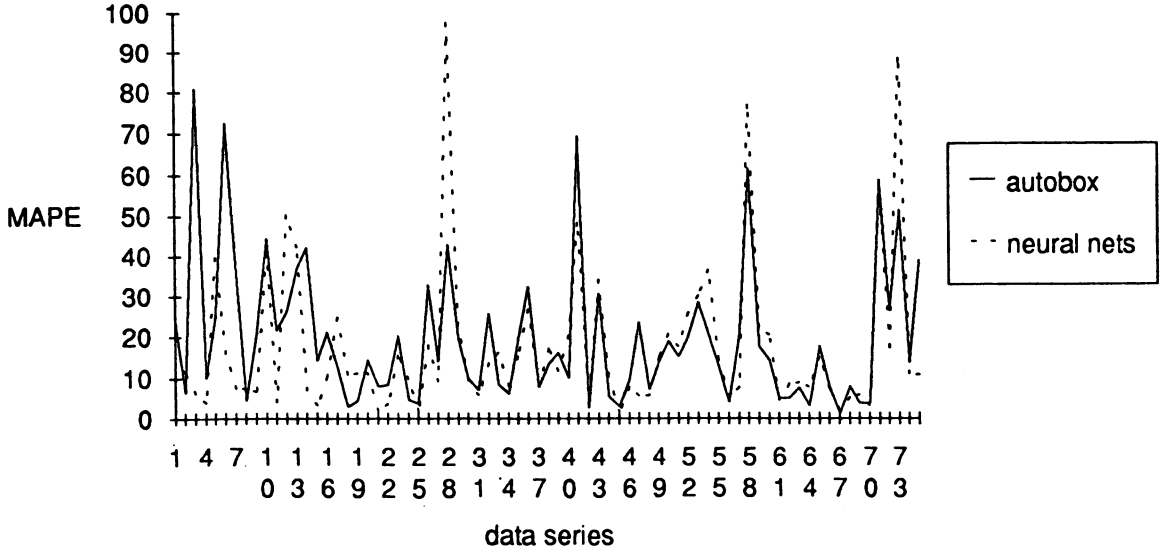
Werbos, P., "Generalization of Backpropagation with Application to Recurrent Gas Market Model", *Neural Networks,* Vol. 1, pp. 339-356, 1988.

Werbos, P., "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences", *Ph.D. Thesis,* Harvard University, 1974.

Table 1. Mean Absolute Percent Error in AUTOBOX and Neural Network

series #	autobox MAPE	neural net MAPE	series #	autobox MAPE	neural net MAPE	series #	autobox MAPE	neural net MAPE
4	23.53199	14.20809	382	32.77625	17.946	652	15.22743	17.52494
13	6.575614	11.51651	400	14.14283	9.147532	661	19.79772	25.85414
31	81.0658	6.276065	409	42.79787	97.43216	670	28.38675	30.25353
40	10.16478	3.846814	418	19.23414	22.73318	679	20.41933	36.5198
49	24.76677	39.85942	427	9.891018	9.315865	688	12.86458	14.30556
58	72.57376	16.1733	436	7.091606	5.853671	697	4.132447	5.241174
85	37.00226	7.533853	445	25.70834	13.53341	706	20.11109	7.86965
112	4.55244	7.533853	454	8.475374	16.16669	715	61.3126	76.65439
184	20.61459	6.668924	463	6.186729	8.034003	724	17.43141	22.26727
193	44.42916	40.79303	472	19.20641	15.28932	733	14.20924	20.34063
202	21.51311	4.081104	481	32.35556	26.7007	742	4.99641	4.510452
211	26.31219	50.35743	490	7.904338	8.114671	751	5.189672	8.625431
220	37.10015	42.24019	499	13.50357	17.14914	769	7.748496	8.867242
229	42.19236	7.109735	508	16.11352	11.52449	787	3.169144	7.897152
238	14.22043	3.233037	526	9.946912	20.28435	796	17.68344	15.16847
265	21.14956	10.50463	535	69.14082	48.262	805	7.930251	7.34254
292	12.67486	24.85503	544	2.720653	3.109809	823	1.290048	2.038724
301	2.913655	10.77437	562	30.62689	34.09323	832	7.964891	5.262468
310	4.462999	11.30166	571	5.568459	10.39477	877	3.726682	5.949852
319	14.30099	11.20045	580	3.027975	1.337604	904	3.878792	3.165669
328	8.01775	1.873045	589	9.658729	7.767564	913	58.52926	57.16597
337	8.522994	3.684558	598	23.48701	5.624283	922	26.35101	17.23289
346	20.22541	15.64341	616	7.216475	5.725161	949	51.11555	88.25377
355	4.750817	9.74922	634	13.70159	15.23605	958	13.66954	10.84574
364	3.734731	1.880796	643	18.92017	20.7128	967	38.74342	10.60136
						average	19.76869	17.65528
						std dev	17.51539	18.85821

Comparison of AUTOBOX and Neural Network



Artificial Neural Networks for Multiple Criteria Decision Making

Jun Wang & B. Malakooti
Department of Systems Engineering
Center for Automation & Intelligent Systems Research
Case Western Reserve University
Cleveland, Ohio 44106

1. Introduction

A majority of the applications of artificial neural networks (ANNs) to real-world problems are concentrated on emulating living activities of biological systems such as perception and memory. This paper is an attempt to apply ANNs to one of the most sophisticated processes: multiple criteria decision making (MCDM) under certainty.

A general problem of MCDM under certainty can be formalized as

$$DR: \max_{d \in \mathbf{D}} c = F(d) \tag{1}$$

where DR stands for Decision Rule, $d = (d_1, d_2, \ldots, d_p)^T$ is a p-tuple decision variable, $\mathbf{D}$ is a set of available decisions; $c = (c_1, c_2, \ldots, c_q)^T$ is a q-tuple vector of criteria (objectives) of the the decision maker's (DM's) concerns, $\mathbf{C}$ is a set of possible consequences. The above formalization of the problem can be interpreted as deriving a DR according to the DM's preference and rationality principle(s), then applying the DR to determine an optimal decision d^* in $\mathbf{D}$.

In the process of MCDM, because of the non-commensurate multiple objectives and the large set of alternatives, it is often desirable to obtain complete knowledge on the DM's global preference structure explicitly expressed by a prescriptive model such as a multiattribute value function (MAVF). Once the DM's underlying preference structure is assessed, then a rational decision rule can be derived, and the application of the decision rule is reduced to a conventional optimization problem.

During the past three decades, great efforts were made by decision theorists and practitioners from a variety of disciplines in developing theory and methodology for assessment of the DM's underlying preference structure, especially in the form of MAVF. In the literature, the common practice for assessment of a DM's MAVF is decomposing the MAVF into different functional forms such as additive, multiplicative or multilinear representations under some independent conditions. Under a particular decomposed form, marginal utility functions and scaling constants then are assessed, and the MAVF is aggregated [1].

Numerous studies indicate that the existing approaches have some inherent disadvantages. The decomposition approaches theoretically require that the problems satisfy one of the independence assumptions to be decomposable, which restricts the applicability of the approaches. Furthermore, a limitation of the existing approaches is the requirement of predetermining functional forms of the MAVFs which are usually assumed to be analytical. This excludes the possibility of logic-style or rule-based descriptors which seem common in practice and in expert systems. From a practical point of view, the approaches impose substantial cognitive burden on DMs in verifying the independence assumptions and also are subject to response biases in the sense of lacking a formal mechanism to minimize errors. Since in general a human decision maker's preference behavior is a complex phenomenon and determination of a functional structure of formal representation for general preferences is usually difficult if not impossible, assessment of a DM's underlying MAVF using existing methods is not an easy task. This reduces the value of multiattribute utility theory which, otherwise, is very effective among existing MCDM theories and methods.

In this paper, we present a connectionist approach to inference of decision rule in a general setting of MCDM under certainty. The objective of the development is to capture the essence of the DM's rational preference with ANN representation. Specifically, our motivation is to construct ANN-based rational preference models (RPMs) from available information through supervised learning which follows given rationality principle(s) and resembles the DM's underlying preference structure.

2. Model Configuration

The primary interest of the research is to represent the DM's preference structures with connectionist RPMs. A RPM of a DM is a prescriptive model which should (i) satisfy a set of specified common axioms of rationality; and (ii) pertain to the DM's individual value assessments for the outcomes and reflects the

DM's implicit tradeoff among the attributes. Since ANNs possess very robust generalization power on a variety of mappings including algebraic and logic functions, they may represent a general form of RPMs.

2.1 Formulation of the Decision Processes: Let's consider an ANN as a mechanism of many-to-one mapping from attribute space or Cartesian product of attribute space to a real line, viz.

$$RPM : \times_{i \in I} \mathbf{C} \rightarrow \mathbf{Z} \iff ANN : \mathbf{X} \times \mathbf{W} \rightarrow \mathbf{Y}$$

where $\mathbf{Z}$ is the set of output of a RPM; $\mathbf{Y}$, $\mathbf{X}$, and $\mathbf{W}$ are the set of output, inputs, and adaptable parameters of an ANN respectively; I is a prespecified index set.

The fundamental assumptions of the development on the MCDM problems are: (i) The DM is able to articulate with adequate discriminatory power, in a format, his or her preference over the power set of alternatives according to the his or her underlying preference structure and knowledge about the decision problem; (ii) the decision situation is fully characterized, i.e. the sets of decision variables $\mathbf{D}$, and the objective functions $F(d)$ are known. Due to the goal seeking nature of prescriptive models, it is necessary to specify a set of axioms of rationality for further development. The basic axiom of rationality on the decision maker's preference relation is Pareto principle which can be characterized on the binary preference relation, i.e.

$$(c' \geq c'') \Rightarrow (c' \succ c'') \vee (c' \sim c''), \tag{2}$$

where $\succ$ denotes *preference* and $\sim$ denotes *indifference*. An alternative c^* is Pareto optimal if $\exists i, \forall c \neq c^*, c_i^* > c_i$.

In this context, the whole decision making process can be decomposed into two sequential phases: learning (system modeling) phase and ranking (decision making) phase. The learning phase consists of three stages: preprocessing stage, training stage, and testing stage. The preprocessing stage begins with determining the representative alternatives as the basis of the inductive inference. The minimum requirement for these alternatives is Pareto optimality (nondomination), since dominated alternatives are less informative and the preference information of dominated alternatives can usually be deduced from that of nondominated ones. The preference relations of particular type (e.g. holistic ratings or paired comparisons) on the selected alternatives then are extracted by presenting the outcomes of these alternatives in some format to the DM for evaluations. The results of the DM's evaluation then are checked according to the predefined rationality principle to eliminate irrationality. Based on properties of the specific RPMs, additional data may be generated from that preference information. The selected alternatives with associated preference relations constitute an information database. The data in the information database are divided into two sets corresponding to their usage in the two subsequent stages: training set S_{trn} and testing set S_{tst}. In the training stage, the instances in the training set are presented to the ANN. Based on a prespecified performance criterion, the ANN modifies its adaptable parameters according to a tuning scheme to learn the DM's rational preference behavior represented by the training instances. Figure 1 illustrates the dynamics of the learning process in the training stage. In the testing stage, tests are performed with the data in the testing set. It is important to test the performance of the inductive inference using data different from those used in the training stage. There is the possibility of retraining the net if the results of tests are unsatisfactory. The purpose of the learning processes essentially is to infer a general robust decision rule which represents the DM's rational preference from a limited number of input-output training instances; more precisely, to discover the DM's rational preference structure based on available preferential information and rationality principle. As prescriptive models, the rationality of the decision rule is ensured by (i) proper selection of training instances; (ii) proper guidance of learning processes; (iii) proper testing of the synthesized models, i.e. the rationality principle must be part of the testing criterion for validation. In the ranking phase, the synthesized ANN-based RPM can act as a rational proxy for the DM to evaluate or rank any given alternatives.

2.2 Constructive Specification of RPMs: The most popular RPM for deterministic MCDM problems is MAVF. A MAVF is a map from attribute set to value set, that is, $M_1 : \mathbf{C} \rightarrow \mathbf{R}$ such that $c' \succ c'' \Leftrightarrow M_1(c') > M_1(c'')$ and $c' \sim c'' \Leftrightarrow M_1(c') = M_1(c'')$ where the input of the model is c; the output of the model is v, assessed value corresponding to the outcome z. In order for the configured M_1 to behave rationally, according to Pareto principle and the model definition, $M_1(c)$ must be monotone nondecreasing with respect to c. The DR for $M_1(c)$ is $\max_{d \in \mathbf{D}} M_1[F(d)]$, i.e. select $d^* \in \mathbf{D}$ such that $\forall c = F(d)$, $M_1(c^*) \geq M_1(c)$.

Since $\succ$, $\prec$, and $\sim$ constitute a partition in $\mathbf{C} \times \mathbf{C}$, another type of *RPM* is Comparison *RPM* (CRPM). A CRPM is a mapping from the Cartesian product of attribute space corresponding to any pair of alternatives to a set of trinary values, e.g. $M_2 : \mathbf{C} \times \mathbf{C} \rightarrow \{0.25, 0.5, 0.75\}$. More specifically, the input is $x^T = (c'_1, \ldots, c'_q, c''_1, \ldots, c''_q)$, an augmented 2q-tuple vector; the outputs z are given based on the results of paired comparisons, a preference relation in $\mathbf{C} \times \mathbf{C}$.

$$z \triangleq \begin{cases} 0.75, & \text{if } c' \succ c''; \\ 0.5, & \text{if } c' \sim c''; \\ 0.25, & \text{if } c' \prec c''. \end{cases} \tag{3}$$

By definition, the *CRPM* has the following properties.

$$\forall c', c'' \in \mathbf{C}, \qquad M_2(c', c'') = 1 - M_2(c'', c'), \quad \& \quad M_2(c', c') = M_2(c'', c'') = 0.5. \tag{4}$$

According to Pareto principle and the constructive specification, $M_2(c', c'')$ is monotone nondecreasing with respect to c' and nonincreasing with respect to c''. The *DR* for M_2 is selecting $d^* \in \mathbf{D}$ such that $\forall c = F(d)$, $M_2(c^*, c) \geq 0.5$.

The output of M_1 is an interval scale variable whereas the output of M_2 is a an ordinal scale variable.

2.3 Functional Behavior of ANNs: Based on the theoretical results of our previous research work [2], the architecture of the *ANNs* used in assessing *RPMs* is shown in Figure 2.

The incorporation of the information in addition to that given by training instances into the supervised learning, or the utilization of prior knowledge of the trainer, is an important aspect of the learning processes, which seems neglected in *ANN* literature. One way of doing this is to ensure that the *ANNs* possess the functional properties of the underlying processes such as monotonicity that the configured *RPMs* possess.

Theorem: *Let each neuron activation function be a sigmoid function, i.e.* $f_k(u_k) = (1 + e^{-\lambda_k u_k})^{-1}$ *where* u_k *is an instrumental variable standing for net inputs to neuron* k*, for output neuron* $u_1 = \sum_{l=2}^{N} w'_{1l} a_l + \sum_{l=1}^{n} w''_{1l} x_l + w''_1$ *and for hidden neurons* $u_k = \sum_{l=1}^{n} w''_{kl} x_l + w'''_k$*; N is the total number of neurons, n is the number of input components,* a_l *is the state variable of the* l^{th} *neuron;* w'_{1l}, w''_{kl}*, and* w'''_k *denote the weight from neuron l to output neuron 1, from input l to neuron k, and threshold of neuron k respectively. The output y of the ANN is monotone nondecreasing coordinatewise with respect to input* x_j*,*
(i) if the parameters satisfy the following inequality

$$\sum_{k=2}^{N} g_k w'_{1k} w''_{kj} + w''_{1j} \geq 0 \tag{5}$$

where $\forall k, g_k = \begin{cases} 0, & w'_{1k} w''_{kj} > 0 \\ 0.25\lambda_k, & w'_{1k} w''_{kj} < 0 \end{cases}$*, vice versa for nonincreasing ANNs; or*

(ii) iff (5) holds where $\forall k, g_k = \lambda_k f_k(\hat{u}_k)[1 - f_k(\hat{u}_k)]$, $\hat{u}_k = \sum_{l=1}^{n} w''_{kl} \hat{x}_l + w'''_k$, *and* $\forall l, \hat{x}_l = \begin{cases} x^*_{\max}, & w'_{1k} w''_{kj} > 0 \\ x^*_{\min}, & w'_{1k} w''_{kj} < 0 \end{cases}$*,* $x^*_{\max}$ *and* $x^*_{\min}$ *are the optimal solutions to* $\max_{x \in \mathbf{X}} (\sum_{l=1}^{n} w''_{kl} x_l + w'''_k)^2$ *and* $\min_{x \in \mathbf{X}} (\sum_{l=1}^{n} w''_{kl} x_l + w'''_k)^2$ *respectively, vice versa for nonincreasing ANNs.*
Proof: Omitted for lack of space.

These constraints characterize a feasible region in the parameter space $\mathbf{W}$. Because of the high nonlinearity it is obviously difficult to incorporate the complicated constraints into learning paradigms. Nevertheless, these constraints can be used as an additional testing criterion for validation of the synthesized *ANNs*.

3. Computer Simulation

Simulations were performed using The *Adaptive Delta Rule* [2] which is based on *conjugate direction* and *golden section search* method to assess MAVFs, M_1, and CRPMs, M_2. Three different types of value functions, a *Chebychev*, an *additive* and a *quadratic* function, were assumed to act implicitly as the DM's underlying rational preference in providing responses of preferential information, i.e.

$$v_1(c_1, c_2, c_3) = \min\{c_1, c_2, c_3\}, \quad v_2(c_1, c_2, c_3) = 0.5c_1 + 0.3c_2 + 0.2c_3,$$

$$v_3(c_1, c_2, c_3) = -0.5c_1^2 - 0.3c_2^2 - 0.2c_3^2 + c_1 + 0.6c_2 + 0.4c_3.$$

Fifty (50) non-dominated (Pareto optimal) alternatives of 3 attributes were generated *randomly* based on uniform distribution over the interval [0, 1]. The alternatives are divided in half arbitrarily into training and testing sets. In assessing M_1, 25 training and 25 testing instances were used. In assessing M_2, to make full use of the preference information, property (4) was used to generate additional training data, i.e. 20 paired comparisons, 40 additional training and testing pairs of data were deduced. For simplicity, in the simulation we also assume the DM is confident about his or her value assessments about all outcomes. Therefore, according to the propositions in [2], the least square error functions was used, and the total number of neurons (including one output neuron) N is 30 for M_1 and 50 for M_2 respectively. The number of iterations is 50,000.

The results of the simulation show that the synthesized ANNs possess a remarkable capability of learning from examples. The synthesized ANN-based RPMs can determine the optimal alternatives *correctly*, and generate the output values very close to the expected values in both *training* and *testing* sets. In addition, the synthesized ANNs also preserve the monotonicity of the RPMs checked with the above Theorem. The following table shows the percentage of *correct* ranking of all alternatives and mean square error (MSE= $\sum_{p=1}^{P}(y-z)^2/P$) of outputs in assessing MAVFs and CRPMs based on *Chebychev*, *additive*, and *quadratic* function respectively.

%/MSE	Chebychev	Additive	Quadratic
M_1 : S_{trn}	$88/5.35 \times 10^{-7}$	$100/6.52 \times 10^{-7}$	$100/1.50 \times 10^{-6}$
M_1 : S_{tst}	$76/8.42 \times 10^{-4}$	$100/3.47 \times 10^{-6}$	$96/3.25 \times 10^{-5}$
M_2 : S_{trn}	$100/4.80 \times 10^{-7}$	$100/3.17 \times 10^{-7}$	$100/6.20 \times 10^{-7}$
M_2 : S_{tst}	$88.3/5.50 \times 10^{-3}$	$93.3/6.74 \times 10^{-3}$	$95.0/1.67 \times 10^{-2}$

4. Conclusion

In this paper, we have demonstrated the applicability of ANNs to assess the DM's rational preference structure. We found that the synthesized ANN-based RPMs are very robust, and represent generalizations of examples. The main advantages of the connectionist RPMs are that they are functional form independent and internal parameter insensitive. The proposed approach could also be extended to model intelligent decision support systems and advanced connectionist expert systems.

References

[1] Chankong, V. & Y. Y. Haimes, *Multiobjective Decision Making: Theory and Methodology*, North-Holland (1983).

[2] Wang, J. & B. Malakooti, "On Training of Artificial Neural Networks," *Proceedings of IEEE/INNS International Joint Conference on Neural Networks*, Vol. II, 387-393, Washington, DC (1989).

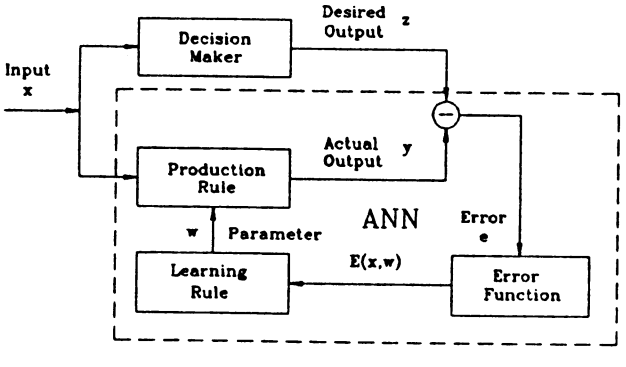


Figure 1: The Dynamics of the Learning Process.

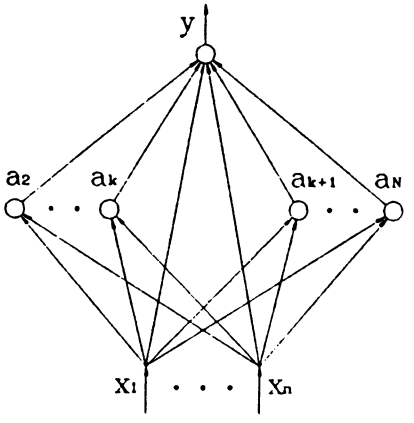


Figure 2: The Architecture of the ANNs.

AN OVERVIEW OF WEIGHTLESS NEURAL NETS

I. Aleksander [1] and H. B.Morton[2]

[1] Professor of Neural Systems Engineering: Imperial College, University of London, London SW7 2BT UK.

[2]Lecturer in Human Sciences, Brunel University, Uxbridge UB8 3PH, UK

INTRODUCTION

Weightless neural systems are distinguished from those classically used in neural computing by the fact that they store required responses to their input patterns in addressable locations rather than as combinations of connection strengths. This line of research has been pursued by the authors (IA) for some time, and it is the aim of this paper to review the way in which the idea lines up with what has now become the classical domain of weight-based systems. In fact, it will be shown here in a non-rigorous way, that weightless systems cover the general neural network paradigm. The reason for building neural nets the weightless way lies both in ease of implementation and in new insights on neural computing.

DEFINITIONS AND TERMS

A RAM node is the most basic type of element for the design of weightless neural systems. It receives N binary inputs at the address terminals (X_1 to X_N)and produces one binary output at the "data-out" terminal (F_j for the jth neuron in a net). Training is achieved by supplying the desired output values of 0 or 1 at the "data-in" terminals for given input patterns, while the writing mechanism of the RAM is energised ("write-enabled" in computer designer's jargon). Logical neural nets have well defined *learning phases* when the elements of the net are write-enabled and changes occur in the stored content of the RAM. In contrast, there are *running phases* during which the writing mechanisms are disabled and the net performs computational tasks based on previously learnt material. A conventional RAM which stores M bits per word is viewed as being M RAM neurons with their inputs receiving the same data, while each of the M outputs can learn independent responses to this data.

Logical neural nets may be run either **synchronously** with all nodes making a new attempt at firing at the simultaneous arrival of a clock pulse at all nodes, or **asynchronously** with nodes firing at random, but making a known average number, W, of attempts per unit time. The latter is precisely the arrangement assumed by Hopfield (1982) and by Hinton et. al in Boltzmann machines (1984).

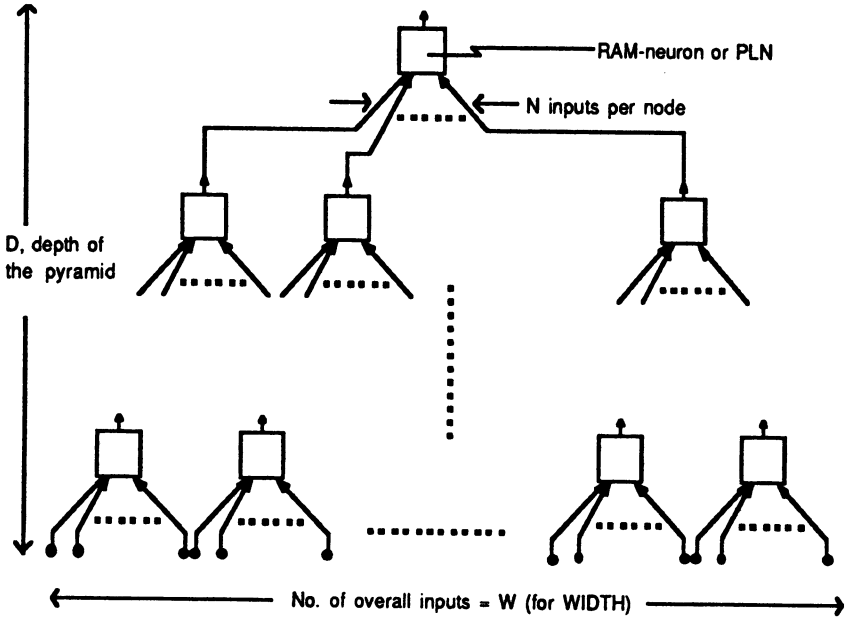


Figure 1: A pyramid.

A **Probabilistic Logic Node** (PLN) differs from the RAM neuron in the sense that a q-bit number is now stored at the addressed location in a RAM. The content of this location is turned into the probability of firing (i.e. generating a 1) at the overall output of the node. Say that q is 3, then the numbers 0 to 7 can be stored in each location of the RAM. One way of treating the actual number stored may be as a direct representation of the firing probability by treating the number as a fraction of 7. So a stored 2 would cause the output to fire with a probability of 2/7, and so on.

A **pyramid** is a network of cells as shown in fig.1. We note a simple relationship betweeen the Width W, depth D and number of inputs per cell, N:

$$W = N^D \quad \text{(N and D are integers)}$$

Therefore, for a given input width (or given depth) the pyramid is seen to be canonical, in the sense that its definition depends only on N. Armed with these definitions we shall now look at examples of the behaviour of some canonical networks.

PARADIGM COVERAGE 1: PERCEPTRONS

The one-layer network shown in fig. 2, is called a **discriminator** . This may be compared to a single perceptron unit in a single-layer net. Each RAMs encompasses an A-unit and a weight in the perceptron. The discriminator starts by having all RAMs set to 0 and has 1s fed to all data-in terminals for all patterns in the training set. The response r of the discriminator, to an unknown pattern U is given approximately by: $(A<max>)^N$, where A<max> (in the range 0 to 1) is the proportion of overlap between that pattern in the training set which is closest in Hamming distance to U and U itself. Details may be found in Aleksander and Morton (1989). It should be noted in passing that it is this scheme which was incorporated in the WISARD neural image recognition machine which, in 1984 , became one of the first commercially developed neural systems to be used in industrial settings. It recognises 512x512 bit images at the rate of 25 per second.

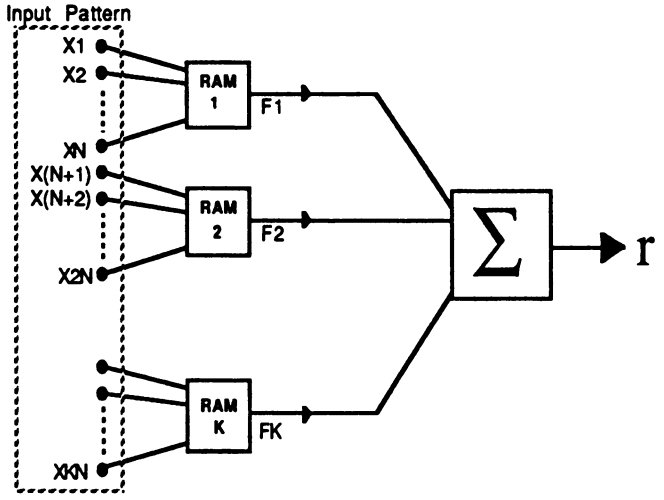


Figure 2: A Discriminator

PARADIGM COVERAGE 2: CONTENT ADDRESSABLE SYSTEMS

a) Discriminator Based.

The Hopfield/Boltzmann axis of the paradigm calls for large neurons, that is, neurons with I inputs for I-element nets. Discriminators, as described above, may be thought of as being such neurons. Their inputs are connected to the outputs of each discriminator via an overall threshold on the responses r of the discriminator. Attractors are created by training the system to output the chosen attractor pattern with the same pattern at the input. It may be shown that the memory capacity in terms of the number of I-bit stored vectors is a variable which goes from 2 with N=1 to 2^I with N=I. In the latter case we have a universally variable finite-state machine with 2^I states. The extreme cases are not of much use, but the ability to hit targets in between has an advantage over the Hopfield/Boltzmann methodology.

b) Fully Connected PLN nets

Such nets have each PLN connected to all the others in the net. Consider PLNs that store only 3 messages: 0, 1 and 0.5. Through storing 0.5, the neuron has the ability to "say" "I don't know". Therefore, before any training takes place, all the nodes in a net are set to "0.5". The initial Makov chain for such systems is easily imagined: equal transition probabilities between all states. An important question is whether the training of a fully connected PLN net , through the creation of stable states, makes those states attractors of transitions from similar states. In the general case, consider an N-element net. Say that it is trained to make the all-0 pattern stable. Now consider what happens if the system is started in a state which has all 0s except one. Say that the kth neuron is in the 1 state.

In the synchronous case, all the neurons except the kth are addressed by a string of 0s with a single 1 in their midst. This finds a "0.5" stored at this majority of addresses and a 0 at the kth neuron. The next state will then have a 0 at the kth neuron and arbitrary values at all the other neurons. This means that about half the neurons will output a 1. So the next state will be even less like the trained state than the state with only one 1. This means that the training has not created an attractor and transitions to the trained state are purely accidental. In the asynchronous case, only one neuron output can change at any one time. The addressing is the same as above so that when a neuron attempts to fire, all except the kth neuron will output a 0 with 50% probability. This means that half of the states with only one disruption will enter the trained state. So, even if the trained state is not a perfect attractor, it is much more successful than the synchronous case.

It has been shown in Aleksander (1989) that the way to overcome the problem in the synchronous case is to "clamp" the desired output while the feedback connections are subjected to small amounts of noise. Here it is worth pointing out that the problem is less severe for nets that are not fully connected because a single disruption finds fewer neurons with a "0.5". The major difficulty with fully connected PLN networks is that their memory cost escalates exponentially with size of net. For an N+1 node net, 2^N storage locations are required in each . In the next case considered in this paper, it will be shown that a very similar performance may be obtained through the use of PLN pyramids which implies a lower cost . This also covers the third leg of the paradigm: error learning.

PARADIGM COVERAGE 3: ERROR LEARNING

In weightless systems the tasks done by error back-propagation can be achieved by feeding error information to all the hidden layers of a feed-forward system in parallel. Consider the structure of fig. 1 with each element a PLN. We also start by assuming that the PLNs are of the three-message variety, storing 0, 1, or "0.5". The key fact about a pyramid is that it inherently has hidden units. This means that an error is known only for the output of the entire pyramid and it is necessary to cater for the training of intermediate layers. An untrained PLN pyramid will output 0s and 1s with equal probability in response to any input pattern as all the nodes contain only "0.5" values. This means that it starts with 50% error . The following steps constitute a training algorithm for a single pyramid:

1. Identify the training set. This must consist first of a set Z^0, say, input patterns for which the response is to be 0, say : $Z^0 = (z^0_1, z^0_2, \ldots)$. Similarly the set for which the pyramid must respond with a 1, that is, $Z^1 = (z^1_1, z^1_2, \ldots)$ is defined.
2. A training *regime* must be selected. This refers to any specific way in which the two sets above are to be presented to the net. For example, the two sets could be applied in random order, interleaved, one after the other, and so on. In quoting experimental results it is always worth stating the regime, in case it might have had an influence.
3. For the first (or, in general, next) training pattern applied to the input, if the output is consistently right (e.g. remains at j for a pattern that belongs to Zj), do nothing. If the output is varying in time between 0 and 1, as soon as it becomes right, examine the current output of all nodes and ensure that this becomes stored. This means that all the nodes whose input currently addresses "0.5" will have the "0.5" replaced by their current output (which, as a result of the "0.5", has been arbitrarily selected). If the output is consistently wrong, all nodes that output a consistent output (i.e. do not have their "0.5" addressed) have the content of their stored location returned to "0.5".
4. Repeat the above, applying the training patterns according to the regime until no errors are detected or the error rate reaches an irreducible minimum.

In general, this algorithm will find appropriate values in the hidden layer and hence not fall prey to "hard learning" difficulties. More taxing tasks, (i.e. the problem of detecting parity) are discussed in Aleksander (1989).

Clearly, should the PLNs store more than three messages the training algorithm needs to be modified. Step 3 above may read something like:

3. For the first (or, in general, next) training pattern applied to the input, if the output is consistently right (e.g. remains at j for a pattern that belongs to Zj), do nothing. If the output is varying in time between 0 and 1, as soon as it becomes right, examine the current output of all nodes and " encourage" the current output. This means that if the addressed location contains a probability of firing greater than 0.5, increase it by some amount ∂, while if it is less than 0.5, decrease it by ∂. . If the output is consistently wrong, all nodes that output a consistent output (i.e. do not have their "0.5" addressed) have the content of their stored location "discouraged" by altering it towards 0.5 by an amount ∂.

Myers (1989) discusses optimal ways of selecting values of ∂.

PLN PYRAMIDS: GENERALISATION

Space does not permit a detailed account of generalisation in PLN pyramids. It is simply asserted that the pyramid has a characteristic that strongly distinguishes it from the discriminator discussed above. It is possible to make the pyramid very sensitive to samll differences. For example it is possible to make the pyramid fire 100% 1 for the all-1 input pattern and 100% 0 (i.e. 0% 1) for all single-0 pattern. Aleksander and Morton (1989), contains details.

SUMMARY

In this paper, weight variation has been replaced by updating procedures for the content of the memory neurons and it has been shown that the central characteristics of perceptrons, Hopfield nets (seeking of energy minima at run time), Boltzmann machines (training of hidden units, escape from local minima) and learning from errors (by methods that are more direct and faster than error back-propagation) have been retained. The methodology however offers additional parameters such as directness of implementation using digital logic techniques, choice of synchronicity (asynchronous systems are fundamentally stable, synchronous ones require a scheme of noise reduction in training , but can provide a richer set of state changes) or the use of probabilistic outputs from the nodes.

At a theoretical level, an analysis (described in Aleksander and Morton, 1989) using Markov chains provides a simple link between the state transition diagram of a logic/memory net and the energy concepts that are use in weight-based systems.

REFERENCES

1. Aleksander, I., and Morton, H.B, *An introduction to neural computing*, Boston: MIT pr., 1989.
2. Aleksander, I., "The Logic of Connectionist Systems", in Aleksander, I., (ed.), *Neural Computing Architectures*, Boston: MIT press , 1989
3. Hopfield, J. J.,, "Neural networks and physical systems with emergent computational abilities". *Proc.Nat. Acad.of Sciences, USA*,79, pp. 2554-2558, 1982.
4. Hinton,G.E. Sejnowski, T.J., and Ackley D.,"Boltzmann Machines: Costraint Satisfaction Networks that Learn" (Tech.Rep.CMU CS84,119,Carnegie-Mellon U) 1984.
5. Myers, C. E., 1989, "Output functions for probabilistic logic nodes", IEE Conf, Neural Nets, London, 1989.

Optimization Search Using Neural Networks

Henzer Chen and Shuo-Jen Lee
GE Corporate Research and Development
P. O. Box 8
Schenectady, NY 12301

ABSTRACT

This paper discusses a novel application of using neural networks for nonlinear search. It is well known that a feedforward network can perform function approximation at any degree of accuracy if enough data and enough hidden neurons are used. In optimization of an implicit function, numerical data is generated sequentially following the optimization algorithm toward the function optimum. Neural networks can conduct a search by approximating the implicit function using sequentially calculated data. The appoximation is refined by adapting newly generated training sets and finally the search leads to the finding of a maximum or a minimum. A comparison study using traditional search algorithms was also conducted. The result shows some promise of using neural networks for function optimization.

INTRODUCTION

Neural Networks have been characterized as a robust, distributed AI tool for problem solving, which can perform adaptive learning instead of using sophisticated algorithms. The most mature and widely used network model is the feedforward network with backpropagation training. Sometimes this network model is referred to as a mapping network, since it does nothing but map between input and output. A backpropagation network has been used to approximate arbitrary functions defined as sets of input variables and their respective function values[1]. With sufficient hidden layer neurons, it can provide fairly accurate approximation. Most significantly, these function approximations can be done without specified explicit function forms. Deploying a large number of hidden neurons, it can increase the accuracy of the mapping similar to other data fitting schemes, but it will lose the capability of generalization. This is true especially when only a small fraction of data is used to represent a wide range of mapping. With the capability and limitation of a backpropagation network in mind, an optimization procedure was developed.

OPTIMIZATION PROBLEM

In an optimization process, numerical search is implemented over the function definition domain. For unconstrained optimization, factors that have a strong effect on optimization results are problem formulation, initial design point, optimization technique and convergent criteria. Using neural networks to conduct optimum search can have a more robust outcome than traditional algorithms, i.e., it depends less on the initial design points and uses more effectively the data that has been analyzed.

To demonstrate the neural network search algorithm, the banana function[2] was selected to simulate the optimization process. The optimization of the banana function using traditional algorithms is also performed for comparison. The banana function is an algebraic function defined as:

$$F(x,y) = 10\,x^{**}4 - 20\,x^{**}2\,y + 10\,y^{**}2 + x^{**}2 - 2\,x + 5$$

where $-2.0 < x < 2.0$, and $-1.0 < y < 4.0$. The iso-value plot of the banana function is shown in Figure 1. This function resembles the shape of a banana with highly nonlinear numerical characteristics when far from the optimum and a flat plateau when near the optimum. Because of the difficulties in obtaining its minimum, it is widely used for testing new optimization algorithms.

NONLINEAR PROGRAMMING APPROACH

Numerical experiments using ADS optimization software[3] on optimization algorithms such as Fletcher-Reeves, Davidon-Fletcher-Powell(DFP) and Broydon-Fletcher-Glodfarb-Shanno(BFGS) and various linear search techniques such as golden section, golden section plus polynomial and polynomial interpolation were performed. The variation on performance for different combinations of these methods was significant. The DFP method with polynomial interpolation was found to be the best in

terms of optimization solution and number of analysis calls, therefore, it was chosen for further optimization runs.

Eleven starting points spread over the design range were chosen to conduct searches. The first nine starting design points were also used as the initial training sets for neural net approach. Although the optimization results shown in Table 1 were generally close to the true optimum of 4 at (1,1), the optimal parameters were widely scattered. For example, the search starting at (-2,4), which was furthest from the optimum, had a low initial value(13) but took the highest number of iterations(16) and analyses(103) to reach the second best result(4.003). On the other hand, the search with the closest starting point (0,0) had the lowest initial value(5), took 13 iterations and 76 analyses to reach the final result of 4.014.

TABLE 1 Summary of Optimization Results of Banana Functions Using ADS Optimizer[3] with DFP technique, Option (0,2,3)

Starting Point	Optimal Parameter	Initial Obj.	Optimal Obj.	No. of Iter.	No. of Analysis	Termination Criteria
(-2, 4)	(0.944, 0.888)	13.000	4.003	16	103	absobj, delobj
(0, 4)	(0.651, 0.357)	165.000	4.166	5	39	absobj
(2, 4)	(2.000, 4.000)	5.000	5.000	1	5	K-T condition
(-2, 2)	(0.952, 0.904)	53.000	4.002	14	89	absobj, delobj
(0, 2)	(0.759, 0.613)	45.000	4.073	8	57	absobj
(2, 2)	(0.925, 0.875)	45.000	4.009	4	40	absobj
(-2,-1)	(0.920, 0.976)	263.000	4.173	4	33	absobj
(0,-1)	(0.813, 0.658)	15.000	4.035	8	55	absobj
(2,-1)	(0.358, 0.160)	255.000	4.422	4	33	absobj
(0, 0)	(0.883, 0.775)	5.000	4.014	13	76	absobj, delobj
(0,1.5)	(0.758, 0.596)	27.500	4.063	9	63	absobj

NEURAL NETWORKS APPROACH

In applying backpropagation networks for nonlinear search, one needs to choose an appropriate number of hidden neurons such that the network can perform good mapping but does not lose the generalization. It is beyond the scope of this study to investigate the criterion for an optimum number of hidden neurons. A feedforward network with one hidden layer of four hidden neurons was used throughout the study. Initially, a number of training data sets, about the number of the connections, were required. In this study, the first nine starting points in Table 1 were used. Since the data sets used to train the network are sparse at the beginning, it usually comes out an approximation very different from the real function. The approximation can then be improved by using more training data sets generated in selected input domain based on the previous trained network.

The initial fitted function surface is plotted in Figure 3, while the banana function surface is shown in Figure 2. At this stage, no information about the global minimum is revealed in Figure 3. But based on the fitted surface(Figure 3) four additional (x,y) sets were selected, which represented two maximum and two minimum values in the mapped surface(see Table 2). Numerical experiments (i.e., to evaluate their banana functions) were then conducted and the new data sets were used for the succeeding training.

TABLE 2 Selected (x,y) Pairs for Numerical Experiments

(x,y)	Fitted Value	Banana Function	(x,y)	Fitted Value	Banana function
(-1.2,2.5)	12.727	20.076	(-0.8,1.5)	13.182	14.636
(1.6,0.5)	249.677	46.796	(1.2,1.5)	231.441	4.076

The result from the second round training is plotted in Figure 4. It is seen that the added information has improved somewhat the approximation of the real surface. After a couple more training runs, the approximation improved further(Figure 5) while the sum of the squared error for the training sets also increased substantially from zero. It is suspected that further addition of training data can improve the mapping, since the sum of the squared error will increase continuously. To conduct

further search, choices are either to increase the hidden layer size or to drop some training sets which are far away from the minimum of the fitted surface. In this study, the latter was chosen. Dropping the training data sets is equivalent to reducing the search range. This action will be successful only if sufficient data sets have been used for training, i.e., the generalization of the neural net mapping can reflect the real function landscape.

By narrowing the range of search, the valley of the mapped surface (Figure 6) was gradually approaching the real valley of the banana function (Figure 7) in the succeeding searches. Training sets were dropped again after sufficient training data was accumulated. It was found that after 25 analyses, the search had reached (1.02, 1.06) for a function value of 4.004. The progression of the optimization process is presented in Figure 8 by the graph of minimum value searched versus the number of numerical experiments.

DISCUSSION

The benefit of using the neural networks for numerical search can be realized in the situation where numerical experiment is very expensive, for instance, using the finite element method to implement engineering analysis and design. The advantage of a neural net search should become more significant when search dimension increases. In the numerical optimization algorithms, only the last few data sets are used to conduct the search, while the neural network approach can use all data obtained during the progress of the numerical search. Therefore, the global optimum may be more likely found in the neural net search than in the traditional search methods.

In most of the earlier search algorithms, initial search point is critical to the success of the optimization. Therefore, multiple searches with intuitionally selected initial starting points have to be conducted before a good optimization result can be of certain. Contrary, the neural network search doesn't depend much on the initial training sets.

The training of a feedforward network using back error propagation is slow, especially when the number of connections is large. Using fewer hidden layer neurons seems more attractive during optimization. However, using too few hidden layer neurons, the accuracy of the function approximation may be jeopardized. As a result, more numerical experiments will be required. The optimum number of hidden layer neurons for a given problem depends on the number of dependent variables and the complexity of the function. Further study in selecting a suitable hidden layer size is desirable.

It is seen that the neural network search can reach the vicinity of optimum quickly, and then the search efficiency drops substantially. This can be explained by the nature of a neural network in that a neural network can approximate a function robustly but usually has difficulty in obtaining a great degree of accuracy. There might be an advantage in switching the search from a neural network to another method when a neural net search becomes less efficient.

CONCLUSION

After studying a feedforward network for optimum search and comparing it to the nonlinear programming search algorithms, the preliminary results indicated that neural nets can perform better than the traditional search algorithms. Neural net search reaches the vicinity of the optimum in a very few trys, but further improvement requires narrowing the search range with the help of the approximated function generated by a neural net. However, the selection of the optimum hidden layer size and the criteria to terminate neural net search need further investigation.

REFERENCES

[1] "Theory of the Backpropagation Neural Network", Robert Hecht-Nielsen, Proceedings of IJCNN Vol. 1, p. 593, June 1989.

[2] "Numerical Optimization Techniques for Engineering Design: with applications", Garret N. Vanderplaats, McGraw-Hill Book Company, 1984.

[3] "Copes/ADS - A Fortran Control Program for Engineering Synthesis Using the ADS Optimization Program", Garret N. Vanderplaats, Engineering Design Optimization, Inc..

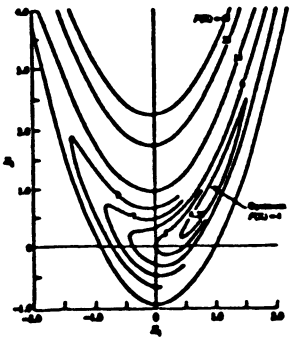


Figure 1. Iso-value Plot of Banana Function

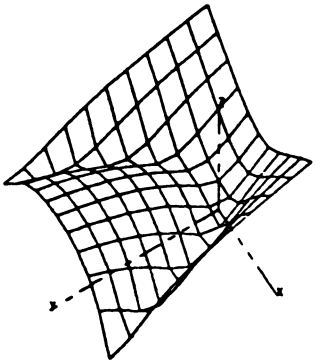


Figure 2. Banana Function

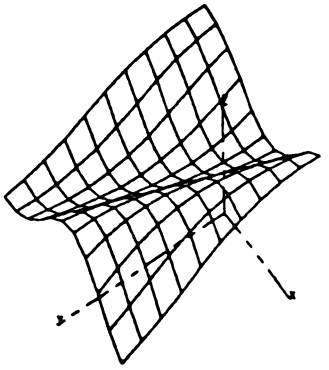


Figure 3. Approximation of Banana Function

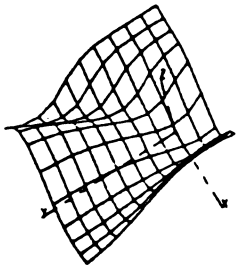


Figure 4. Approximation of Banana Function

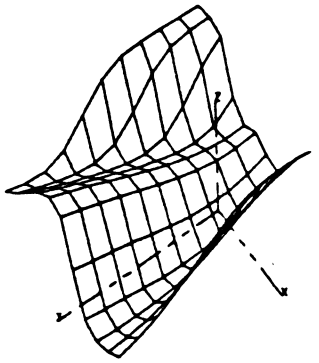


Figure 5. Approximation of Banana Function

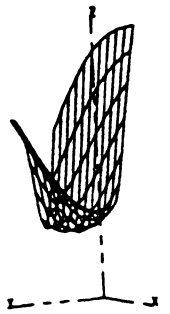


Figure 6. Approximation of Banana Function

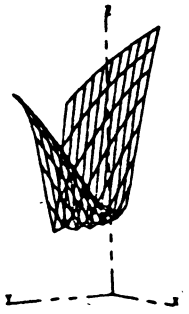


Figure 7. Banana Function(Narrowed Range)

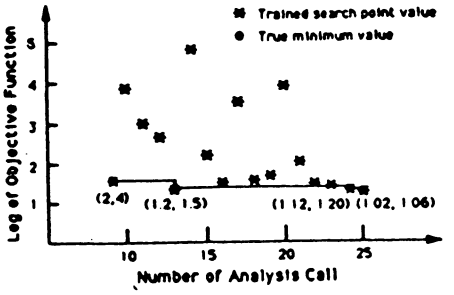


Figure 8. Progress of Optimization

Fault Tolerant Random Mapping Using Back Propagation

Kejitan Dontas, Jayshree Sarma, Padmini Srinivasan, Harry Wechsler
School of Information Technology and Engineering
George Mason University
4400 University Drive
Fairfax, VA 22030

Abstract

The paper describes the architecture and performance of neural networks capable of learning and computing complex mappings using the back propagation algorithm. Implementation of hashing functions is an important application of these connectionist networks, offering several advantages over traditional hashing methods such as robustness, and limited error correction. The user does not have to provide the hashing function explicitly, nor does he/she has to worry about collision detection and overflow management. Multiple indexing can be easily implemented by training a network for each key attribute. The neural network approach can be used to train very large number of pattern associations by dividing the problems into smaller problems. Our neural network consists of several subnetworks, each of them solving a specific mapping task. This could facilitate incremental learning and selective forgetting relatively efficiently. Our experimental results show that small neural networks with simple processing elements can learn complex mapping which implement index search in constant time.

1. Introduction

A neural network with **n** inputs and **m** outputs can be seen as a hardware implementation of some transform function or mapping. Hashing functions (Lewis and Cook, 1988) are a very important class of mappings used in database applications to store and retrieve records given some subfield, which acts as the key attribute. This should be done with minimal search effort, ideally in a single access. Databases are usually not static, and many updates in the form of additions and deletions are made, which makes sequential data organizations awkward. A hashing function which maps the key attribute to an index, should have several desirable properties. It should avoid clustering. Collisions should be kept to a minimum, and the hashing function itself must be simple enough so that computation of the function is fast compared to access on mass storage. The hashing function should also not waste mass storage. It is not possible to design two hashing functions such that two key attributes for a given record would map same index value for each record. Neural networks, on the other hand, can incorporate multiple indexing by training the network on each key attribute. Hashing is very sensitive to even minute errors in the input and usually yields unpredictable indexes for such cases. Apart from the input level, fault tolerance is also desirable at the hardware level. We show that neural networks (NN) can implement efficient and robust mappings.

An example of a hashing function is to compute an index of disk storage from a key attribute such as *name* or *Social Security Number* (SSN.) The number of distinct possible names or SSNs is very large, and in a typical application, such as a payroll database, we are interested only in a very small fraction of possible names or SSNs 1000s or 100,000s (depending on application) rather than billions of combinations. A hashing function is chosen such that each name in the given database maps to an index. In general this mapping is many to one and causes conflicts. Conflicts (or collisions as they are called) are resolved by special techniques such as chaining, bucketing and overflow management (Maurer and Lewis, 1975). We show how one can implement mapping of one large space on another space, using the methodology of neural networks (Lippmann, 1987). In particular, we have designed and built connectionist networks which would take names as input and give a nine digit index as output. The neural networks exhibit fault tolerance for some predefined classes of errors, and can be easily extended to incorporate multiple indexing by training additional networks to compute indexes for other key attributes.

2. Parallel and Distributed Information Retrieval

Parallel search techniques are characteristic of an emerging area of research known as Parallel Distributed Processing (PDP) or Neural Networks (Anderson and Rosenfeld, 1988). There are few working PDP attempts for parallel retrieval and/or inference, and they have been implemented using Distributed Associative Memory (DAM) (Kohonen, 1988) or the Connectionist (Ballard, 1986) models. DAMs are akin to physical holograms, and result when Hebbian-like (classical conditioning) associations between stimulus and response vectors are distributed across the memory. Connectionist models are multi-layer networks which accrue evidence for some interpretation. These models include 'hidden units' and the synaptic weights connecting the units are the result of learning and measure the relative

strength of the interconnections. Success of a neural network model is judged based upon its performance subject to the following criteria.
a) **Scalability:** A network should be scalable, i.e. the number of nodes and learning time or recall time should not grow faster than the complexity of the problem, (measured by the number of names that must be mapped, for example.) Ideally the network should correctly map as many input-output pairs as possible without altering the number of nodes or interconnections in the network.
b) **Incremental learning:** Addition and deletion of records are essential capabilities for a database system. Incremental learning means that new associations should be learnt with incremental effort without altering the performance in the already-known knowledge base. Most attempts by current NN models to learn another association usually result in global weight changes which, in turn, lead to loss of all/most of the associations learned before.
c) **Fault tolerance:** This refers to the capability of the neural network to continue functioning, possibly with marginally degraded performance, in the presence of malfunctioning nodes or links.
d) **Performance:** Performance is related to accuracy and learning time. If a name on which the network has been trained is presented, we expect to retrieve an error free index for the individual, but in reality there may be a nonzero probability of error on recall. The network should be also able to learn associations fast.

3. Connectionist and Back Propagation

Connectionist models are multi-layer networks and back propagation is one of the learning algorithms used in training such networks (Lippmann, 1987.) The back propagation learning algorithm minimizes the mean square error between the actual output of a multilayer network and the desired (correct) output. The weight change criteria, which minimizes the mean square error requires the activation function to be differentiable as many times as there as are hidden layers. The network topology also has certain restrictions. Sigmoid functions defined as $f(\alpha) = 1/(1 + e^{-(\alpha-\theta)})$ are commonly used as activation function. i.e. if α is the algebraic sum of all the inputs to a processing element, the output level is $f(\alpha)$. The weights change during learning and the corresponding algorithm is briefly described below. 1) Let $x_0, x_1, .., x_{n-1}$ denote continuous valued input vector, $d_0, d_1, .., d_{m-1}$ denote desired output vector, and $y_0, y_1, .., y_{n-1}$ the actual output vector. 2) $w_{ij}(t+1) = w_{ij}(t) + \eta.d_j.x'_i + \alpha(w_{ij}(t) - w_{ij}(t-1))$, where w_{ij} denotes weight from node$_i$ to node$_j$. d_j is an error term for node$_j$. i) If node$_j$ is an output node, then $d_j = y_j(1-y_j)(d_j-y_j)$, ii) If node$_j$ is an internal hidden node, then
$d_j = x'_j(1-x'_j)\Sigma\, d_k w_{jk}$, where k varies over all nodes in the layer above node$_j$. η is called the gain factor and α is called the momentum factor. The term $d_j.x'_i$ indicates the correction to be done to the weights so as to minimize the total sum of squares in this cycle. However, all the input output pairs interact on the same set of interconnections, and it is desirable to reduce the rate of weight change. The gain term takes care of desired attenuation in the weight change rate. On the other hand, the momentum term reflects the desire to continue the weight change in the direction of the previous weight change. It assists in damping oscillations and instability in the network. Both the terms are chosen such that $0 < \eta < 1, 0 < \alpha < 1$. The algorithm iterates until the total sum of squares has reached the pre-specified level.

The Hopfield net approach (Lippmann, 1987) and DAM approach were not considered because there is no guarantee about the orthogonality of the inputs in our application. These network approaches are good when the inputs are orthogonal. There is the issue of storage; the Hopfield net is binary, so 8 character input would require 48 nodes. Since each node is connected to all other nodes, there would be a total of 48*47 links. However, the capacity of this net is about 15% of the nodes; hence, we would not be able to store more than about 7 names, which is of limited use. It is not possible to store more patterns by increasing the number of nodes since, with the same representation, the input to the additional nodes will be zero, violating orthogonality the condition. Hence, essentially poor response can be expected despite enormous increase in the size of the network! Similar difficulties apply to DAM as well. We need extremely large number of nodes together with the additional assurance that the input data pattern is highly orthogonal. We chose the back propagation learning algorithm for the connectionist model because it can handle non-orthogonal inputs, and can store many patterns in a small-sized network.

4. Connectionist Architecture and Experimental Results

All the networks described below implement connectionist models and were run on a SUN3/60 computer using the back propagation module (McClelland and Rumelhart, 1988). The networks map names to indexes and have the following constraints: names are limited to lowercase, the maximum length of a name is set to 8 characters in order to limit the complexity of the input. The names are assumed to be distinct (the same as in hashing, where the hash

function is defined from key attribute to an index), so that a case of one name, such as "*smith*" mapping to different indexes is eliminated in principle.

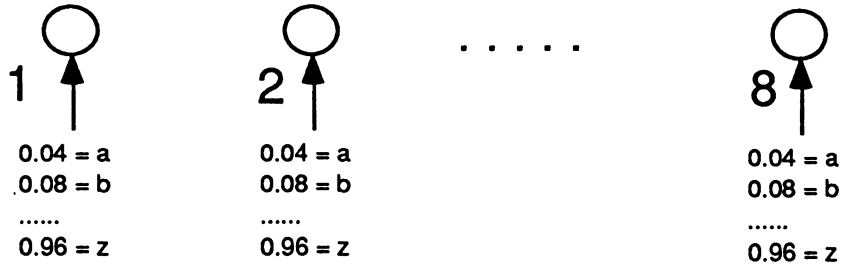


Figure 1: Input Encoding

The character input is encoded using a continuous range. Each character is represented as a single number between 0 and 1 and inputs to a single node. The 26 letters are represented by splitting the range of allowable input [0.0, 1.0] into 26 parts and assigning a unique sub-range to each letter. For example: *"a"* = 0.04, *"b"* = 0.08 and so on (Figure 1). At the output, decoding is done by identifying a character with a subrange rather than a single number. Thus, an output in the range 0.02-0.06 is interpreted as character *"a"*. Output in the subrange 0.06-0.10 is interpreted as character *"b"* and so on. The same approach is adopted for representing the 10 digits, for example, *"1"* = 0.1, *"2"* = 0.2 and so on.

Figure 2: Three Level Neural Network

Instead of constructing a single network to achieve fault tolerance, we trained three different networks, each performing a distinct function. The networks are organized as in Figure 2. The network N1 takes input name (possibly misspelled) and is expected to correct small errors and reduce large errors to smaller errors which can hopefully be corrected by other networks The network N2 takes the output from N1 as its input and recalls the associated indexes. Under normal conditions, correct indexes are retrieved. However, relatively large errors at the input of N1 still leave small errors at the output of N1 (which is the input of N2) and causes an erroneous index to be computed. Another source of error is inadequate training of network N2 itself. The possibility of errors in network N2 is greater than in N1 (or N3) since the mapping from names to indexes is random. These possibly erroneous indexes generated by N2 are fed to the network N3, which is expected to output corrected indexes (Figure 2). The network configuration is as follows:

1) N1 network: The input layer consists of 8 nodes, a hidden layer had 50 nodes, and output layer had 8 nodes.
2) N2 network: The input layer consists of 8 nodes, a hidden layer had 50 nodes, and output layer had 9 nodes.
3) N3 network: The input layer consists of 9 nodes, a hidden layer had 50 nodes, and output layer had 9 nodes.

Each of the three networks were trained independently, using the back propagation algorithm but connected during normal operation. Perfect recall on error free input required 1200 epochs. An epoch is defined as one training sequence of all patterns. This in itself is an encouraging result, since it demonstrates the usefulness of BP for efficient pattern association. Fifty names, each eight characters long, need 2000 bits of storage, assuming a minimal 5 bits representation for a character. Similarly, fifty 9-digit indexes with each digit encoded in four bits requires 1800 bits of storage. The network N2 contains only 67 nodes and is able to store 3800 bits of information encoded as a random mapping. Besides, this network was not yet saturated and could hold additional name-index pairs. The results indicated that the network is capable of learning complex (random) mappings very efficiently, and would work adequately if error-free input is guaranteed. However, these networks were sensitive to errors at the input, and the output was altered as well. For example, if the N2 network was given a misspelled name *"kaufmamn"* (correct name *"kaufmann"*) the output index was *"514790268"* instead of the correct index which is *"514790278"*. The network N3 was built and trained to map all combinations of small errors to a valid output. Thus, if the N3 network was given the incorrect output produced by the N2 network, for example *"514790268"*, then it is expected to correct this small error and map to the correct index. The input errors to the network N3 were provided by inadequately

trained network N2, i.e., weights after 700 epochs were used rather than after 1200 epochs, at which time the network performance was error-free. Three strategies were used to train N3 networks:
1) Each digit in the index was corrupted at random direction by +1 or -1. This gave 450 different digits in 50 indexes. The N3 network was trained on these 450 corrupted indexes plus the 50 correct ones.
2) Each digit in the index was corrupted systematically in each direction by +1 and -1. The 450 different digits in 50 indexes (which was corrupted twice, once in each direction), gave a total of 900 corrupted indexes. The N3 network was trained on these 900 corrupted indexes plus the original 50 correct indexes.
3) One N3 network was trained on the set of 50 error-free indexes to serve as a control.

The performance of each of them was about the same. In fact, the network trained on the correct indexes did a little better than the nets trained on the corrupted patterns. The networks trained with erroneous inputs were able to correct some errors, however, they also introduced some errors of their own (with error-free input) as a result of a tendency to deviate from the input. The N3 network trained completely on correct indexes performed better in terms of recall and correction to corrupted indexes compared to N3 network trained on corrupted indexes. This was counter to what was anticipated (evidence from Kukich, 1988), and yet another method was tried to generate an error-correcting network.

The N1 and N3 networks implement the identity function for error free input. There exists a trivial network performing identity function. A network of 9 input nodes, 9 output nodes with weights of 1 for corresponding input - output node pairs and weights of 0 for other input-output pairs performs such a function. Such a network, however does not generalize at all and passes all errors to the output unattenuated. We trained a three level network, each having 9 nodes. The links connecting nodes in identical positions in the three layers were initialized to weight of 0.9 and the other weights were initialized to 0.1. The idea was to train the network to correct small errors; the final network would still compute a function close to identity. The cross links between nodes in different positions would give a limited error correction capability to the network. Two networks were trained, one with error-free input patterns and the other with 950 training patterns. The weights observed for both the networks had similar structure. However, the error correction capability of the network trained on 950 patterns was not satisfactory. The network corrected some errors of small magnitude, but it also tended to introduce spurious errors of small magnitude when presented with error free input.

5. Conclusions and Future Research

We have shown in this paper that connectionist networks and BP are suitable for implementing random mapping. With appropriate representation, BP can allow the storage of large number of patterns in a small network. The networks do filter out some of the small errors. As of now, we do not have a good method for training networks for error correction without the overgeneralization effect. This may be remedied by further manipulations in network size, learning rate etc. Encouraging signs from the research show that it should be possible to efficiently interconnect networks trained independently to perform more complex tasks as well as to perform larger size of tasks. For example, 10 networks, each storing 50 indexes, can be combined horizontally to achieve a 500 index system. Alteration in such a database would require retraining of only the affected subnetworks, thus making the update process efficient. Multiple indexing can be implemented easily by training a network on each of the keys.

References:

Anderson, J.A. , and Rosenfeld, E. (Eds.) (1988), *Neurocomputing*, MIT Press.
Ballard, D (1986), Cortical Connections and Parallel Processing Structure and Function, *The Behavioral and Brain Sciences*, Vol 9, 67-120.
Kohonen, T. (1988) *Self-Organization and Associative Memory*. Springer-Verlag.
Kukich, K. (1988) Variations on a Back Propagation Name Recognition Net, in *Proceedings of the USPS Advanced Technology Conference*.
Lippmann, R.P. (1987), An Introduction to Computing with Neural Nets, *IEEE ASSP Magazine. No. 4*, 4-22.
Lewis, T. G., and Cook, C. R. (1988), Hashing for Dynamic and Static Internal Tables. *IEEE Computer. Vol. 21, No. 10*, 45-56.
Maurer, W. D., and Lewis. T. G. (1975) Hash Table Methods. *Computing Surveys, Vol. 7, No. 1*, 5-19.
McClelland, J. L. and Rumelhart, D. E. (1988) *Explorations in Parallel Distributed Processing, A Handbook of Models, Programs, and Exercises*, MIT Press.
Rumelhart, D.E., and McClelland, J.L. (Eds.) (1987), *Parallel Distributed Processing: Explorations in the Mircostructure of Cognition. Vol. 1: Foundations*, MIT Press.

Setpoint Control Based on Reinforcement Learning

Aloke Guha and Anoop Mathur
Honeywell Sensor and System Development Center
1000 Boone Avenue North
Golden Valley, MN 55427

Abstract

We present some new results on setpoint control based on reinforcement learning. Based on our results, we show that one can design adaptive controllers for setpoint control problems. Our primary application vehicle was the control of a nonisothermal continuously stirred tank reactor at its unstable state. Experimental results demonstrate good learning performance when reinforcement learning with state recurrence heuristics are used.

1. INTRODUCTION: REINFORCEMENT LEARNING AND SETPOINT CONTROL

Reinforcement learning has been applied successfully for simple control problems, such as the pole-cart problem, or the inverted pendulum problem, where the goal is to maintain the pole in a quasistable region, but not at a specific setpoint [1, 2, 3]. However, a large class of control problems requires maintaining the system at a desired operating point. In solving the problem of setpoint control, we have employed Barto's [1] Adaptive Heuristic Critic (AHC) that uses two linear elements, the Associative Search Element (ASE), responsible for the control actions, and the Adaptive Critic Element (ACE), that provides a secondary reinforcement to the ASE. The action output of the ASE is a binary output, y(t), defined by:

$$y(t) = f(\Sigma\, w_i(t)x_i(t) + \text{noise}(t)) \qquad (1)$$

where f is a thresholding step function, and $x_i(t)$, $0 \le i \le N$, is the current linearly decoded state. The added noise term results is stochastic learning. The weight updating rule during learning is given by equation (2).

$$w_i(t+1) = w_i(t) + \alpha\, r_1(t)\, e_i(t) \qquad (2)$$

where α is the gain, $r_1(t)$ the secondary reinforcement, and $e_i(t)$ is the eligibility trace [1] of the state $x_i(t)$.

Rosen [3] speeded up the AHC using the state recurrence heuristic that reinforces cycles around the desired operating region. The weight updating equation given by (2) is modified as follows:

$$w_i(t+1) = w_i(t) + \alpha\, r_1(t)\, e_i(t) + \alpha_2\, r_2(t)\, e_{2i}(t) \qquad (3)$$

where α_2 is a constant gain, $r_2(t)$ a positive reinforcement, and e_{2i} the state recurrence eligibility that is defined as follows for reinforcing shorter cycles more than longer cycles:

$$\begin{aligned} e_{2i}(t) &= \beta_2\, x_i(t) y(t_{i,lsat})/(\beta_2 + t - t_{i,last}), && \text{if } (t - t_{i,last}) > 1 \\ &= 0, && \text{otherwise} \end{aligned} \qquad (4)$$

where β_2 is a positive constant, and $t_{i,last}$ is the last time the system visited ith state $x_i(t)$.

Unlike previous work in applying reinforcement learning, we examined how it can be applied to applications that require controlling for specific setpoints not regions of success or failure. We were interested in examining i) the setpoint control of a complex highly nonlinear systems, and ii) the speed of learning since high learning rates are appropriate for quick in-situ learning and generalization. A further goal was to explore how the learning depended on the parameters of the AHC algorithm.

To define the points where reinforcements must be provided to the controller, we set tolerance limits around desired setpoint (or the state), say X_d. If the tolerance of operation defined by the level of control sophistication required in the problem is T, then the controller is defined to fail if $|X(t) - X_d| > T$. The controller must learn to maintain the system within this tolerance window 2T. If the level of discrimination within the tolerance window is 2T/n, then the number of states required to represent the control variable is n.

2. APPLICATION: UNSTABLE CSTR TEMPERATURE CONTROL

Our application vehicle for testing reinforcement learning for setpoint control was the unstable CSTR (continuous stirred tank reactor) [4] in which an irreversible exothermic reaction occurs (Figure 1). The exothermic heat released during the reaction is sigmoidal while the control coolant loop linear. The reactor has two stable states and an unstable state with the unstable state being the optimum operating point. Thus, if the CSTR reaches close to, or near the set limit of the asymptotically stable temperature limits, a failure is recorded. The CSTR temperatureis controlled by adjusting the real-valued coolant flowrate. In our initial experiments, we let the controller produce appropriate incremental changes of fixed gain to the initial flowrate given the current temperature and flowrate.

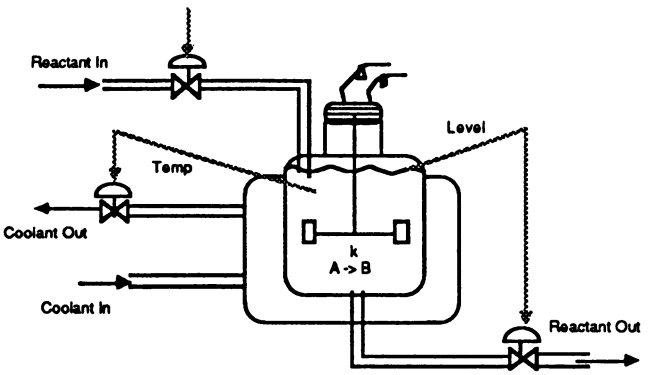


Figure 1. CSTR and Control

The following initial conditions were used. The initial temperature of the CSTR was chosen randomly from within the tolerance window 2T. The initial flowrate was chosen as either the median value (of possible flowrates) or one of two, high or low, values. A number of different experiments were run on: i) the effect of different parameters on learning rate, ii) the comparison of different versions of reinforcement learning , and iii) constant control output versus variable control output. The three versions of reinforcement algorithms were compared were:

1) The basic Adaptive Heuristic Critic (AHC) [1]
2) Adaptive Heuristic Critic with Recurrence Learning of Short Cycles (RSC) [3]
3) Recurrence Learning of Short Cycles near Desired Target (RSCDT) - an algorithm derived from the RSC algorithm by modifying the eligibility equation (4) as follows:

$$e_{2i}(t) = \beta_2 x_i(t) y(t_{i,last}) 2T/(\beta_2 + t - t_{i,last})/(2T + |x_i(t) - X_d(t)|) \quad \text{if } (t - t_{i,last}) > 1$$
$$= 0, \quad \text{otherwise} \qquad (5)$$

where the factor $2T/(2T + |x_i(t) - X_d(t)|)$ reinforces short cycles around the desired setpoint $X_d(t)$.

3. RESULTS

The parameter dependence experiments verify in part the results reported by Sutton [7] who found that the learning performance, measured by the number of steps until failure, is a bell-shaped curve with respect to the gain α. Our experience shows that the performance depends strongly the ASE gain parameter α (and also β) and the decay rate δ: AHC shows no significant change in performance for a wide range of α (10^{-2} to 10^{5}). However, RSC and RSCDT exhibit a performance improvement for some specific range of values of α (optimal values of α were not determined). For example, in the case of the RSC algorithm, a "good" value for α was 30000 (~10^{5}). Similarly, in case of RSCDT, the "good" value for was 1. The effect of varying δ is equivalent to varying the effect of the reinforcement on the the probability of actions taken before a failure occurs. Thus, a high value of δ will affect the most recent actions executed before a failure, while a low value of δ will affect the actions taken in the distant past. The desired value of δ therefore depends on the response time of the system.

Our initial learning experiments were based on a simulation model of the CSTR as a set of difference equations. All variables normalized between 0 and 1: desired temperature setpoint of 0.5, tolerance limits at 0.1 and 0.9. Only negative reinforcement values, [-1, 0], were used. We found that since the desired state not stable, a positive reinforcement provided at the desired setpoint tended to steer the CSTR away from the setpoint. Positive reinforcements were only used in experiments with state recurrence learning, in RSC and RSCDT. The performance of the controller was measured on three different criteria: i) mean number of steps to failure (N), ii) mean average temperature, and iii) mean

variance or the mean square difference (MSD), which for a given trial is defined as

$$MSD = \sqrt{(\Sigma (X_d - X)^2)}/N \quad (6)$$

where N is the number of time steps over which the summation is applied. Measurements of performance data were taken at increasing number of trials to observe learning and generalization. To minimize statistical variations, each experiment was averaged over 10 runs.

Figure 2 shows a plot of the mean number of steps to failure as a function of the number of trials. We note that state recurrence learning with or without reinforcing cycles near the desired target (RSC or RSCDT) performs much better than the basic AHC algorithm. There does not appear to be a significant difference in the performance of RSC and RSCDT algorithms. We only present the mean square difference MSD measure in Figure 3. RSC performs slightly better than RSCDT. We found, however, RSCDT does better than RSC when mean temperature is monitored.

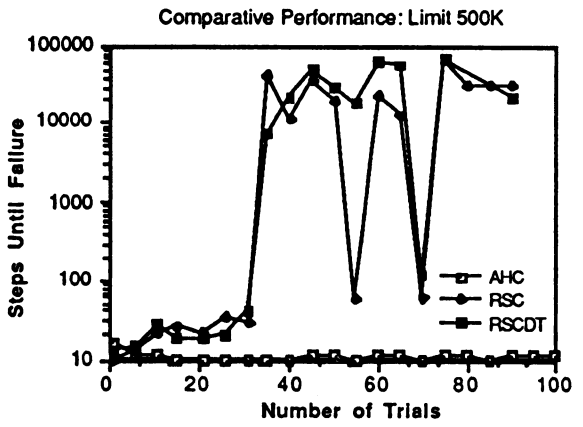


Figure 2. Learning Rate for Three Different Algorithms

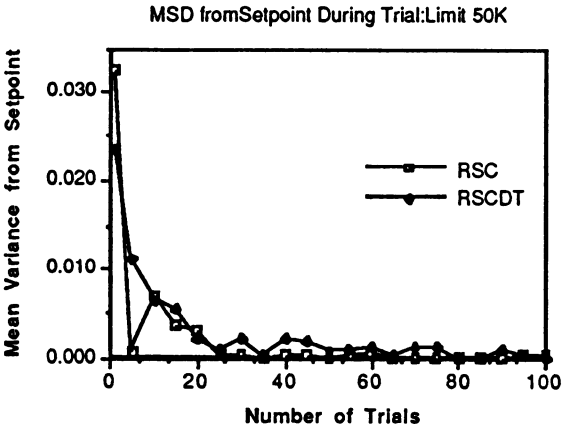


Figure 3. Mean Variance from Setpoint

As a test of how the placement of the tolerance window affects the mean temperature, an experiment was synthesized using asymmetric tolerance limits. We set the setpoint at 0.65 while the tolerance limits remained at 0.1 and 0.9. We found that the mean temperature was always in the range of 0.25 - 0.29. The system thus learns to avoid failure-prone region near 0.5. When the tolerance limits were placed at 0.3 and 0.9, which is still asymmetric but less than in the previous case, the mean temperature increased to between 0.53 - 0.55. Our empirical results indicate that the tolerance limits for setpoint control must be placed symmetrically about the setpoint.

A more accurate CSTR model, represented by a system of differential equations, was also examined. An added variable, control valve lag, was considered in some experiments. Another difference in the accurate model was the incorporation of absolute setpoint temperature values: desired setpoint of 600°, tolerance limits at 590° and at 610°. Also, the sigmoid function describing the heat rate was asymmetric. Two key results were obtained. First, for precise setpoint control, the constant gain controller does not work well. Second, continuous output control (flowrate) does. In our experiments, the control output gain was proportional to the reinforcement which in turn was proportional to the error. This resulted in faster learning. Our scheme is similar in philosophy to that of Gullapalli [6] where the mean and variance in the control output are explicitly controlled. The performance of the RSCDT algorithm is shown in Figures 4 and 5, where P and C indicate constant gain control and proportional gain control respectively. These experiments assume no valve lag.

While setpoint control can be achieved for systems without lag, for systems with lag some modifications are necessary in the controller design. The basic setpoint controller cannot learn the proper control for a lagging system because of the incorrect association of the failure and its most recent control action. In a system with large lag, the failure results from an action that was executed a number of steps earlier. One recommendation to correct this faulty association, beyond the scope of current experiments, is to include the control actions of the last few, say four, steps as part of the state definition.

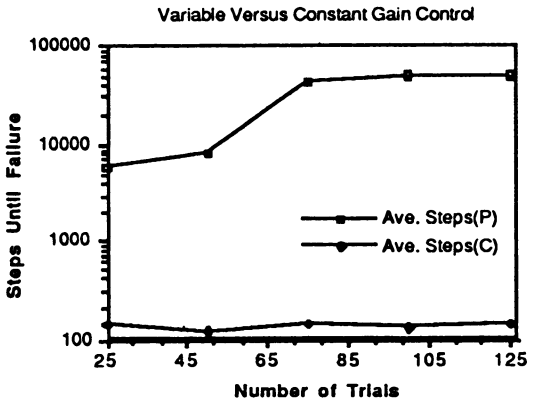


Figure 4. Mean Steps to Failure (RSCDT) for Accurate CSTR Model

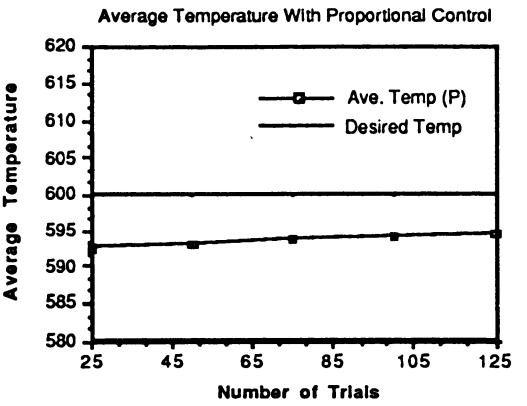


Figure 5. Mean Temperature (RSCDT) for Accurate CSTR Model

4. CONCLUSIONS AND FUTURE WORK

We have presented some interesting results of using reinforcement learning algorithms to develop adaptive setpoint controllers. A number of extensions to this work is anticipated. One immediate goal is to investigate how the learning strategy can be applied to systems with lag. Other experiments directly extend the current work for continuous control outputs [6]. Further, to avoid state explosion, we can also examine nonlinear encodings of the input space [5] by using multilayer inputs in the ASE. Another research issue that was uncovered was the reinforcement algorithm parameter selection. Besides using available knowledge about the system, other options include multivariable search techniques such as geneti < ˜ê 4Yet another extension of the current control strategy that can be investigated is that of optimization, that is, satisfying multiple control objectives such as minimizing energy expended while maintaining the desired setpoint.

REFERENCES

[1] A.G. Barto, R.S. Sutton, and C.W. Anderson, *Neuronlike Adaptive Elements that can solve Difficult Learning Control Problems*, IEEE Trans. on Systems, Man, and Cybernetics, September/October 1983.

[2] D. Michie and R. Chambers, *Machine Intelligence*, E. Dale and D. Mitchie, Ed.: (Oliver and Boyd, Edinburgh, 1968), p. 137.

[3] B. E. Rosen, J. M. Goodwin, and J. J. Vidal, *Learning by State Recurrence Detection*, IEEE Conference on Neural Information Processing Systems - Natural and Synthetic, AIP Press, 1988.

[4] W.B. Field, *Compatible Controls for Chemical Reactor Systems*, AIChE Workshop on Industrial Process Control, 1979.

[5] C.W. Anderson, *Learning and Problem Solving with Multilayer Connectionist System*, Ph. D. Dissertation, Department of Computer and Information Science, University of Massachussetts, Amherst, 1986.

[6] V. Gullapalli, *A Stochastic Algorithm for Learning Real-Valued Functions via Reinforcement Feedback*, COINS Technical Report 88-91, University of Massachussetts, Amherst, September 1988.

[7] R.S. Sutton, *Temporal Credit Assignment in Reinforcement Learning*, Ph. D. Dissertation, Department of Computer and Information Science, University of Massachussetts, Amherst, 1984.

[8] S.A. Harp, T. Samad, and A. Guha, Genetic Synthesis of Neural Networks, Proceedings of the International Conference on Genetic Algorithms, Washington D.C., June 1989.

Pattern Recognition of Handwritten Phonetic Japanese Alphabet Characters

Kazuhito HARUKI, Hisaaki HATANO
Toshiba Corporation, Systems and Software Engineering Laboratory
70, Yanagi-cho, Saiwai-ku, Kawasaki, 210 Japan

Abstract

In this paper, a new pattern recognition method using neural networks is proposed, and its performance on handwritten phonetic Japanese alphabet data is examined and reported. This neural network method is considered a useful improvement on conventional pattern recognition methods, such as the multiple similarity method. The effectiveness of the method has been tested on a data set of 32,000 handwritten phonetic characters. The recognition rate was improved from the 96.6% obtained by a mutiple similarity method to 98.0%.

Introduction

Recently, neural networks have been studied in various application fields, with regard to pattern recognition, control, optimization problems, and expert systems. Character pattern recognition is one of the fields where neural networks are expected to achieve new progress. Although research on neural networks, especially for the visual mechanism, has a long history [1], research and development of pattern recognition techniques for actual use has for decades been based on statistical and other approaches, rather than the neural network approach.

The neural network approach was not considered to be a realistic approach for the realization of an OCR (Optical Character Reader), which is an important functions in OA (Office Automation) appliances, because there was no theoretical foundation for the implication of connections, and there were no learning method adequately which covered the whole recognition process. In the proposed method, the neural network is applied only to the final decision making part of the recognition process. Therefore, the recognition rate can be always higher than the rate achieved by conventional methods.

Input/Output of Neural Networks

The structure of our neural network system is shown in Fig.1. The input data to the system is a set of similarities, calculated by the conventional pattern recognition method, "the multiple similarity method"[2]. Since there are 46 characters in the 'katakana' phonetic Japanese alphabet, the input consists of 46 values normalized into the range [0,1]. The output is the character's name recognized by the system.

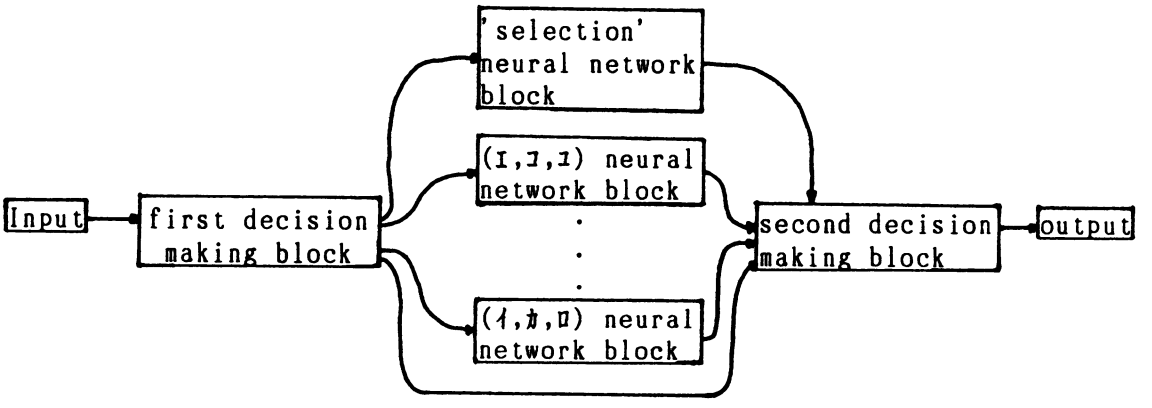


Figure 1: The proposed system of neural networks

Structure in Networks

The system consists of the first decision making block, 20 neural network blocks and the second decision making block, as shown in Fig.1.

There are 19 neural network blocks, each of which is specialized for recognizing characters from a particular set, such as (ㅣ, ㅋ, ㅈ), (ㅅ, ㅅ, ㅿ) and so on. The reasons for constructing 19 neural networks are as follows. 1) The multiple similarity method seems to make recognition errors only within small categorized group. 2) Training many small networks is easier than training one large equivalent network and reduces the total learning time, because convergence is faster and more reliable due to the small data set. The special 'selection' neural network block selects the right categorized group from the 19 categorized groups mentioned above. The 19 neural network blocks are structually identical: each neural network has 46 units on the input level, 10 units on the hidden level, and 3 units on the output level. The 'selection' network has 46 units on the input level, 50 units on one hidden level, 19 units on the output level.

Table 1-a: Distribution of total training samples

		difference between first and second largest similarity values										
		0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.10
largest similarity	1.00	191	251	515	704	945	997	1252	1125	1126	1194	11386
	0.95	246	242	276	297	368	399	454	413	455	464	5314
	0.90	122	102	97	96	116	108	128	125	111	111	1387
	0.85	57	44	50	37	53	42	42	41	35	30	277
	0.80	28	22	20	20	10	8	14	17	8	11	51
	0.75	16	10	5	4	4	6	3	4	5	4	7
	0.70	6	3	1	1	0	1	1	0	0	0	1
	0.65	0	0	1	0	0	1	0	0	1	0	0
	0.60	0	1	0	0	0	0	0	0	0	0	0

Table 1-b: Distribution of training samples on which the multiple similarity method fails

		difference between first and second largest similarity values										
		0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.10
largest similarity	1.00	60	32	15	9	2	2	5	2	3	1	2
	0.95	112	64	41	29	21	12	7	5	5	4	2
	0.90	52	31	23	11	8	7	3	3	4	4	4
	0.85	28	22	20	4	9	11	3	1	1	2	0
	0.80	19	15	10	6	4	3	3	0	0	2	2
	0.75	9	2	1	4	2	0	1	0	1	0	0
	0.70	2	2	1	0	0	0	0	0	0	0	0
	0.65	0	0	1	0	0	0	0	0	0	0	0
	0.60	0	1	0	0	0	0	0	0	0	0	0

The first decision making block determines whether or not the input data must be processed by the neural networks. The rule is; "if the difference between the first and second largest values is small, then the character selected by the neural networks is believed, otherwise the character, whose similarity value is largest, is selected." The distribution of data with respect to 'the largest similarity value' and 'the difference between the first and the second largest values', is listed in tables 1-a,-b. If the training sample, whose difference between first and second largest similarity values is less than 0.05, is used,

the data used to train the 'selection' neural network is reduced from 32,000 samples to about 5,000 samples. In the second decision making block, the final answer is decided according to the data sent from the 20 neural network blocks.

Learning by Sample Data

The data of 64,000 handwritten characters, were evenly divided to two groups, 1) training data and 2) test data (non-training data). As the dictionary pattern table in the multiple similarity method was made using the training data, the recognition rate of the multiple similarity method was 97.6% for the training data, which is better than the 96.6% rate for the test data. The weights in each of 19 networks were converged by the backpropagation learning method [3] after about 200 iterations for each of 2,100 training samples. The recognition rate for each neural network is shown in table 2, where values in () are recognition rates obtained by the multiple similarity method.

Table 2: Recognition rates for training data

group	recognition rate	group	recognition rate
1. イ,コ,ユ	100.0% (98.53%)	2. ハ,ヘ,ム	99.85% (98.36%)
3. ミ,ヨ,ヲ	99.93% (99.25%)	4. ソ,ツ,リ	99.07% (96.90%)
5. シ,ソ,ン	99.30% (96.90%)	6. シ,ル,レ	100.0% (98.62%)
7. チ,テ,ナ	97.84% (96.17%)	8. サ,テ,ナ	99.56% (98.78%)
9. ク,フ,リ	98.48% (97.75%)	10. コ,ニ,ユ	99.81% (98.44%)
11. ウ,ク,ラ	99.90% (98.79%)	12. ク,タ,フ	98.78% (98.19%)
13. ケ,チ,ナ	99.42% (97.42%)	14. キ,マ,ヤ	99.67% (99.24%)
15. オ,ネ,ホ	99.62% (98.62%)	16. セ,ヒ,モ	99.95% (99.42%)
17. ア,ス,メ	99.67% (98.76%)	18. ト,ノ,メ	100.0% (99.63%)
19. イ,カ,ロ	100.0% (99.78%)		

The 'selection' neural network learning needed 37 iterations to achieve the 95.9% recognition rate on about 5,000 training samples. The output error was reduced little by little as shown in Fig.2, but the recognition rate was not improved for the non-learning data after 37 iterations.

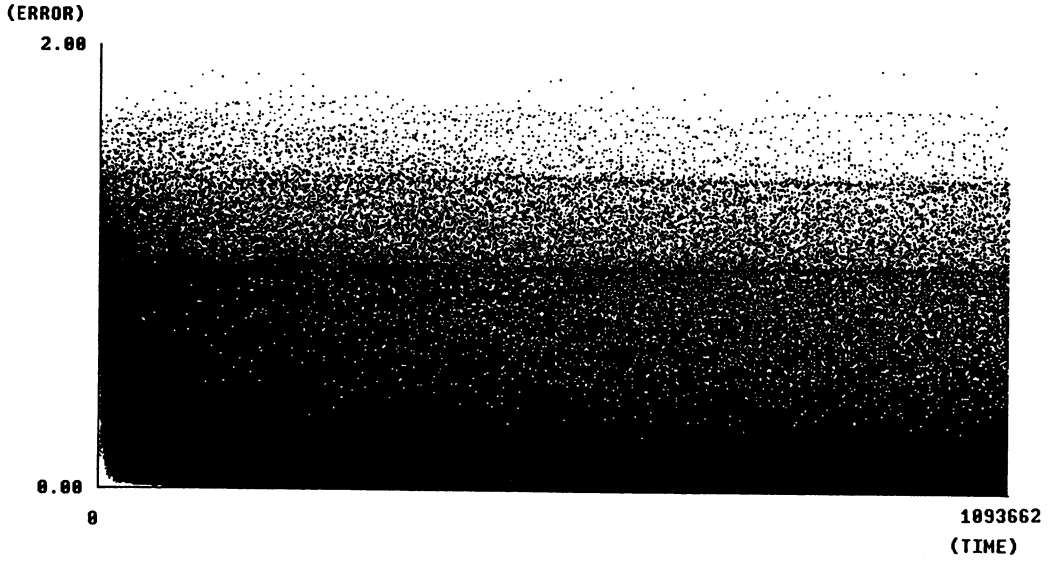


Figure 2: Error convergence in backpropagation learning

Recognition of the Test Data

Each neural network block recognized the test data at the rate listed in table 3. The selection of these neural networks was performed at the 91.4% accuracy rate. For the test data, the recognition rate after being processed by the system of neural networks was 98.0%, while the rate given by the multiple similarity method was 96.6%.

Table 3: Recognition rates for test data

group	recognition rate	group	recognition rate
1. イ,コ,ユ	98.58% (97.35%)	2. ハ,ヘ,ム	99.18% (97.80%)
3. ミ,ヨ,ヲ	99.69% (98.70%)	4. ソ,ツ,リ	97.93% (95.50%)
5. シ,ソ,ン	97.81% (96.52%)	6. シ,ル,レ	99.79% (98.51%)
7. チ,テ,ナ	96.66% (95.58%)	8. サ,テ,ナ	99.02% (97.59%)
9. ク,フ,リ	98.09% (96.57%)	10. コ,ニ,ユ	99.15% (97.68%)
11. ウ,ク,ラ	98.50% (98.41%)	12. ク,タ,フ	98.25% (97.66%)
13. ケ,チ,ナ	98.34% (96.48%)	14. キ,マ,ヤ	99.14% (98.76%)
15. オ,ネ,ホ	98.72% (98.30%)	16. セ,ヒ,モ	99.37% (98.99%)
17. ア,ス,ヌ	98.91% (97.81%)	18. ト,ノ,メ	99.86% (99.81%)
19. イ,カ,ロ	99.87% (99.60%)		

In this method, there are three sources of error: 1) the first decision making block, 2) the 'selection' neural network block, and 3) the neural network blocks of the categorized groups. The error rate 2.0% was the sum of the 0.5% error rate due to the first decision making block and the error rate 1.5% due to the 'selection' neural network block. There were no errors due to the neural network blocks of the categorized groups. In order to further reduce errors due to the 'selection' neural network, a new structure of the neural network and the learning method seem to be needed. In order to reduce errors due to the first decision making, it seems that the parameters in the multiple similarity method should be adjusted.

Conclusion

This proposed method gave us very promising results for an OCR application of neural networks. However, there are two problems to be overcome before implementing the method on OCR equipment. The first problem is that training is extremely slow. The second problem is that the present learning method is not sufficient for processing new additional training data. We plan to examine the effectiveness of the proposed method with many kinds of real data gathered from various fields, and also explore what neural network structures and training algorithms are well-suited for character recognition.

References

[1] Fukushima,K.: A Neural Network for Visual Pattern Recognition, COMPUTER, March 1988

[2] Iijima,T., Genchi,H., and Mori,K.: A Theory Character Recognition by Pattern Matching Method, Proc. First ILCPR, pp.50-56, 1973

[3] Rumelhart,D.E., McClelland,J.L. and the PDP Research Group: Parallel Distributed Processing, MIT Press, 1986

SMILES, PARITY, AND FEATURE RECOGNITION

J M Minor and R L Waterland
E. I. DUPONT
ENGG DEPT
Newark, DE 19714-6090

ABSTRACT

A previous article[1] described the similarity least squares (SMILES) procedure and its associated artificial neural network with optimal number of hidden nodes for general vector mapping. This paper discusses network behavior when this number is smaller due to latent structures in specific applications. In this case the network is forced to fit associations using reduced dimensions where it searches for basic factors or features which adequately reproduces the higher dimensional patterns.

1. INTRODUCTION

The original problem is to determine a vector mapping function from p dimensions to q dimensions,

$$g(x) = y \ ,$$

from a sample of N data points. The domain x and range y data vectors are transposed and concatenated as N rows to form the matrices **X** and **Y**. Similarity least squares (SMILES) solves this problem using

$$w_x' \bullet C_j = \hat{y}_j \ ,$$

where the weight vector w_x is composed of vector similarity values, discussed below. The coefficient vector C_j is the j^{th} column vector Y_j from matrix **Y** plus corrections by the SMILES procedure to optimize the fit to the data. For moving average filters and local regression estimation, C_j equals Y_j.

It is well known that a feed forward artificial neural net (FFANN) can learn arbitrary vector mapping functions from its input (domain vector) through a hidden (representation) layer to its output layer (range vector). The SMILES method shows how to build such a network. A procedure is developed for designing a neural net requiring polynomial-in-time training effort to associate two finite sets of vectors. If the domain set of data vectors used to train the network is adequate, the net generalizes or predicts from an arbitrary input vector in a reasonable and stable fashion.

Similarity of two vectors is a symmetric scalar function which is monotone decreasing with the magnitude of the difference between the two vectors. The following expression is selected to calculate similarity between the row vectors of **X**:

$$r_{ij}^2 = \sum_{k=1}^{p} (X_{ik} - X_{jk})^2,$$

$$S_{ij} = \frac{1}{1+e^{+br_{ij}^2 - a}}$$ with scale factor b and shift a.

We will augment **X** with one column containing the magnitude squared of each row vector (denoted r^2) so that the corresponding FFANN will require only N hidden nodes to calculate S (since each node can calculate this similarity value between each data x vector and any network input vector from their vector magnitudes and dot product). See Figure 1. Each hidden node is associated with a unique specific point in the data set.[1]

2. FEATURE RECOGNITION

If the number of hidden nodes K is less than N, one could select K data points strategically located in the data set to construct the K columns of S so as to minimize the fit degradation. However the very best K points, called support points, may not correspond to any of the data points in the data set, but are optimally located in the domain of the vector mapping function. During training the FFANN will find these K optimal domain vectors as represented and stored in the weights of the K hidden nodes. See Figure 2.

3. APPLICATIONS

This restricted SMILES as applied to example problems produces an adequate fit to the data while using a similarity matrix of significantly smaller size. This indicates that features or factors produce the patterns in the data. In theory given adequate efficient samples of data, the full similarity matrix is required to represent the patterns in the data. In practice the optimal locations for data are not known for unknown functions. The SMILES support points found by neural net training tend to collect at or near max or min areas of the function surface as shown in Figure 3. For the N-bit parity problem, the N support points tended to be uniformly spaced at N locations on the main diagonal of the associated N-cube (with a support point at each end). Each point locates in a unique hyperplane of constant parity.

4. CONCLUSION

This paper shows how SMILES both restricted and unrestricted can be used to fit arbitrarily complex vector mapping functions and discusses the artificial neural net architectures which perform both SMILES procedures. Understanding SMILES gives one insights into how neural nets learn and perform tasks such as feature recognition, pattern associations, and the N-bit parity function.

REFERENCES

[1] J M Minor (1989). Smiles and Ffanns: Similarity Least Squares and Artificial Neural Networks, IJCNN89 Proceedings II-607

[2] Rumelhart, D. E., & McClelland, J. L.,*et al* (1987). *Parallel Distributed Processing.* Cambridge, MA: MIT Press.

[3] W S Cleveland (1979). Robust Locally Weighted Regression and Smoothing Scatter Plots, *J. Amer. Statist. Assoc.*, 74, 829-836.

[4] A Buja, T Hastie, and R Tibshirani (1989). Linear Smoothers and Additive Models, *Ann. Statist.* **17** 453-555

Figure 1. jth Hidden Node

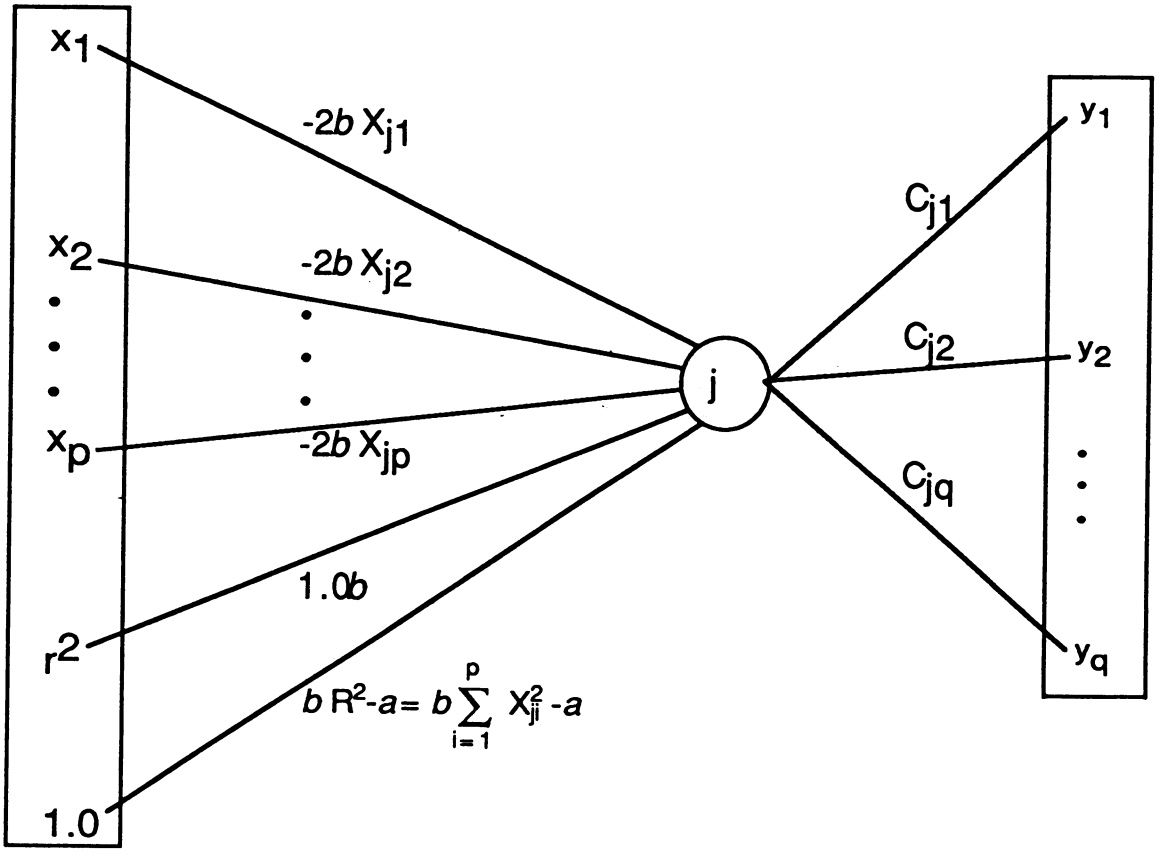


Figure 2. Parity or Feature Network

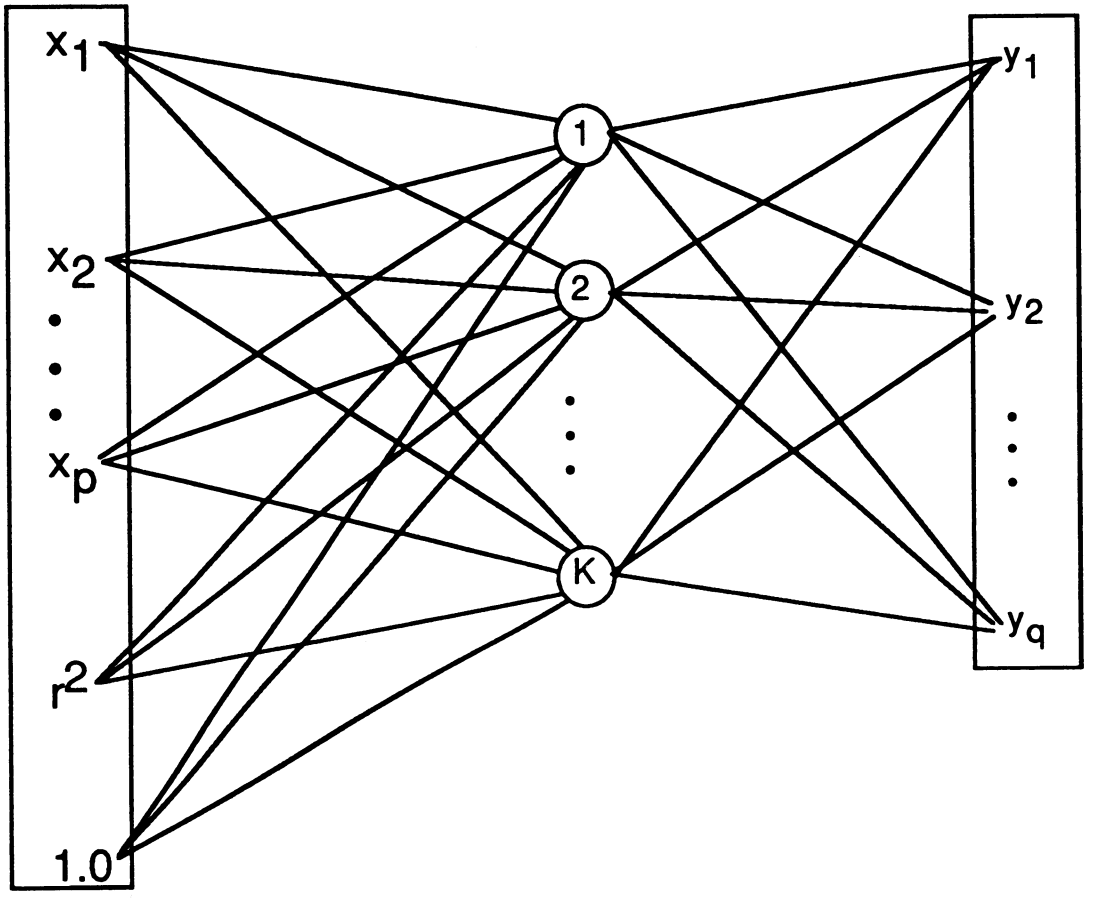
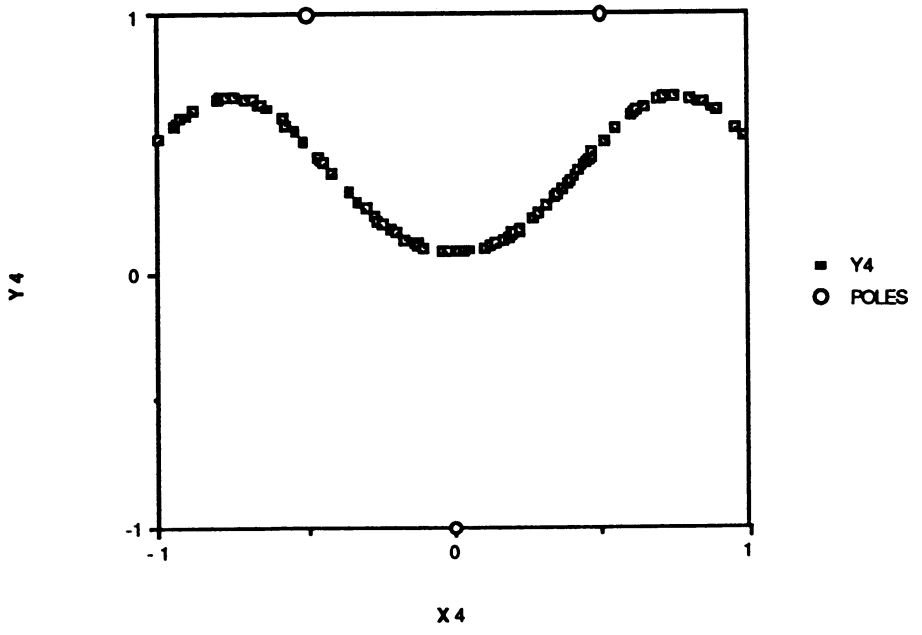


Figure 3. SMILES WITH THREE SUPPORT POINTS



Comparative Performance Measure for Neural Networks Solving Optimization Problems

Peter W. Protzel*

Institute for Computer Applications in Science and Engineering
Mail Stop 132C, NASA Langley Research Center
Hampton, VA 23665

Abstract A common problem in using neural networks to solve optimization problems is the lack of an appropriate measure to assess the performance. This makes it very difficult to compare the effects of modifications or parameter variations on the solution quality. The definition of a simple, comparative performance measure proposed in this paper can be used to overcome these problems. The use of the performance measure is demonstrated for two examples, the Traveling Salesman Problem and the Assignment Problem. The numerical results indicate a principle difference in the performance characteristics of the two types of networks caused by a quadratic versus a linear cost function as the only difference between the networks.

1. Introduction

Since Hopfield's and Tank's (H&T's) demonstration in 1985 [3] that a special type of Artificial Neural Network (ANN) can 'solve' complex optimization problems *in principle*, there has been much effort in applying H&T's method to other optimization problems, or in trying to improve on H&T's original results [1, 2, 5, 8, 9]. However, one problem is the apparent lack of statistically relevant measures to assess the actual performance of a modification or of a new approach. Since the performance of a network solving the Traveling Salesman Problem (TSP), for example, varies considerably not only for different city-distributions and different problem sizes, but also for different random initializations, it is usually not possible to prove a point or make a meaningful comparison by looking only at a few examples.

Therefore, it would be desirable to have some performance measure that gives a quantitative answer to the following questions: 1) What is the effect of a parameter variation or a certain modification of the energy-function on the solution quality, 2) How good is the solution with respect to the global optimum, 3) How does the performance change with problem size, 4) How does the performance degrade under the presence of (simulated) faults, and 5) What is the performance difference of two networks solving two different problems, that is, are there principal, problem-dependent performance differences in the ANN approach to optimization problems?

The definition of a performance index described in the next section is aimed at answering the above questions, and numerical results are given for two examples, the Traveling Salesman Problem (TSP) and the Assignment Problem (AP). The paper concludes with a discussion of the results and a personal perspective on the use of ANNs to solve optimization problems.

* This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1–18605 while the author was in residence at ICASE.

2. How Good Is Good: Definition of a Performance Index

It is a common practice to compare the ANN-solution for an optimization problem to the average or to the distribution of random trial-solutions by stating, for example, that a solution is approximately among the 10^8 best out of 4.4×10^{30} possible solutions [3], or that 92% of the solutions are among the best 0.01% of all solutions [8]. While this can provide some insight in the ANN performance, it is hardly a practical way of answering the questions raised above.

Our approach is based on the fact that every instance of an optimization problem has two distinct characteristics, the value for the global optimum (minimal cost) c_{opt}, and the average cost value c_{ave} for a sufficient number of random trials. Then, for a given ANN-solution c of that problem instance, a solution quality q is defined as

$$q = \frac{c_{ave} - c}{c_{ave} - c_{opt}} \tag{1}$$

By relating the difference between random average and actual solution to the difference between average and optimal solution, the quality index q becomes a normalized factor with a value q=1 if a given solution c is optimal ($c=c_{opt}$) and a value q=0 if the answer corresponds to an average random tour ($c=c_{ave}$).

While it is not difficult to obtain values for c_{ave}, the above definition of q does also require the knowledge of the optimal solutions c_{opt}. However, this is not a difficulty, because in almost all cases where ANNs have been applied to optimization problems, there are very powerful, conventional algorithms readily available to provide values for c_{opt}. For NP-complete optimization problems like the TSP, where the global optimum is generally unknown except for special cases, heuristic algorithms like the Lin-Kernigham algorithm [4] can be used to provide values for c_{opt}. A possible event $c<c_{opt}$ is reflected by a value q>1. A negative value for q indicates a solution that is worse than the random average ($c>c_{ave}$). Thus, the normalized value for q is independent of a particular problem instance and independent of the problem size. Numerical examples to illustrate the use of this performance measure are given in the next section.

3. Examples: Traveling Salesman vs. Assignment Problem

A detailed description of the original approach of using an ANN to obtain solutions for the TSP can be found in [3]. In order to get statistically relevant results we generated a test-set of 10 different city-distributions for each problem size of n=10, 20, and 30, and 5 different distributions for n=50 and 100. Values for c_{opt} and c_{ave} were obtained for each city-distribution by using the Lin-Kernigham algorithm [4] and 10^5 random trials, respectively. Since the performance varies considerably for different random initializations, 10 different initializations were used for each city-distribution of size 10 to 50, and 5 initializations for n=100. The value for q was calculated after each run of the simulator. The average values for q are shown in Table 1 for different approaches and problem sizes.

Another important performance criterion not reflected by q is the proportion of valid solutions, which is shown in Table 1 in parentheses. Since H&T's original equations did not consistently produce valid tours, we implemented and compared different modifications. The best results so far have been obtained by using the approach published by Brandt et al. (1988) [1]. All approaches in Table 1 use the same cost-function, but with different 'weight-factors' D [3]. The following list gives the ANN-parameters for the different approaches listed in Table 1 by using H&T's original notation:

1. $T_{Xi,Yj} = -500\delta_{XY} - 500\delta_{ij} + 1000\delta_{XY}\delta_{ij} - 200$, $I_{Xi} = 200(n+5)$, $D = 500$
2. $T_{Xi,Yj} = -400\delta_{XY} - 400\delta_{ij} + 400\delta_{XY}\delta_{ij} - 200$, $I_{Xi} = 200(n+1)$, $D = 90$

3. $T_{Xi,Yj} = -2\delta_{XY} - 2\delta_{ij} + 4\delta_{XY}\delta_{ij}$, $I_{Xi} = 2$, $D = 1$†
4. $T_{Xi,Yj} = -5\delta_{XY} - 5\delta_{ij} + 2\delta_{XY}\delta_{ij}$, $I_{Xi} = 9$, $D = 3$

Note that all listed modifications of H&T's original approach require a change of the energy-function and are not merely a variation of H&T's original parameters [1, 6]. All modifications use an additional self-inhibition as the main difference to the original approach. There seems to be an inherent tradeoff between obtaining a better quality and producing consistently valid tours. A forthcoming technical report includes a more detailed discussion of these results.

Traveling Salesman : Different Approaches	Problem Size n (Number of Cities) 10	20	30	50	100
1.) Original Method of Hopfield and Tank	0.905 (0.15)	0.903 (0.11)	0.851 (0.02)	- (0.00)	-/-
2.) Our Modification of Hopfield and Tank	0.836 (1.00)	0.844 (1.00)	0.808 (0.99)	0.817 (1.00)	0.860 (1.00)
3.) Modified Method of Brandt et al. (1988)	0.829 (1.00)	0.816 (1.00)	0.830 (1.00)	0.852 (1.00)	0.902 (1.00)
4.) Brandt et al. (1988), different parameters	0.936 (0.98)	0.926 (0.97)	0.923 (0.84)	0.913 (0.58)	0.927 (0.18)

Table 1. TSP solution quality q and proportion of valid solutions (in parentheses) for different problem sizes and modifications. The values shown are averages over 100 different simulation-runs each (for n=10, 20, and 30), 50 runs (n=50), and 25 runs (n=100), respectively.

Assignment Problem: Different Parameters	Problem Size n (Number of Elements) 10	20	30	50	100
1.) A=B=2, C=2, D=1	0.988 (1.0)	0.960 (1.0)	0.975 (1.0)	0.978 (1.0)	0.987 (1.0)
2.) A=B=200, C=20, D=50	1.0 (1.0)	0.999 (1.0)	0.999 (1.0)	0.998 (1.0)	0.998 (1.0)
3.) A=B=200, C=3, D=50	1.0 (1.0)	0.999 (1.0)	1.0 (1.0)	1.0 (1.0)	0.999 (1.0)

Table 2. AP solution quality q and proportion of valid solutions (in parentheses) for different problem sizes and parameters. Each value is an average over 10 different simulation-runs.

The Assignment Problem (AP), or sometimes called 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. What makes the AP interesting as an example for an ANN implementation is the surprising similarity to the TSP. Both problems have identical constraints and the only difference between an ANN solving the TSP or the AP is that the TSP has a quadratic cost function with an encoding of the data in form of interconnections, while the linear cost function of the AP maps the data onto the external current of the ANN. When the method of Brandt et al. is used to enforce the constraints [1], the interconnection values for the AP are $T_{Xi,Yj} = -A\delta_{XY} - B\delta_{ij} + C\delta_{XY}\delta_{ij}$, and the external current $I_{Xi} = A + B - C/2 - Dp_{Xi}$ contains the cost p_{Xi} for a pairing between X and i. A test set with 10 problem

† Additional differences of the approach of Brandt et al. (1988) are a lower gain, an offset in the transfer function, and an initialization in the center of the hypercube [1].

instances of random values for p_{xi} was generated for each problem size, and the values for the global optima c_{opt} were obtained by using a simple textbook algorithm [7]. Unlike the TSP, the ANN solving the AP does not require any random initialization and actually converges to the same solution despite of initial random perturbations. Numerical results for different parameters are shown in Table 2.

The results clearly indicate a fundamental difference in the performance characteristics between the TSP and the AP, given that the AP-Solver never converged to an invalid solution (with a 'constraint-network' identical to the TSP!) and obtained the global optimum in most cases with an average quality far better than the TSP. It is expected that ANNs with a linear cost function generally show a much better performance than those with a quadratic cost function. If this can be substantiated by either theoretical or empirical investigations, then this is an important classification that helps to identify problems which are more suitable to an ANN-implementation.

The use of conventional algorithms to solve optimization problems provides another interesting perspective on the ANN performance. For example, simulating an ANN to solve the TSP for n=100 cities takes more than a day CPU time on a VAX 780; the Lin-Kernighan algorithm obtains a much better solution in less than 3 minutes! Although a hardware implementation of the ANN might solve the problem in milliseconds, the need for a VLSI -chip with 10,000 integrated Operational Amplifiers to solve a 100–city TSP in real time is truly questionable. Thus, we do not think that large-scale, classical or NP-complete optimization problems are the appropriate candidates for an ANN approach. There are other, small-scale, special purpose, real-time control problems that could benefit from the key characteristics of ANN-hardware implementations: speed, fault-tolerance, and low weight and power consumption. Future efforts should concentrate more on identifying these problems than on fighting a battle against conventional algorithms that seems hard to win.

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] Hedge, S., Sweet, J., and Levy, W. Determination of parameters in a Hopfield/Tank computational network. In *Proceedings of the IEEE International Conference on Neural Networks, San Diego, CA* (July 1988), pp. II–291–298.

[3] Hopfield, J. J., and Tank, D. W. "Neural" computation of decisions in optimization problems. *Biological Cybernetics 52* (1985), 141–152.

[4] Lin, S., and Kernigham, B. W. An effective heuristic algorithm for the traveling salesman problem. *Operations Research 21* (1973), 498–516.

[5] Page, E. W., and Tagliarini, G. A. Algorithm development for neural networks. In *SPIE Symposium on Innovative Science and Technology, Los Angeles, CA* (January 1988).

[6] 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.

[7] Syslo, M. M., Deo, N., and Kowalik, J. S. *Discrete Optimization Algorithms*. Prentice Hall, Inc., Englewood Cliffs, NJ, 1983.

[8] Tagliarini, G. A., and Page, E. W. A neural network solution to the concentrator assignment problem. In *IEEE Conference on "Neural Information Processing Systems – Natural and Synthetic", Denver, CO* (November 1987).

[9] Wilson, G. V., and Pawley, G. S. On the stability of the traveling salesman problem algorithm of Hopfield and Tank. *Biological Cybernetics 58* (1988), 63–70.

FISH DETECTION AND CLASSIFICATION USING A NEURAL-NETWORK-BASED ACTIVE SONAR SYSTEM - PRELIMINARY RESULTS

N. Ramani, P.H. Patrick, W.G. Hanson and H. Anderson
Ontario Hydro Research Division
800 Kipling Avenue
Toronto, ONTARIO M8Z 5S4, CANADA

ABSTRACT

Sonar is currently being used to collect quantitative fisheries data (for classification, enumeration etc.) as a safer, cost-effective and environmentally attractive alternative to netting. This study explores the potential of a neural-network-based system in detecting and classifying the fish from the sonar echoes. Preliminary results are encouraging and a simple neural network was able to identify up to 91 percent of the test samples. When the identification problem is divided into three subproblems, over 93 percent of the samples are identified correctly. If borderline responses are discarded, the success rate increases to over 96 percent.

1. INTRODUCTION

Recently, there has been considerable interest in the use of active sonar for collecting environmental data. Sonar is currently being used to collect fisheries data by industry, university and government groups for assessing fish stocks and also for impact studies. This recent emphasis on sonar is attributable to the fact that data collection can be done more safely than present netting practices, is non-consumptive and is more cost effective and less labor intensive. However, much work has to be done before sonar can replace current netting practices. Although advancements have been made in the areas of developing real time data collection and analysis systems, existing systems are limited in their ability to detect or classify fish (a feature that is critical if sonar is to be used as a truly remote system), and the application of a neural network for the purpose may prove to be superior.

This paper outlines some preliminary work being done on laboratory data at Ontario Hydro to explore the potential of neural networks for distinguishing fish from debris and for speciating fish from the sonar echoes. Although neural nets have been used for other active sonar uses before (Gorman and Sejnowski 1988, Roitblat et al. 1989), this is the first fisheries application that we are aware of.

2. EXPERIMENTAL PROTOCOL AND DATA COLLECTION

A dual beam 420 kHz transducer (Biosonics) with time varied gain was used for the acoustic sampling. The reflected signals were received through a Biosonics 101 sounder and a wideband sonar detection module (WSDM), and were recorded using an FM recorder (Racal Store 7). The recorded data were digitized at a 60 kHz sampling rate for use by the neurocomputer (SAIC). A block diagram of the data collection and analysis system is shown in Figure 1.

The sampling was conducted in the laboratory where the transducers were placed horizontally in a large semi-anechoic tank measuring approximately 15.5 m in length and width and 1.5 m in depth. The dual beam transducer was placed at one end of the tank, and the targets were from 3 to 9 m from the transducer.

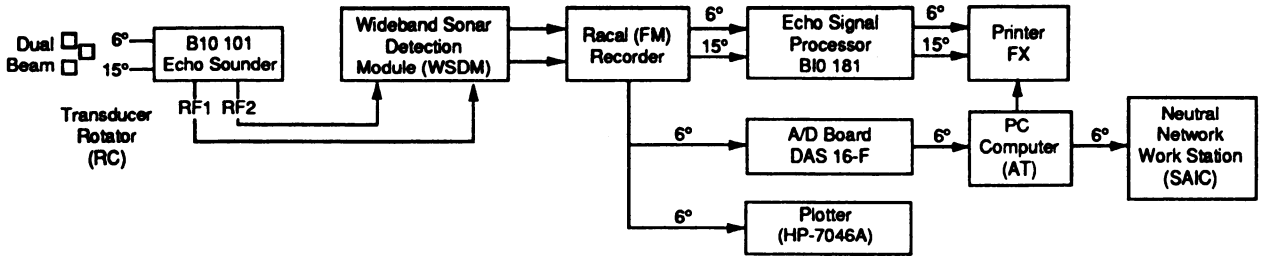


FIGURE 1 - DATA COLLECTION AND ANALYSIS SYSTEM FOR SONAR SIGNALS

Data were collected on single targets which included a ping pong ball, a leaf, air bubbles (defined as debris) and four species of fish (walleye *Stizostedion vitreum 70 cm* T.L., rainbow trout *Oncorhynchus gairdneri* 27 cm, brown bullhead *Ictalurus nebulosus* 23 cm, sturgeon *Acipenser fulvescens* 80 cm). Although it is somewhat unrealistic that these species and debris would occur in nature at the same time, it was felt that this would represent a good test case. Data collected on fish involved tranquilized (MS 222) individuals suspended approximately 6 m from the transducer using monofilament. The leaf and the ping pong ball were also suspended in the water column using monofilament. The monofilament line was not detected using the 420 kHz transducer. A continuous stream of small air bubbles was created using a wand (Bioweve Inc). The four fish samples were positioned laterally (broadside) with respect to the transducer. Two additional sets of signals for the sturgeon, with the fish positioned frontally and at 45 degrees to the transducer, were also collected, making a total of nine targets.

Single target data were obtained using a transmitted pulse width of 0.5 ms. Only the 6 degree data were used and forty eight signals from each of the nine different targets were digitized at 60 kHz. The pulses were transmitted every 0.5 seconds and thus the digitized data consisted of short pulses of signal echoes interspersed by relatively long periods of noise. The signal echoes (192 points each) were retrieved from the digitized data using the transmitted pulse as a guide.

3. DATA ANALYSIS

3.1 The Neural Network Architecture

Feedforward networks with three layers of neurons were used and the backpropagation algorithm (Rumelhart et al., 1986) was used to train the network. The first layer consisted of 192 units and the size of the output layer depended on the subproblem being studied, as described below. The number of units in the middle layer was varied from 10 to 30 units in an attempt to study the impact of the hidden layer size on the results. The weights of the network were initialized to random values between -0.3 and 0.3 and the learning rate and momentum factors (Rumelhart et al., 1986) were set at 0.01 and 0.6 respectively. All the simulations were done on a SAIC Sigma II Neurocomputer workstation.

The training set consisted of 38 of the 48 signals for each of the targets and the remaining 10 were used to test the trained network. Training continued until all the training samples were learned, or until learning saturated. The analysis consisted of three experiments described below.

3.2 Experiment 1

The first experiment consisted of a network with nine output neurons corresponding to the nine targets described above. Table 1 (case A) shows the performance of the network on the test set of 90 patterns. As the size of the hidden layer increases, the performance improves, but when the hidden layer size is increased to 30, the performance drops, perhaps a case of overtraining.

The best performing net correctly identifies an impressive 91.1% percent of the test patterns. In comparison, the average human (non-expert) performance on the same set was about 75 percent.

3.3 Experiment 2

It was decided that the network would next be tested by dividing the classification problem into three subproblems. In the first stage, the net is asked to detect fish from debris; in the second stage, to speciate the fish once the first stage has identified the echo to be that from a fish; and finally in the third stage, to identify the fish angle after the second stage has identified the fish to be a sturgeon. The three stages have 2,4 and 3 neurons respectively in the output layer.

The performance of three layered networks on these subproblems are shown in Table 1 as case B, case C and case D respectively. The best performances in the three stages are 94.4%, 97.5% and 100% respectively. A combination of the three stages scores 93.3% on the original test set. It is interesting to note in all the cases, the performance increases as the hidden layer is increased to a certain size and then begins to drop.

TABLE 1 - Success Rates (in percentages) on the Test Set

Middle layer neurons	10	15	20	25	30
A: All 9 targets	77	81	86	91	86
B: Fish/Debris	93	93	94	94	94
C: Fish Type	93	98	98	95	95
D: Fish Angle	94	97	100	97	97
E: Best of B+C+D			93		

3.4 Experiment 3

The third experiment consisted of accepting only unambiguous responses from the network. In the above cases, the network is forced to come up with a guess for each input signal. Typically, there are m neurons in the output layer to identify m classes. The ideal outputs correspond to corners on an m-dimensional hypercube and diagonal hyperplanes are used to divide the space into m regions corresponding to the m classes. Figure 2(a) illustrates the case for m=2. It is perhaps more logical to allow the network to respond "not sure" when the distance r (suitably defined) from the actual output to the ideal outputs are all greater than a preset value. Figures 2(b) and 2(c) illustrate the idea when the maximum distance allowed, r, is based on the Euclidean norm and the max norm respectively. Clearly, as r decreases, the confidence in the response increases at the expense of an increased number of "not sure" responses. Note that, to ensure that two regions of interest do not intersect, r should be less than r_0, where r_0 is half the distance between two ideal outputs.

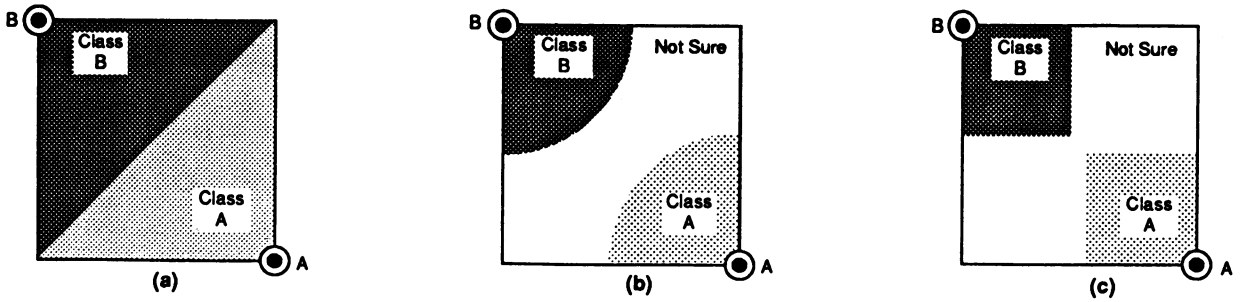


FIGURE 2 - ALLOWING "NOT SURE" RESPONSES

Table 2 shows the performances of the nets in the above two experiments with $r = 0.9r_0$ (max norm). In the table, the success rates are based on the unambiguous responses alone. As expected, the accuracy of the net increases (comparing the corresponding cases in Tables 1 and 2) and the best performing net scores over 96 percent in the test set. It was found that (not shown in the table) the accuracy could be increased to 100 percent, if one is willing to accept a 35 percent "not sure" rate. This may indeed be the case in fisheries, since all the fish in a school tend to be of the same species and thus not all the echoes need be identified.

TABLE 2 - Success Rates when "Not-Sure" Response (Percentages in Brackets) is Allowed

Middle layer neurons	10	15	20	25	30
A: All 9 targets	80 (10)	85 (12)	87 (9)	96 (17)	90 (16)
B: Fish/Debris	93 (1)	93 (1)	95 (3)	95 (3)	94 (1)
C: Fish Type	97 (13)	100 (10)	100 (13)	100 (13)	100 (15)
D: Fish Angle	100 (13)	100 (13)	100 (7)	100 (10)	100 (3)
E: Best of B+C+D			96 (6)		

4. CONCLUSION

This paper has described some preliminary work on the use of a neural network, based on the back propagation algorithm, for the detection and classification of fish from sonar echoes. The results on the laboratory data are quite encouraging. The success using only the time domain data is somewhat surprising and is a tribute to the wide band detection module used. However, the results are preliminary and further studies are required. Studies involving fish at other different angles to the transducer, multiple target analysis, as well as evaluations of other network architectures are in progress. Studies using power spectral methods are also under way.

ACKNOWLEDGMENTS

Gerry Hunt developed the Wideband Sonar Detection Module. Blair Sim made significant technical contributions which made the data collection system both feasible and efficient. Special thanks to Scott Poulton and Ernest Fischer for preprocessing the analog data for digitization.

REFERENCES

Gorman, R.P. and T.J. Sejnowski. (1988). Learned classification of sonar targets using a massively parallel network. *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. 36, pp. 1135-1140.

Roitblat, H.L., Moore, P.W., Nachtigall, P.E., Penner, R.H. and Au., W.W.L. (1989). Dolphin echolocation: identification of returning echoes using a counterpropagation network. *Proceedings of the International Joint Conference on Neural Networks*, June 1989, Vol. I, pp. 295-300.

Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986). Learning internal representations by error propagation. In D.E. Rumelhart and J.L. McClelland (Eds.) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Cambridge: MIT Press, pp. 318-362.

VECTOR PAIR CORRESPONDENCE BY A SIMPLIFIED COUNTER-PROPAGATION MODEL: A TWIN TOPOGRAPHIC MAP

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. A modified and simplified version of the Counter-Propagation Network is proposed to self-organize all the possible vector pair correspondences of two vector sets onto two counterfaced topographic maps (together called a twin topographic map) and furthermore to find a combination of one-to-one correspondences between vectors of the two sets with the distances between each pair minimized.

1. Introduction. By combining a portion of the topographic self-organizing map of Kohonen and the outstar structure of Grossberg, Hecht-Nielsen [1] proposed a new type of mapping neural network (Counter-Propagation Network or CPN for short) which functions as a statistically optimal look-up table and could be applied to solve problems of pattern recognition, function approximation, statistical analysis, and data compression [2]. In this paper, with some simplification and modification of the CPN, a new model, called a twin topographic map, is proposed to explore the potential possibilities of extending the CPN as well as the topographic map to some matching problems. Such problems are encountered in structural pattern recognition, especially in computer vision.

As a preliminary stage, here we mainly discuss the problem of vector pair correspondence. Given two sets of vectors $x_1, x_2, \ldots, x_n$ and $y_1, y_2, \ldots, y_m$, with $m \geq n$, the purpose of vector pair correspondence is to find a combination of vector pairs (e.g., n pairs) with one vector x_i corresponding to one vector y_j under some specific criterion such as the minimum distance between each pair or the minimal total distance of n pairs. A computational solution would be to at first construct a list of all the possible vector pairs with corresponding mutual distances, and then based on the list, to select the desired combination under the given criterion.

In the following section, with some modifications and simplifications on the CPN model, a twin topographic map is proposed consisting of two counterfaced topographic maps in which the counter-cooperative competing method of the CPN is used to search for a best-matching unit on this twin map. By randomly choosing pairs (x_i, y_j) as input, all the possible vector pair correspondences of the two vector sets can be self-organized onto the twin topographic map. Furthermore, based on these ordered pairs, it is possible to obtain a combination of one-to-one correspondences between vectors of the two sets with the distance between every pair of vectors minimized. In the last section, simulation examples are given.

2. Vector Pair Correspondence By A Simplified CPN Model. A Twin Topographic Map is defined as follows. As shown in Fig.1, two Kohonen topographic self-organizing layers are counterfaced, the lattices of both having the same structure and the same number of units with the same dimensional weight vectors. For convenience, one lattice is denoted U-lattice with the weight vector of its $i-$th unit denoted by u_i, and the

This work was supported by Tekes Grant 4196/1988 under Finsoft project

other lattice is denoted V-lattice with the weight vector of its $i-$th unit denoted by v_i. We call such a system a twin topographic map and call a pair of corresponding units (u_i, v_i) the $i-$th twin unit t_i.

In order to self-organize all the possible pairs (x_i, y_j) of the two sets $x_1, x_2, \ldots, x_n$ and $y_1, y_2, \ldots, y_m$ on such a twin map, vector pairs are repeatedly input to train the map. Each pair (x, y) is randomly chosen from the two sets with x being drawn with equal probabilities from $x_1, x_2, \ldots, x_n$ and y likewise from $y_1, y_2, \ldots, y_m$. Vector x is input to the U-lattice and y to the V-lattice, and two steps are implemented. One is a cooperative-competitive step, the other is a weight vector updating step.

The Cooperative-Competitive Step: The method developed in the CPN model is almost directly used here. The twin units t_i will cooperatively compete for the input pair (x, y) and a unit t_s will be the best matching twin unit (denoted $\tau_s = 1$) under the following condition:

$$\tau_s = \begin{cases} 1 & \text{if } I_s < I_i, \forall i; \\ 0 & \text{otherwise,} \end{cases} \tag{1.a}$$

$$I_i = \|x - u_i\|^2 + \|y - v_i\|^2. \tag{1.b}$$

The Euclidean distance is used like in the original form of the topographic map [3]. Such a counter-cooperative competing method has an importance role in catching all the possible vector correspondences which are implicitly contained in the input data.

The Weight Vector Updating Step: After an input pair (x, y) has found its best matching twin unit t_s, the two weight vectors u_s, v_s of this twin unit will be updated by a topographic learning rule which has some difference both from that used in the CPN model and that used in the topographic self-organizing map. In the CPN only the best matching unit itself is updated. We do not consider this case here. In the Kohonen map, since an organized topological map is expected, the best matching unit as well as the units within its certain symmetrical neighborhood are updated. Although the latter updating rule will be suitable for the twin topographic map, too, in this case a poorly ordered map is obtained when the neighborhood has a symmetrical form. The solution is to let the two lattices become ordered in a way as shown in Fig.2, i.e., each vector is ordered in one row (or column) and similar vectors are ordered in nearby rows (or columns). To this end, we use a neighborhood of rectangular structure as shown in Fig.3. The neighborhood is placed horizontally on one lattice and vertically on the other lattice.

Finding a Minimum Distance Combination: In order to obtain on the ordered map a combination of one-to-one correspondences between vectors of the two sets with the distance between every pair minimized, the U- and V-lattices are labelled in the following way:

(1) Sequentally re-input each vector of the set $x_1, x_2, \ldots, x_n$ into the U-lattice, and for each x_i, find the unit k on the U-lattice with

$$\alpha_k < \alpha_j, \forall j, \alpha_j = \|x_i - u_j\|^2 + \|u_j - v_j\|^2 \tag{2.a}$$

and label the unit by x_i (or simply i).

(2) Then sequentally re-input each vector of the set $y_1, y_2, \ldots, y_m$ into the V-lattice, and for each y_q, find the unit p on the V-lattice with

$$\beta_p < \beta_r, \forall r, \beta_r = \|y_q - v_r\|^2 + \|v_r - u_r\|^2 \qquad (2.b)$$

and label the unit by y_q or simply q.

(3) Label a twin unit by (x_i, y_j) (or i, j) if its unit on the U-lattice has label x_i and its unit on the V-lattice has label y_j; otherwise, label the twin unit by $(0, 0)$.

After labeling, the nonzero (x_i, y_j) labels constitute a solution of vector pair correspondences since they indicate a combination of several vector pairs (e.g., n pairs) with the minimum distance between each pair. For the case of $m = n$, if the number of such (x_i, y_j) is just n and there are no more than one pair having the same x_i or y_j, then the resulted n pair combination is a solution of the so-called bipartite matching problem. Computer simulations have shown that a good solution or even the optimal solution could be obtained for a number of examples of the bipartite matching problem.

3. Simulation Examples. Due to the limited space, only examples on one data set are illustrated here. This is a two dimensional data set with $m = n = 8$. The eight x vectors are $(0,0)$, $(0.5,0)$, $(1,0)$, $(0.95,0.5)$, $(1,1)$, $(0.5,0.98)$, $(0,1)$, $(0,0.5)$ and the eight y vectors are $(0,0)$, $(0.53,0.1)$, $(1,0.05)$, $(0.98,0.5)$, $(1.1,1.1)$, $(0.5,1)$, $(0.1,0.94)$, $(0.1,0.5)$. The parameters used in self-organization are as follows: the gain factor (α in [3]) is initially 0.5 and in the first 2000 steps linearly reduces to 0.1, after which it linearly reduces to zero during further 20000 steps. The structure of neighborhood given in Fig.3 is used with the ratio of the two sides 2 : 1. The initial neighborhood size N_c of [3] is 3 and in the first 2000 steps it reduces to 1 and then remains unchanged at 1.

By randomly choosing pairs (x_i, y_j) to train the twin map, it was found that after 2000 steps, 60 pairs (from the total of 64) were already ordered on the twin map and the optimal bipartite matching solution was obtained as shown in Fig.4a. Furthermore, after 5000 steps, the ordered list of all the possible pair correspondences was obtained as shown in Fig.4b. Note that the order is quite near to the desired form given in Fig.2.

Experiments on symmetrical square shaped neighborhoods were also conducted using the same parameter set. A result is shown in Fig.5. After 12000 steps, the twin map was still poorly ordered with only a few pair correspondences being learned, and only 5 minimized distance pairs of correspondences were obtained. Experiments using the original CPN method, in which only the best matching twin unit was updated, were also conducted and the result is given in Fig.6. The result is quite bad; after 15000 steps the results both on vector pair ordering and the minimum distance pair correspondences were much worse than those in Fig.5.

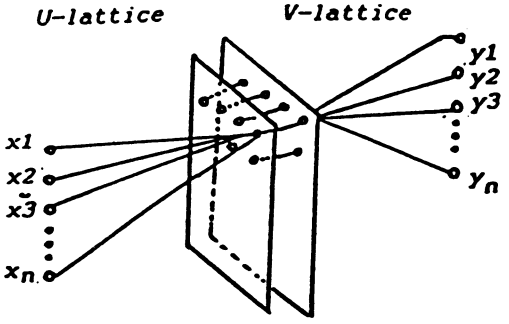
REFERENCES

[1]. R.Hecht-Nielsen, **Applied Optics**, Vol.26, No.23, pp 4979 - 4984.

[2]. R.Hecht-Nielsen, **Neural Networks**, Vol.1, pp 131 - 139, 1988.

[3]. T. Kohonen, *Self-Organization and Associative Memory*. Springer Verlag, Berlin, 1988.

[4]. L. Xu and E.Oja, "Adding Top-Down Expectation Into The Learning Procedure of Self-Organizing Maps", to be published.



(two Kohonen map counter faced)

Fig.1 A Twin Kohonen Map

```
x1 x2 x3 x4 x5 x6      y1 y1 y1 y1 y1 y1
x1 x2 x3 x4 x5 x6      y2 y2 y2 y2 y2 y2
x1 x2 x3 x4 x5 x6      y3 y3 y3 y3 y3 y3
x1 x2 x3 x4 x5 x6      y4 y4 y4 y4 y4 y4
x1 x2 x3 x4 x5 x6      y5 y5 y5 y5 y5 y5
x1 x2 x3 x4 x5 x6      y6 y6 y6 y6 y6 y6
U-lattice              V-lattice
```

Fig.2 The Desired Form of Ordered List

```
                Nc                          Nc/2
x1 x2    x4 x5 x6          y1 y1 y1 y1 y1 y1
x1 x2 x3 x4 x5 x6          y2 y2 y2 y2 y2 y2
x1 x2 x3 x4 x5 x6    Nc    y3 y3 y3 y3 y3 y3
x1 x2 x3 x4 x5 x6          y4 y4 y4 y4 y4 y4
x1 x2 x3 x4 x5 x6          y5 y5 y5 y5 y5 y5
x1 x2 x3 x4 x5 x6          y6 y6 y6 y6 y6 y6
U-lattice                  V-lattice
```

Nc/2

Fig.3 The Structure of Neighborhood

```
0 0 4 0 0 0 0 0   0 0 4 0 0 0 0 0      5 6 4 3 2 1 8 7   4 4 4 4 4 4 4 4
0 0 0 3 0 0 0 0   0 0 0 3 0 0 0 0      5 6 4 3 2 1 8 7   3 3 3 3 3 3 3 3
0 0 0 0 2 0 0 0   0 0 0 0 2 0 0 0      5 6 4 3 2 1 8 7   2 2 2 2 2 2 2 2
0 0 0 0 0 1 0 0   0 0 0 0 0 0 1 0 0    5 6 4 3 2 1 8 7   1 1 1 1 1 1 1 1
0 0 0 0 0 0 8 0   0 0 0 0 0 0 0 8 0    6 5 4 3 2 1 8 7   8 8 8 8 8 8 8 8
0 0 0 0 0 0 0 7   0 0 0 0 0 0 0 0 7    6 5 4 3 2 1 8 7   7 7 7 7 7 7 7 7
6 0 0 0 0 0 0 0   6 0 0 0 0 0 0 0 0    6 5 4 3 2 1 8 7   6 6 6 6 6 6 6 6
0 5 0 0 0 0 0 0   0 5 0 0 0 0 0 0 0    6 5 4 3 2 1 8 7   5 5 5 5 5 5 5 5
  U-lattice         V-lattice              U-lattice         V-lattice
      after 2000 steps                         after 5000 steps
a. The Minimum Distance Pairs              b. The Ordered List
```

Fig.4 The Result With Neighborhood of The Structure in Fig.3

```
7 0 0 0 0 0 0 5   7 0 0 0 0 0 0 5      6 6 7 6 6 6 6 6   7 8 1 2 3 3 4 5
0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0      6 6 6 6 6 6 6 6   7 8 1 2 2 3 4 5
0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0      5 5 5 6 7 7 7 7   7 8 8 2 3 3 4 5
0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0      4 4 4 6 8 8 8 8   7 8 1 2 3 3 4 5
0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0      4 4 4 2 1 1 1 1   6 4 2 3 3 3 4 5
0 0 4 3 0 0 0 0   0 0 4 3 0 0 0 0      4 4 4 2 2 1 1 1   5 4 3 3 3 3 6 6
0 0 0 0 2 0 0 0   0 0 0 0 2 0 0 0      3 3 3 3 2 1 8 8   6 6 2 2 2 2 8 7
0 0 0 0 0 8 0 0   0 0 0 0 0 1 8 0      2 2 3 3 2 1 8 8   6 7 8 1 1 1 8 7
  U-lattice         V-lattice              U-lattice         V-lattice
a. The Minimum Distance Pairs              b. The Ordered List
```

Fig.5 The Result With Square Structure Neighborhood After 12000 Steps

```
0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0      6 3 4 6 4 4 5 4   2 1 4 4 6 4 7 6
0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0      4 5 6 6 4 4 4 4   6 3 6 4 4 4 4 6
0 0 0 0 0 0 0 0   0 0 0 0 0 0 0 0      6 4 4 6 1 4 6 6   4 2 4 4 5 6 6 4
0 0 0 0 0 0 0 0   0 0 0 0 0 0 6 0 0    4 6 4 4 6 4 2 7   4 6 6 4 4 6 7 1
0 0 0 0 0 0 0 6   0 0 0 0 0 0 0 0 7    4 3 4 6 4 4 4 6   4 7 6 4 4 8 4 7
0 0 0 0 0 3 0 8   0 0 0 0 3 0 0 8      2 4 8 4 2 4 4 1   5 4 3 4 3 3 4 4 1
0 0 5 0 0 0 4 0   0 0 5 0 0 0 4 0      4 4 5 4 7 4 4 4   6 1 5 4 5 6 3 4
0 0 0 0 0 0 7 0   0 0 0 0 0 0 0 0      7 4 1 4 4 4 8 4   3 4 3 4 4 4 6 4
  U-lattice         V-lattice              U-lattice         V-lattice
a. The Minimum Distance Pairs              b. The Ordered List
```

Fig.6 The Result With Neighborhood Nc=0 after 15000 steps

A SPECIAL PURPOSE NEURAL NETWORK FOR SCHEDULING SATELLITE BROADCASTING TIMES[†]

P. Bourret[*+] F. Remy[#] S. Goodall[*]

[*]Department of Computer Science
University of Maryland
College Park, MD 20742

[+]ONERA-CERT/DERI
2 Avenue E. Belin
BP 4025
31055 Toulouse CEDEX
France

[#]ENSAE
10 Avenue E. Belin
31055 Toulouse CEDEX
France

In a previous paper [Bour89] we have shown that optimization problems can be solved by neural networks using the competitive activation mechanism [Regg87], instead of the usual Hopfield model. The application studied was the scheduling of low level satellite broadcasting times for a given set of antennas. Several drawbacks arose in simulating some of the scenarios. In this paper, after a short presentation of the application, we describe the drawbacks encountered and present a new implementation with a modified activation rule which give amazingly good results in each case in which it has been tested.

I. The Application

Low level satellites gather information during each revolution around the Earth. Since they have a limited amount of memory they need to send this information to a ground station, or antenna. This can only be done when the satellite is in sight of an idle antenna. Each satellite has a priority level which is related to the importance of the information it has gathered. In addition, there is a certain minimum broadcast time which the satellite will need to send its stored data. Thus, given the required broadcasting time, the priority level of each satellite and the time intervals within which the satellites are in sight of the various antennas, the problem consists of optimizing the total broadcasting time weighted by the priority of satellites. This optimization must be done with respect to the following constraints:

- Only one satellite can be assigned to a given antenna at a given time.
- The required broadcasting time of a satellite may be split between several antennas.
- The actual broadcasting time of a satellite is no more than its required broadcast time.
- A satellite cannot send its information to several antennas during the same time interval.

II. Notation

Let $S_\alpha, S_\beta, \ldots$ be the satellites, $A_a, A_b, \ldots$ be the antennas.
Let U_α be the required broadcasting time of S_α, and p_α be the priority of S_α.
Let $[T_a^\alpha, T'^\alpha_a]$ be the time interval within which S_α is in sight of A_a.
Let $[(t_a^{i\alpha}, \tau'^{i\alpha}_a)]$ be the time slice assigned in the solution to the ith broadcasting of S_α for a given antenna. Then the optimization function is $\sum_\alpha p_\alpha Min[\sum_i \sum_a (t'^{i\alpha}_a - t_a^{i\alpha}), U_\alpha]$.

III. The previous solution and its drawbacks

We have proven in [Bour89] that, for s, a given total number of divisions of the satellite broadcasting times, there is an optimal solution such that the time slices $(\tau_a^{i\alpha}, \tau'^{i\alpha}_a)$ of the solution are time slices resulting from the s divisions of the satellite broadcasing times (the set σ_s). Therefore the problem amounts to assigning the time slices of the set σ_s to the satellites. To perform this assignment, we defined a three layer network. A competitive activation mechanism was used between two of the layers. The central layer consists of R nodes in which $R^i_{a\alpha}$ represents the

[†]Acknowledgements: Supported in part by NASA Award NAGI-885 and ONERA-CERT.

assignment of satellite α to an antenna a, during the time slice i. Once the network has stabilized, the nodes of the R layer at or near unity represent the solution set; the assignment of the antennas to these satellites in the time slices indicated represents the network solution.

The R nodes of the same antenna and time slice compete for activity from a node of the C layer; there is a node in the C layer for each antenna–time slice pairing for which multiple satellites are competing. The nodes of the T layer keep track of the amount of antenna time assigned to a particular satellite (there is one T node for each satellite). The activation of a T node is then used to modify the weights on the competitive connections between the R and C node layers. Initially, the weights on these connections are set to the priority of the R node's satellite. As the total assigned time for a satellite increases towards its required broadcasting time, the weights on the connections between the corresponding R nodes (of the satellite) and C nodes decreases towards zero.

The solutions with the previous network were quite good, especially for those situations in which there was no solution which could satisfy all of the requests; in such a situation, the optimal solution was reached. But two drawbacks surfaced. First, with this method, an interpretation of the observed activity level towards which the system converges is needed, since the activity level could be any real number between zero and one. An antenna is assigned during a time slice to the satellite with the most active R node among the time slice competitors. In some instances, not all of the R nodes have stabilized to their maximum or minimum activation level, so there is no clear–cut winner and interpretation is difficult.

Secondly, in this method the interpretation is quite impossible when there are several equivalent optimal solutions. In this case the activity level is divided between R nodes which belong to one of the optimal solutions. Sometimes, when there are many equivalent optimal solutions, an R node which does not belong to one of the optimal solutions wins, producing a non–optimal solution.

IV. The Revised Implementation

The basic equation of the competitive activation mechanism is as follows. Let $a_i(t)$ be the activation level of node i at time t and let $In_i(t)$ be the summation of the inputs to node i for this time interval. Then:

$$\frac{da_i(t)}{dt} = [In_i(t)-a_i(t)][1-a_i(t)]$$

with $In_i(t)=\sum_j out_{i,j}(t)$; $out_{i,j}(t)$ stands for the output from node j to node i. Let w_{ij} be the strength of the link from node j to node i. In our previous network, activation from node j was *competitively* distributed to node i and its neighbors using:

$$out_{i,j}(t)= \frac{w_{i,j}a_i(t)a_j(t)}{\sum_k w_{k,j}a_k(t)}$$

Our modification to the implementation has been to use the following activation output functions. The output from the C layer to the R layer nodes competing for a time slice on a particular antenna is competitively distributed according to

$$out_{i,j}(t)= w_{i,j}(t)a_i(t)a_j(t)(1 - \sum_k a_k)$$

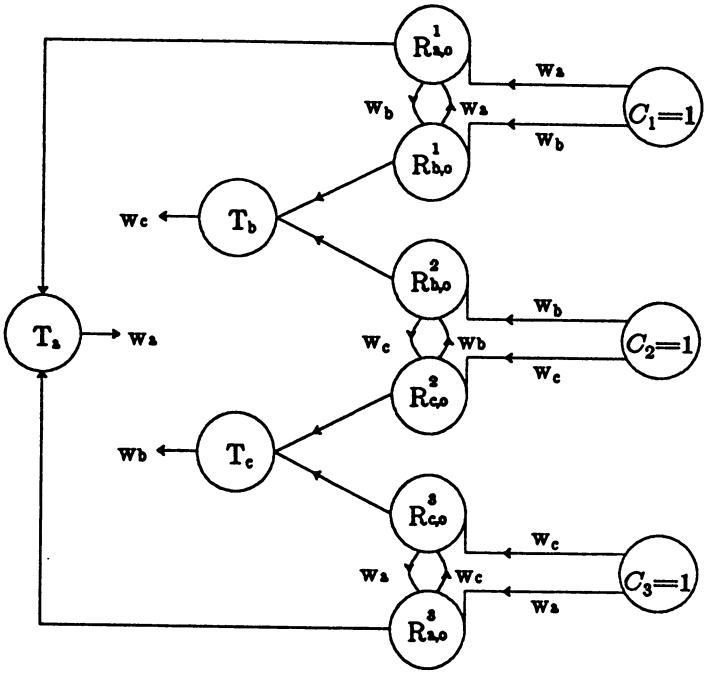
where k ranges over the competing R layer nodes. The output among competing R layer nodes is $out_{i,j}(t)= w_{i,j}(t)a_i(t)a_j(t)$. Along with these new output functions, we have modified the activation update rule for R layer nodes to be

$$\frac{da_i(t)}{dt} = [In_i(t) - Out_i(t) - K(1-a_i(t))a_i(t)][1-a_i(t)]$$

where $Out_i(t)$ is the sum of the outputs from node i. The C layer nodes always remain at the activation level 1.0.

The activation update function for the nodes of the T layer has also been modified. Let $r^i_{s\alpha}(t)$ be the activity level of $R^i_{s\alpha}$. $T_s(t) = \sum_\alpha \sum_i p_s r^i_{s\alpha}(t)$ is the activation level of the T node T_s at time t. The weight modification function remains the same: $w_s(t) = p_s\left[1 - \frac{T_s(t)}{U_s}\right]$.

Our network is illustrated in figure, which shows an example with three satellites a,b,c, one antenna o, and three time slices 1,2,3. The two satellites a and b compete for time slice 1, b and c compete for time slice 2 and satellites a and c compete for time slice 3. In the figure, the nodes with the number 1 in the circle are nodes with a constant activity level which equals 1.



The differences between the original and the new versions described here are the following:
a) A new competitive output function distributes C layer activation among R layer competitors.
b) Links have been added between R nodes which compete with each other.
c) Decay has been added to the activation rule for the nodes of the R layer. Each R node $R^i_{s\alpha}$ decays $K(1-r^i_{s\alpha})r^i_{s\alpha}$ at each time step, where $r^i_{s\alpha}$ is the activation level of the node $R^i_{s\alpha}$, and K is a small constant (10^{-4}). The decay factor takes some activity from each of the R nodes at each time step; more activity is drained from the R nodes with low activation values than from those with activities close to the maximum. Thus, when a solution is possible, the decay, in combination with the other R node inputs, produces a stable network with all R nodes at or near either their maximum or minimum activation values (1.0 and 0.0, respectively).
d) Noise has been added to the activation update rule for the R nodes. At time $t+dt$, the new activation value of node a_i is

$$a_i(t+dt) = a_i(t) + \delta\frac{da_i(t)}{dt}[0.999999 + 10^{-6}\sin(10000a_i(t)R]$$

where R is the number of nodes in the R layer. The noise factor breaks the symmetry given by totally equivalent solutions.

V. An Example

We have simulated the first example in [Bour89] with the new network formulation to observe the differences between the two methods. In the example, four satellites compete for nine time slices:

Time Slice	0–1	1–2	2–3	3–4	4–5	5–6	6–7	7–8	8–9
Competing Satellites	a b c	a b d	b c d	a b	a c	a c	b c	b d	c d

With durations $U_a=5$, $U_b=2$, $U_c=1$ and $U_d=1$ and priorities $p_a=1$, $p_b=2$, $p_c=3$ and $p_d=4$. The initial method produced the following results:

Time Slice	0–1	1–2	2–3	3–4	4–5	5–6	6–7	7–8	8–9
a	0.818	0.850	*	0.885	0.868	0.868	*	*	*
b	0.072	0.085	0.264	0.115	*	*	0.666	0.709	*
c	0.109	*	0.390	*	0.131	0.131	0.333	*	0.492
d	*	0.065	0.346	*	*	*	*	0.065	0.346

The new method gives these results:

Time Slice	0–1	1–2	2–3	3–4	4–5	5–6	6–7	7–8	8–9
a	0.999	0.999	*	0.999	0.999	0.999	*	*	*
b	0	0	0.999	0	*	*	0.999	0	*
c	0	*	0	*	0	0	0	*	0.999
d	*	0	0	*	*	*	*	0.999	0

With our new method, the network stabilizes with all R nodes near their maximum or minimum activation level. Those activations near 1.0 represent assignments of satellites to antennas for a particular time slice. The lower table of result shows that the system solution assigns each satellite its requested broadcasting time, using the given time slices.

VI. Conclusions

Several drawbacks were identified with our original solution to this problem. Modifications were made to the network and the activation functions, which kept the activation mechanism competitive. The addition of a decay factor forced the network to make a decision since nodes were driven to their maximum or minimum values. As the above example illustrates, the modified system avoids the previous drawbacks and gives definitive results for this scheduling optimization problem.

[Bour89] Bourret, P., S. Goodall, and M. Samuelides. "Optimal Scheduling by Competitive Activation: Application to the Satellite–Antenna Scheduling Problem," *Proc. IJCNN*, I:565–572, 1989.

[Regg87] Reggia, J. "Properties of a Competition Based Activation Mechanism in Neuromimetic Networks," *Proceedings First International Conf. on Neural Networks*, II:131–138, 1987.

DASA/LARS, A LARGE DIAGNOSTIC SYSTEM USING NEURAL NETWORKS

Fred Casselman
GTE Government Systems
Command, Control and Communications Systems
77 "A" Street
Needham, MA 02194

Major Jody DeJonghe Acres
Defense Communications Agency
Defense Communications Engineering Center (DCEC)
1860 Wiehle Avenue
Reston, VA 22090

This paper describes the DASA/LARS, a large system making extensive use of neural networks to perform a number of diagnoses on satellite communication networks. The system was developed by GTE under contract DCA100-88-C-0063 to DCEC, and is currently being used operationally at the DSCS OC (Defense Satellite Communications System, Operations Center) at Ft. Detrick, MD.

DCEC's interest in neural networks was based on a desire to automatically detect anomalies that cannot be detected with current conventional systems. Neural networks were preferred over expert systems because the networks are trained from examples rather than defined by rules. Furthermore, neural networks have the ability to generalize, identifying problems that are not specifically included in the training.

The goal for the DASA/LARS application of neural networks was to determine the usefulness of neural networks using live sensor information in an operational environment. As an added benefit, if the prototype was successful, it could be left in place with an immediate operational capability. The secondary goal was to learn about how to apply neural networks and what applications they are best suited for. This information is very valuable for future applications.

DASA/LARS SYSTEM. Currently, the monitoring and diagnosis at the OC's are accomplished in part by the DSCS Automated Spectrum Analyzer (DASA). DASA is a conventional spectrum analyzer system that was developed and fielded employing 1970's technology. The system is used to diagnose spectrum anomalies associated with the Frequency Division Multiple Access (FDMA) satellite links and requires extensive operator involvement. The DASA/LARS is a DASA-like system using neural network technology to extend and automate the diagnoses. The system was designed using GTE's proprietary LARS™ environment for designing and training large back propagation type ANNs (artificial neural networks).

The diagnoses are based on two inputs: data obtained from a swept-frequency spectrum analyzer and database information obtained from another OC subsystem, the DOSS computer. See Figure 1. The spectrum analyzer monitors the satellite downlink in such a manner that all the FDMA carriers transmitting through the satellite may be observed. The DOSS database provides a listing of all the authorized carriers and their planned parameters. DASA/LARS diagnoses problems by comparing the observed spectral data with the planned parameters.

DASA/LARS uses nine different ANNs to diagnose a total of thirteen different problems. See Figure 2. Data format problems such as wrong modulation, wrong coding, and wrong SENU (transmit filter) result from operator error at a satellite earth station. Data transmitted from these earth stations may not be usable or may interfere with data transmitted from other earth stations. Problems such as saturated transponder occur when the transponder operates in a non-linear region due to excessive power demands. This can cause severe degradation of service to the users and

often results from one or more earth stations inadvertently increasing their power beyond authorized limits. Autotrack failure is a type of earth station failure which will degrade data transmitted from and received by that earth station.

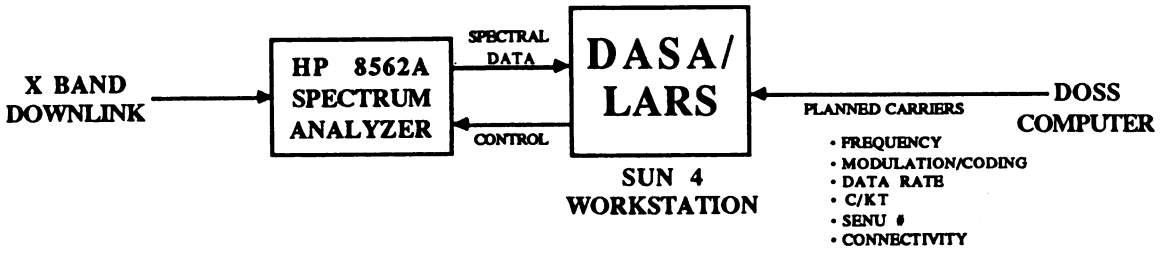


FIGURE 1: DASA/LARS INTERFACES

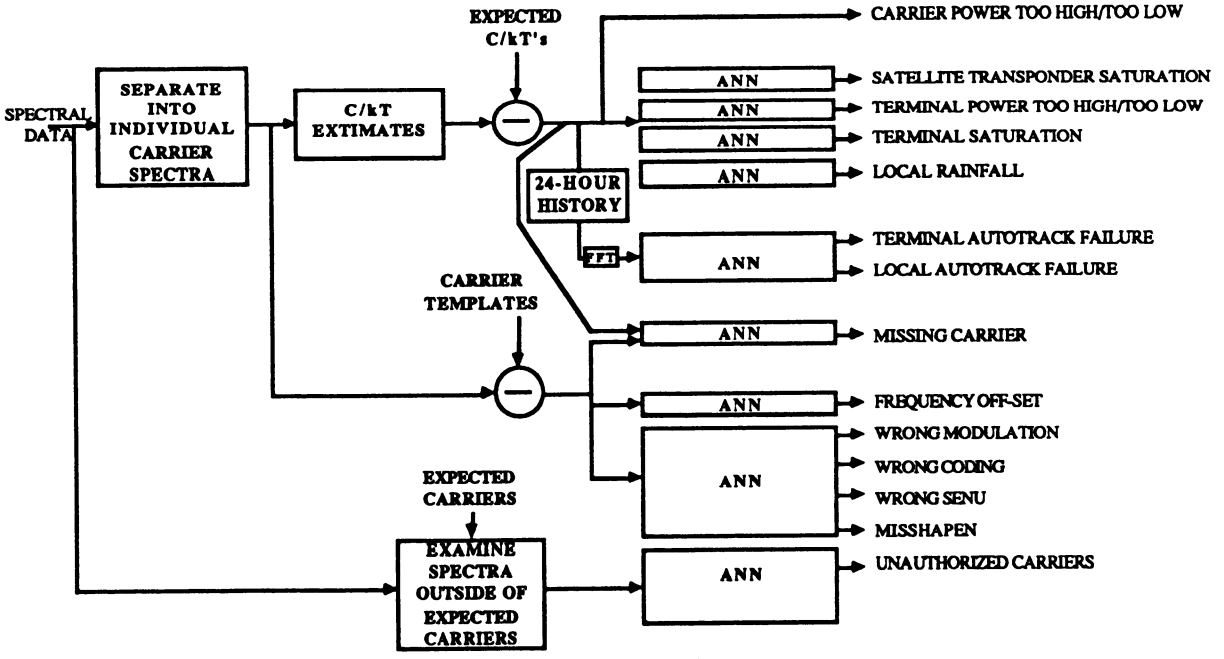


FIGURE 2: APPLICATION OF ANN'S IN DASA/LARS

DESIGN AND TRAINING. The challenge was to employ neural network technology to the maximum extent reasonable to diagnose the satellite communication problems. The symptoms for many of these problems were not well understood at the start. In order to characterize the symptoms and develop training examples, extensive recordings were made at the ITF (Integrated Test Facility) at Fort Monmouth, NJ. The ITF has a satellite simulator as well as a large population of earth station equipment. Most of the end problems of interest were simulated at the ITF and subsequently studied extensively in order to define the features which would categorize each problem into a distinct class. The following paragraphs describe the design and training approaches used for three of the more interesting diagnostic features.

Saturated Transponder. This feature diagnoses the onset of transponder saturation. The examples of transponder saturation simulated and recorded at the ITF were initially studied to see which phenomenon might be used for the diagnosis. Of several possible phenomena, the only one consistently present was the occurrence of one or more carriers whose power was higher than predicted, causing the remaining carriers to be suppressed (or attenuated). The closer the transponder was to saturation, the more pronounced this phenomenon tended to become. The resulting design first measured the carrier levels in terms of C/kT (ratio of carrier power to noise power density), compared these measurements with expected levels, and used this difference data to train a neural network. The network was trained to recognize four different levels of saturation, ranging from an early warning indication to total saturation . By recognizing the onset of saturation, the offending carrier(s) can be identified and the problem corrected before performance is affected.

In addition to training on the simulated data, the training set was supplemented with hand constructed counter-examples of cases likely to cause misdiagnosis. The concern was that some combinations of rainfall (which attenuates all carriers) and permissible carrier power variations might mimic symptoms of saturation. The examples were easily generated by applying common sense reasoning. This demonstrated the ease by which a neural network can often be molded and manipulated to perform as desired. The saturated transponder design used a 4-layer network and a total of 131 training examples.

Data Format Problems. This category of faults pertains to individual carriers and includes wrong modulation, wrong coding, and wrong SENU (transmit filter). This is a complex diagnostic task since each carrier has a unique spectral shape. The first step in the diagnosis is to normalize each carrier to a common bandwidth by applying a spline interpolation algorithm. Next, theoretical templates are constructed for each carrier for the nominal case and for all the possible data format faults. The parameters obtained from the DOSS data base are used to construct these templates. The nominal template is subtracted from the carrier and the difference is fed into the neural network which decides if this is a good fit. If not, all the fault templates are subtracted from the carrier and the differences are fed into the neural network, one at a time. The network determines which, if any, of these templates provides a good fit. If the neural network does not recognize any of the conditions, a diagnosis of "misshapen" will be reported. The design for the neural network to detect data format faults underwent the greatest change during the project. It was originally thought that it would be possible to use some average of carriers measured at the ITF as the basis for template formation. Experience demonstrated that this gave inadequate performance, though it was possible to compensate for some of the distortions by a complex adjustment procedure. In the end, it was decided that nothing short of a complete theoretical calculation of templates would suffice, and this was the approach finally adopted.

At first glance, finding the degree to which templates fit may appear to be a trivial problem, not requiring the power of a neural network. However, it turns out that actual carriers have certain regular types of minor deviations from theoretical. These deviations were easily accommodated by modeling the deviations in the training examples and also by including simulated and actual carriers in the training set. The data format design used a 4-layer network and a total of about 500 training examples.

The sensitivity of the design may be adjusted by the selection of the training examples. For instance, if one uses a training set with large deviations as opposed to small deviations, there will be (1) an increased occurrence of declaring faulty carriers as normal, (2) a decrease in false alarms for normal carriers that have substantial irregularities, and (3) a decrease in the occurrence of the "misshapen" diagnosis.

Autotrack Failure. A failure of an earth station's autotrack feature will result in failure of the antenna to track diurnal variation due to orbital incline. This will result in significant losses during

certain times of the day, and produces a very predictable carrier level pattern over a 24 hour period. Collecting autotrack failure data would have been time consuming and would have disrupted the normal operation of the system. The design, therefore, was based on purely theoretical training examples. A mathematical description of the orbital mechanics yielded training examples for the various types of periodic power variation patterns to be expected. Preprocessing by an FFT proved successful in isolating the periodic components. The design used a 4-layer network and a total of 17 training examples.

TEST AND EVALUATION. Four days of testing to a DCEC test plan was performed on the system at the ITF in November 1988. As a result of this testing, design updates were made and a final four days of testing was conducted at the ITF in April 1989. All diagnostic features appeared to work correctly, with a few exceptions where it had not been possible, within the limited time span of the effort, to adequately model, simulate, or otherwise characterize the fault.

The system was then installed at the satellite operations center (OC) at Fort Detrick. MD. It was operated for a period and appeared to be performing exactly as it had at the ITF. A plan is presently being formulated for performing operational testing, following which the system will be turned over to the operators for their use. The system was designed at the outset as an operational system and as such includes extensive user friendly features. A complete user's manual has been produced which describes the theory and operation of the system.

DASA/LARS ENHANCEMENTS. It became obvious during design and development that the neural network approach could easily deal with a yet richer information source. Richer information would enhance the quality of the diagnoses and extend the diagnoses to a larger class of problems. One candidate source is an enhanced spectrum analyzer which would provide phase information in addition to amplitude information. Vector analyzers of this sort are available off the shelf. Another candidate source is data from other OC subsystems. Some of these subsystems have information available directly from the satellite which could greatly strengthen the diagnoses.

Another enhancement would be to utilize an expert system following the neural network diagnoses. By this means, the operator could be presented with specific recommendations for corrective action rather than a raw diagnosis. This enhancement is in accord with the goals of increasing the effectiveness of satellite network control while reducing staffing requirements. Future upgrades will consider these enhancements.

CONCLUSION. The conclusion is that back propagation ANN technology is sufficiently mature for use in building operational diagnostic systems of a large scale. The approach of defining the problem by means of training examples has proven to yield rapid designs. Training examples may be readily used from a variety of sources such as real data, simulated data, theoretical data, and examples constructed by common sense reasoning. The back propagation type network provides a memory-efficient means for storing large numbers of training examples and a computationally-efficient means for applying the examples to perform the diagnoses.

ACKNOWLEDGEMENTS. Credit goes to Dr. Robert Sims of DCEC who had the foresight and mettle to sponsor this work. The design and development of the DASA/LARS system and the writing of this paper was a concerted team effort involving the following DCEC and GTE personnel. DCEC: Major Jody DeJonghe Acres, Jolynn Baugher, John Cormack, John Rogers, and Dr. Robert Sims. DCEC/STel: Pat Gordon. GTE Government Systems, Command, Control and Communications Systems: Michael Allen, Fred Casselman, Joseph DePrima, Dr. David Freeman, Debra Gennardo, Michael Grimaldi, Susan Landstrom, Scott Lane, Robert Martin, Mark Morisi, Henning Olesen, Peter Reinhard, and John Roder. Appreciation also goes to Larry Wood and George Leonard for their enthusiasm, foresight, and drive in establishing an extensive neural network capability at GTE Government Systems.

SCHEDULING BY SELF ORGANISATION

Ahmed Hemani, Swedish Institute of Microelectronics, Isafjördsg. 22, S-164 21, Kista, Sweden.
Adam Postula, Royal Institute of Technology, S-100 44, Stockholm, Sweden.
Tel: 46 -8 -752 11 70, email: ahmed@inmic.se

ABSTRACT

Scheduling data path operations into control steps is a crucial step in automatic synthesis of digital systems from behavioural specifications. This has been proven to be an NP-complete optimisation problem. This paper presents a new scheduling algorithm based on Kohonen's rule for self organisation.The algorithm's performance on well known benchmark examples is on par with the best reported in the field. We present a few such examples. The salient points of the algorithm are an inherent hill climbing mechanism, efficient implementation on massively parrallel structures and the ability to cope with a comprehensive set of constraints.

1.0 Introduction

In this section we give an overview of high level synthesis and concentrate on scheduling as its subtask.

High level synthesis of digital systems consists of translating an abstract behavioural specification of the system to a register transfer level structure. This is usually achieved in four steps:

1. Translating the abstract behaviour expressed in algorithmic style to a graph based representation. This graph, structurally represents the control and data dependencies of the operations in the behaviour. The graph is usually called *control and data flow graph* (CDFG).

2. This step is concerned with allocation of hardware resource. These are functional units like adders, multipliers etc, to realise operations, registers to store intermediate results between control steps and lastly busses and muxes for transferring data. Often only functional units are explicitly allocated and an attempt is made during scheduling (Step 3) and later during binding (Step 4) to minimise the use of registers, busses and muxes.

3. In this step operations are assigned to control steps or clock cycles. This is called operation scheduling and is the main concern of this paper. The primary objective of scheduling is to minimise the number of control steps required to fulfill a behaviour, given a certain amount of hardware resource. Alternately, given a global time constraint the scheduler could be asked to minimise the amount of resource required.

Steps 2 and 3 are highly interdependent and are not necessarily done in the order presented here. It is common to iterate between them to improve the overall peformance of the system.

Scheduling is crucial as it directly influences the following cost variables of the synthesised structure:
a. The number and type of functional units. A smart scheduler can schedule a behaviour in same number of control steps using fewer functional units. This is illustrated by one of the example in section 4. By serialising data independent arithmetic operations, it can influence replacement of many single functional units like adders, subtractors by one multifunction ALU.

b. Timing constraints on functional units. By increasing the number of control steps between data dependent operations it can suggest use of slower and thereby cheaper functional unit in terms of area.

c. The storage requirements (registers, memory). As the schedule decides how many intermediate results must be held at any instance, it influences the storage requirement.

d. Scheduling together with binding also influences the data transfer requirements.
The above cost variables are what a scheduler should minimise in proportion of their hardware area cost.

4. The last step in data path synthesis is binding allocated functional units to specific operations, registers and other storage elements to specific intermediate results and architecting muxes and/or busses to reallise data transfers.

2.0 Related work

In this section we briefly review the scheduling techniques in use and their associated problems.

The simplest technique is to let the user do the scheduling. This was used in the Silc system, one of the earliest so called silicon compilation systems. Combinatorial explosion rules out applicability of this technique for even modest size problems.

The simplest automatic technique is known as *As Soon As Possible* (ASAP) scheduling. This approach was used in CATREE and the Emerald/Facet systems among others. If the amount of hardware is restricted, the operations are conditionally postponed when there is a resource conflict. This approach was used in the MIMOLA, CMUDA and Flamel systems. The problem with this technique is that often less critical operations block more critical ones, resulting in longer than necessary schedules.

This problem is alleviated in *list scheduling* algorithms. In list scheduling, operations are sorted according to their data and control dependencies. These are then scheduled one at a time in the order in which they are sorted. If there is a resource conflict a priority function is used to decide which operations to defer. The priority function usually only considers the local effect of assigning operations to control steps, resulting in locally minimum solutions. The priority function varies across the systems using this technique, examples of which are SLICER, EMUCS and CATHEDRAL-II.

Force directed scheduling as used in the HAL system tries to overcome the problem of locally minimum solutions by taking into account more global effect of assigning an operation to a control step. At each iteration an operation is assigned to a control step which causes the least increase in overall concurrency of operation, storage and interconnect requirement, weighed in proportion to their hardware area cost. Though this algorithm is more global at each iteration, it is essentially a steepest decent algorithm. Once an operation has been assigned to a control step, its assignment is not reconsidered: It lacks a hill climbing mechanism.

Devadas and Newton incorporates a hill cllimbing mechanism in their system by using an algorithm based on simulated annealing. They use a complex cost function and model the problem as a placement problem, an area where simulated annealing has been very successfull. The best that can be said about the difficulty in parallelising simulated annealing is that it not easy: the tradeoff between accuracy and speed is tricky.

The algorithm described in this paper is inherently parallel in nature, has a hill climbing mechanism and has a built in cost weighing mechanism which allows it to do tradeoffs in functional unit, register and interconnect requirements based on their hardware area cost. It is similar to HAL in the sense that it reduces the resource usage by trying to uniformly distribute them over control steps as much as possible. It differs from HAL and others in: a) treating schedule space as a continuous space. b) all operations influence assignment of an operation to a position in the schedule space; HAL considers the effect of the assignment on concurrency of only successors and predecessors. c) it evalauates several solutions many of them worse than previous, before settling for a near minimal solution.

3.0 The Self Organising Scheduling Algorithm

The primary input to the algorithm is a control and data flow graph (CDFG). It is a directed graph (V, A). V is a set of nodes corresponding to the operations to be scheduled. $A \subseteq V \times V$, is set of arcs connecting nodes of V, these are called data arcs.
Besides CDFG, the other input to the algorithm is a set of constraints to be satisfied. These constraints take the following form:

. Type and number of functional units.
$F = [(T_1, N_1), (T_2, N_2), \ldots (T_i, N_i), \ldots (T_n, N_m)]$
Where T_i is the ith functional unit type and N_i is the number of functional units of type T_i. As the user does not know how many units are needed N_i is kept sufficiently large.

. A function g, mapping V to F; $g: V \rightarrow T$.

. Global time constraint MAX_K, cycle time and propogation delay of each functional unit type.

The output from the algorithm is a schedule table $S = MAX\_K \times F$ showing which operation has been assigned to which control step. The number of units of type T_i actually used are generally less than N_i, decided by the maximum used in any contol step for each type.

The components and organisation of the network used in the algorithm is as follows: The network is made up of an input node pair $I = (I_k, I_t)$, a set of output nodes V corresponding to the operations to be scheduled. Every output node $v \in V$ is connected to the input nodes (I_k, I_t) by a weight pair $W_v = (W_k, W_t)$. Let I_k and I_t be continuous random variables having uniform probablity distribution over the range $1..MAX\_K$ and $1..N$ respectively. Where, MAX_K is the maximum number of control steps and N is the total number of functional units of all types. Further, let every output node $v \in V$ occupy a place in the rectangular space $MAX\_K \times N$ as specified by its associated weight pair (W_k, W_t). Then by applying Kohonen's algorithm for self orgainsation to adapt the weight pairs, we would uniformly distribute the output nodes V over the space $MAX\_K \times N$.

We use the following concepts to describe the algorithm:

The d*ata dependency distance* between two data dependent operation is defined to be the longest path between them in the CDFG *(V, A)*.

At any instance *t*, the *neighbourhood* $NE_v(t)$ of operation v is defined as the set of predecessors and successors to v whose dependency distance with v is less than $\sigma(t)$. $\sigma(t)$ is initially large and slowly decreases over time.

We also use a *gain* term which is a function of dependency distance x and the current neighbourhood value and the initial neighbourhood $INIT\_\sigma$. It also decreases with time and is defined as follows:

$$\eta\ (x,\ t) = \frac{1}{\sqrt{2\ \pi}\ \ (INIT\_\sigma\ -\ \sigma\ (t)\ +\ 1)}\ exp\left(\frac{-\ x^2}{2\ \sigma(t)^{\ 2}}\right)$$

With every operation is associated a *time frame* and *type frame*. The time frame of an operation is defined as the range of control steps between ASAP and ALAP values. The ASAP and ALAP values are determined by performing as soon as possible and as late as possible scheduling. Each functional unit type T_i occupies a contiguous slot of length N_i in the N dimension of the schedule space $S = MAX\_K \times N$. This slot is called the type frame of operations which are mapped to this type by the function $g: V \rightarrow F$.

The Algorithm:

Step 1. Initialise the positions, i.e, the weight pairs $W_v = (W_k, W_t)$ of all operations to small random values. Set the initial value of $\sigma(t)$ to $INIT\_\sigma$. We found that setting $INIT\_\sigma$ to half the critical path in CDFG *(V, A)* was adequate.

Step 2. Present a new input by generating a random vector $I(t) = (\ I_k(t),\ I_t(t)\ )$, such that probablity distribution of I_k and I_t are uniform over the ranges $1..MAX\_K$ and $1..N$.

Step 3. Compute the distance d_{iv} between $I(t)$ and the weight pair $W_v(t)$ of all operations whose time frame and type frame includes the vector $I(t)$.

$$d_{iv} = \sqrt{(I_t(t) - W_t(t))^{\ 2} + \ \alpha\ .(I_k(t) - W_k(t))^{\ 2}}$$

α is a control parameter. It takes values: $\alpha \geq 1.0$. When α is one, the algorithm tries to position operations such that euclidean distance between them is as uniform as possible. By making it more than one we make the algorithm more sensitive to the K (control step) dimension of the schedule space. And by making α proportional to the area cost of resources, we enable the algorithm to do tradeoffs based on the area cost of resources.

Step 4. Select the operation v^* with the minimum distance d_{iv}.

Step 5. Update the positions of operation v^* and its neighbours by adapting the weights as follows:
$W_v(t+1) = W_v(t) + \eta(x,t)\ (I(t) \pm min\_dist\ (v,\ v^*) - W_v(t))$
For $v \in NE_{v^*}(t)$.
Here x is the dependency distance between v and v^*
min_dist $(v,\ v^*)$ is similar to data dependency distance, except that it takes into account the actual propogation delays and the effect of alligning operations to control step boundaries. Plus sign is used if v is successor of v^* and minus sign if v is predecessor to v^*. The above equation shows that both W_k and W_t weights of v^* and its neighbours are adapted. But in reality W_t weights are adapted for v^* and only those neighbours which are mapped to the same operation type.

Step 6. Decrease $\sigma(t)$. Repeat by going to STEP 2 until $\sigma(t) \leq 1.0$.

Step 7. Round the W_k weights of operations to nearest control step or time boundary if the operations are chained in a control step.

As every output node is connected to the input nodes by an independent weight pair, we can compute the distance function of STEP 3 simultaneously for all output nodes, if massively parallel structures were available. Same holds for step 5.
Initially, when the gain and the neighbourhood is large (analogous to high temperature in simulated annealing). Operations can move over large distances, influence far off neighbours and ignore the presence of other operations. As the neighbourhood and gain decrease with number of iterations, operations move smaller distances, influence closer neighbours and become sensitive to the presence of other operations.This annealing like behaviour is responsible for the hill climbing mechanism in the algorithm.

4.0 Results

We ran the algorithm on a number of benchmark and other examples of moderate to large size. The result in all cases were on par with the best reported in the literature. Here, we present two of these examples:

The fifth order eclliptic wave filter from KUNG85 was used as a benchmark in 1988 workshop on High level synthesis and has been used by HAL88 and others. This example has 43 operations subjected to 60 data precedence constraints. We use this example to demonstrate the algorithm's ability to handle multi-cycle operations and pipelined units. Like HAL88, we assume addition and multiplication takes one and two control steps respectively. A multiplier is a two stage pipelined unit and coefficients are not necessarily multiples of two, so multiplication is not performed by a shifter.
We set the global time constraint to 19 and 17 control steps and made available more than the necessary number of functional units. In the first case the algorithm found the cheapest schedule using 2 adders and 1 multiplier. In the second case it found the fastest schedule using 3 adders and 2 multipliers. In terms of number of functional units this is equivalent to HAL88. The results are summarised in Table 1.

Our next example is borrowed from MAHA86 and used by PAULIN88. This example illustrates the algorithm's ability to handle mutiple operations per cycle. Briefly the constraints are: four control steps, more than necessary adders and subtractors, both having a propagation delay of 40 ns and a cycle time of 100 ns. The results are summarised in Table 2.

	HAL88		Self organised Scheduler		KUNG85
component \ Control Steps	19	17	19	17	17
adder (+)	+ +	+ + +	+ +	+ + +	+ + + +
multiplier (*)	*	* *	*	* *	* * *
time (secs)	6 min	2 min	3 min	3 min	N.A

Table 1. Results of the fifth order eccliptic wave filter.

Note for KUNG85 the multipliers are not pipelined

	PAULIN88	Self organised Scheduler	MAHA86
component \ Control Steps	4	4	4
adder (+)	+ +	+ +	+ +
subtractor (-)	- -	- -	- - -
time (secs)	N.A	3 min	N.A

Table 2. Results of the MAHA code sequence example.

5.0 Conclusion

We have presented a new algorithm for scheduling data path operations using Kohonen's Self Organising algorithm. We have also presented the alogorithm's performance on well known benchmark example and other example taken from current literature. We are presently working on extending the algorithm to handle algorithmic pipelining.

REFERENCES

[KOH88] T. Kohonen. Self Organisation and Associative memory. Second Edition. Springer Verlag.
[KUNG85] S.Y.Kung, H.J. Whitehouse, T. Kailath, "VLSI and Modern Signal Processing", Prentice Hall, 1985.
[PAULIN88] High level synthesis of Digital circuits using Global Scheduling and Binding Algorithms", Ph.d thesis, Carleton University, February 1988.
[HAL88] High level synthesis Benchmark Results using a Global Scheduling Algorithms",
[MAHA86] A.C. Parker et. al, "MAHA: A Program for Dara Path synthesis", 23rd DAC, July 1986.
[NEURAL89] Ahmed Hemani, Adam Postula. Cell Placement by Self Organisation. Neural Networks Journal. Accepted, but date not yet known.
[FLAMEL] H. Trickey. Flamel: A high level synthesis compiler. IEEE transactions on CAD, March 87.
[Devdas] Devdas et. al. Algorithms for Hardware Allocation in Data Path Synthesis. IEEE transactions on CAD, July 89.
[CATHEDRAL-II] H.D Man. A silicon compiler for DSP. IEEE Design and Test, Dec. 86.

DEVELOPMENT OF A NEURAL NETWORK AUTOPILOT MODEL FOR A HIGH PERFORMANCE AIRCRAFT

Gary M. Josin

Neural Systems Incorporated
2827 West 43rd Avenue
Vancouver, B. C. CANADA
V6N 3H9

Abstract

A conceptual neural network has been substantially advanced to deal with a larger class of problems. When the target outputs are not known in advance and must be generated in real-time, reinforcement learning can be implemented using an appropriate performance function and/or a human expert. A simulation shows how to use the conceptual neural network for the preliminary development of an autopilot model for a high performance aircraft. The essence of this technique is to provide precise control of the aircraft during flight maneuvers utilizing an appropriately constructed cost function which characterizes the physics of the maneuver. During training, the network first learns a function to capture the aircraft and then it learns to generalize the function to reliably perform the flight maneuver. It is significant that when the autopilot represented in the neural network is engaged, it generates functions that provide autopilot command signals to the normal aircraft control system for a wide range of off-conditions. This indicates that the neural networks may enable the expansion of autopilot functions of use for aircraft control problems.

Introduction

Previous research investigations dealt with the feasibility of using a conceptual neural network for robotics. The research work discovered that neural networks are capable of *generalizing* unknown topological inverse kinematic transformations. Initial studies involving robots showed that a neural network theory can generalize transformations from cartesian end-point coordinates to joint angle coordinates based only on correct input and output examples of the mapping - *supervised learning* [1].

This current research work deals with the problem of learning an unknown transformation when there are no output examples of the transformation, but there is a predefined performance criteria or human expert which grades the performance of the network - *reinforcement learning*. For instance, in flight control, when controlling the longitudinal axis of the aircraft, it is not known apriori what the appropriate control system input should be to perform a particular flight maneuver.

The purpose of this article is to show the potential neural networks hold for the real-time control of aircraft. As an example, to assess the capabilities of a neural network for real-time flight control, computer simulations of a high performance aircraft integrated with a neural network were performed. For these simulations straight-and-level flight was chosen for the flight maneuver.

Aircraft Model

The aircraft model simulator used in this research was developed at the NASA Ames Dryden Research Center, Edwards, California by Mr. Eugene L. Duke. This model consists of a detailed, full-envelope non-linear aerodynamic model, a complete non-linear control system model, and a simplified linear thrust model. The vehicle represented by this model would be that of a modern fighter aircraft such as the F-15, F-16 or F-18. The simulator is written in the FORTRAN language.

The nonlinear equations of motion used in the program are general six-degree of freedom equations depicting the flight dynamics of a rigid aircraft flying in a stationary atmosphere over a flat non-rotating earth. The simulation program contains a 120 variable observation vector of such observables as surface deflections and power settings and a 30 parameter control vector corresponding to such parameters as pilot stick and throttle. For further details of the aircraft model see reference [2].

For the experiment reported in this article, the simulation was limited to a two-degree of freedom model in which angle of attack and pitch rate were allowed to vary but velocity, angle of sideslip, roll rate, and yaw rate were fixed. Thus altitude and pitch attitude were controlled indirectly using the longitudinal command signal. Which through the aircraft control system model, directly controlled longitudinal surfaces that generated pitch rate and ultimately changed angle of attack. A simplified representation of the model being controlled with the neural autopilot is taken from [3] and is shown in Figure 1.

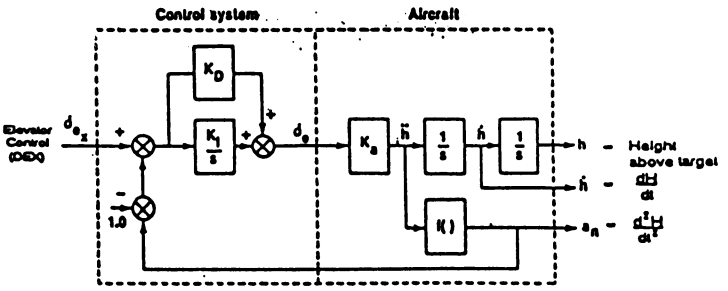


Figure 1. Simplified longitudinal model of the high performance aircraft and its control system

Cost Function

The performance criteria or cost function J for the flight dynamics of the longitudinal axis of the aircraft during straight-and-level flight can be described by the following equation:

$$J = H + (dH/dt)\,t + (d^2H/dt^2)\,t^2 \qquad (1)$$

where H is the height above or below the desired altitude of flight in units of feet. When the above expression for J is minimized, the flight path is constant and there is straight-and-level flight.

To use the cost function to control an unknown system, the desired control input $U_{desired}$ (target for reinforcement learning) at time (t), is selected in the direction of increased performance of the unknown system by the following equations:

$$U_{desired}^{(t)} = U_{desired}^{(t-1)} + \triangle U \qquad \text{Reinforcement} \qquad (2a)$$

$$U_{desired}^{(t)} = U_{desired}^{(t-1)} - \triangle U \qquad \text{De-reinforcement} \qquad (2b)$$

where $\triangle U$ is a change applied to increase performance in the next time-step. The value of $U_{desired}$ is then used as a target to train the neural network.

Reinforcement/De-reinforcement Learning

In order to understand how the conceptual neural network has been advanced to deal with a larger class of problems we construct a closed-loop model which integrates a neural network with the aircraft system (see Figure 2a). For this system, input/output examples or more specifically the target outputs are not available. These target outputs must be determined dynamically during the training cycle for the neural network.

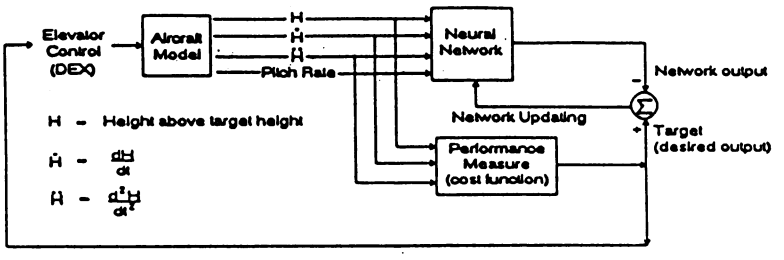


Figure 2a. Closed loop system of neural network and aircraft model during network training.

At the first time-step the elevator setting from the performance function is input into the aircraft model control system. The parameters H, dH/dt and d^2H/dt^2 which are observable output signals from the aircraft model enter the performance function and are used to calculate the current value of the cost function. Then the parameters of the cost function and the pitch of the aircraft were input into the network generating a network output.

If the cost function is closer to minimization then at the previous time-step, then at next instant of time the elevator setting is incremented by a predetermined quantized setting (Equation 2a). This new elevator setting then defines the desired target for network learning. This desired target with H, dH/dt and d^2H/dt^2 as inputs are used to adapt the weights of the neural network.

On the otherhand, if the calculated cost function is not minimized from the elevator setting at the previous time-step, then the elevator setting is decremented by a predetermined quantized setting (Equation 2b).

After training the neural network is used to control the unknown system in real-time and it does not lag one time-step behind the operation of the unknown system as it does while training. (See Figure 2b).

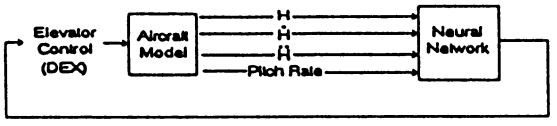


Figure 2b. System during autopilot operation.

Neural Network Autopilot Application

The neural network used for this application was a layered network of three layers. Each layer of the network completely connected to the next layer only. Back-propagation of errors was used as the update rule to change the network weights. When to apply the update rule was determined by reinforcement learning as described. The network was configured with 4 input processing units, 8 hidden layer units and 1 output unit. The network input and output representation is as follows:

INPUTS	OUTPUTS
H, height above target height dH/dt d^2H/dt^2 Aircraft pitch rate	Aircraft elevator control (DEX)

The neural network was developed in software and was interfaced to the non-linear high performance aircraft model. The elevator setting is used to control the flight of the aircraft for 100 milliseconds. The initial connection strengths of the network before learning were set to values randomly distributed within the range -0.2 to 0.2.

The real-numbers representing observable parameters of the aircraft system are encoded as neural signals (firing rates) within the neural network by entering them into the first layer of the neural network. The input signals vary over a wide range and scaling of the signals was required. The input signals were scaled using a hyperbolic tangent function (tanh) in order to emphasize the mid range about zero of the input signal information.

Results

Figures 3 shows the flight path of the simulated aircraft for target heights above and below the training height. Three different target heights of 15000, 15200 and 14900 feet are shown.

Figure 4 shows the deviation about the target height. The maximum deviation above target height is 3.13 feet, the maximum deviation below target height is -1.9ft.

Figure 5 shows the flight path for the aircraft when the thrust is varied. Notice that the neural autopilot compensates for the variation in thrust and controls the aircraft to the target height.

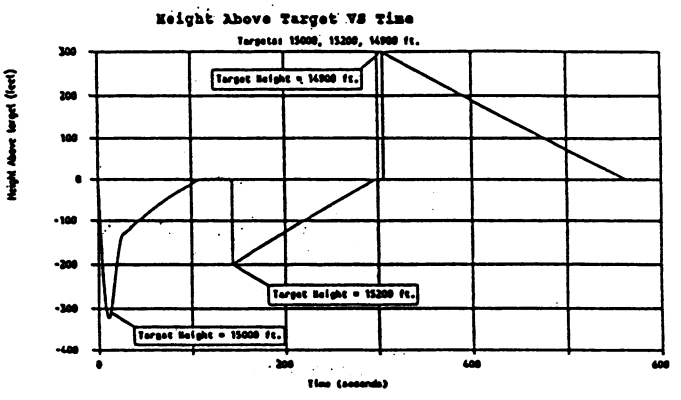


Figure 3.

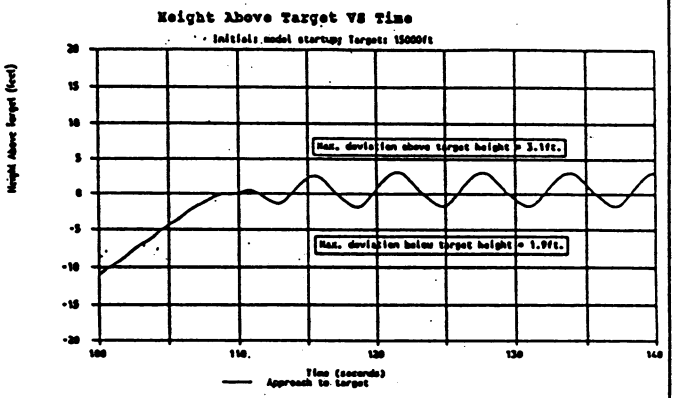


Figure 4.

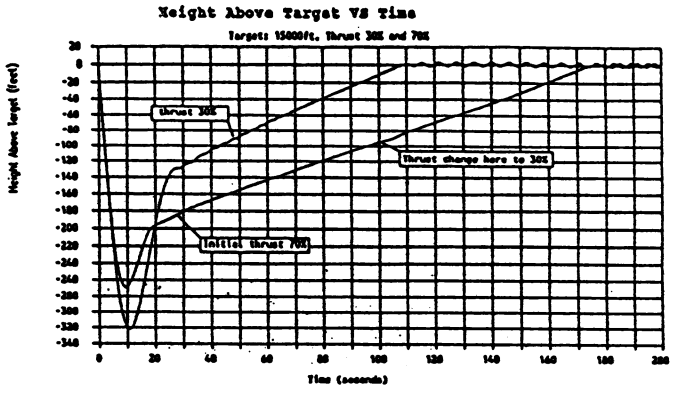


Figure 5.

Conclusion

Neural networks provide a potential tool for the development of a neural autopilot for highly maneuverable aircraft. They can be used as described here to control the longitudinal flight path of a high-performance aircraft model by training the network on an appropriately constructed cost function.

This reveals that the neural networks may enable the expansion of autopilot functions to control the thrust, lateral stick and rudder for flight maneuvers. We will develop these neural autopilot functions for high performance aircraft at a later date for live experimental flight test maneuvers.

References

[1] Josin, Gary M., **Neural-space Generalization of a Topological Transformation,** Biological Cybernetics, 59, 283-290, (1988).

[2] Duke, Eugene L.; Jones, Frank P.; Roncoli, Ralph B., **Development and Flight Test of an Experimental Maneuver Autopilot for a Highly Maneuverable Aircraft,** NASA Technical Paper 2618, September, (1986).

[3] Duke, Eugene L., **Application of Flight Systems Methodologies to the Validation of Knowledge-Based Systems,** NASA Technical Memorandum 100442, July, (1988).

Modular Back-Propagation Neural Networks For Large Domain Pattern Classification

Nagesh Kadaba Kendall E. Nygard Paul L. Juell Lars Kangas

Department of Computer Science and Operations Research
North Dakota State University
Fargo, North Dakota 58105-5075

Abstract

A significant problem associated with application of the Back Propagation learning paradigm for pattern classification is the lack of high accuracy in generalization when the domain is large. In this paper we describe a multiple neural network system, which uses two self-organizing neural networks that work as teaching data filters (feature extractors), producing information that is used to train a generalization neural network. We find that the modular neural network system described in this paper learns fast and generalizes very efficiently. The technique was successfully applied to the selection of control rules for a Traveling Salesman Problem (TSP) heuristic, thus making the TSP heuristic adaptive to the input problem instance. In these experiments, the multiple neural networks system was trained with a collection of 50 problems and achieved 100% accuracy in identifying the best control rules for the problems in the training set. These networks correctly classified 47 of 50 test problems not contained in the training set. In parallel experiments with single generalizing neural networks (with differing numbers of layers and topologies), generalization accuracy rates were in the 10-20% range. The high accuracy in the multiple network system described in this paper is especially noteworthy because, despite the importance of the problem in Operations Research, there was only a very limited body of knowledge upon which to base the control rule selection decision until this study was carried out. In addition, the results shown here should generalize to many applications employing back propagation for large domain data classification.

1 Introduction

The back propagation (BP) learning paradigm [1] is very popular among neural network researchers due to its property of capturing high order structure inherent in the data. But, the accuracy of generalization depend upon many factors. They are i) Network topology selection, ii) Network learning parameters selection, iii) Maintaining rotation and translation invariant feature extraction of teaching patterns. iv) Completeness of representation of the problem space in the teaching patterns. The key to high accuracy in generalization is human identification of the salient invariant features in the input problem and presentation of these features to the neural network.

The BP networks have a problem of handling input descriptors of high cardinality. One way of reducing the cardinality of the training data would be to decrease the resolution of the input/output descriptors, but this would lead to lot of information loss. The other way is to use some kind of data compression. In this paper we describe a multiple neural network system which uses the BP self-organizing network [2] to eliminate the ambient noise, created during data encoding and thus produce a concentrated high order form of the training input/output. This filtered training data is then used to train the BP generalization network. [3] The BP generalization network which is trained with these high order concentrated features clearly outperformed BP generalization networks that were exposed to training data without the feature extraction. We illustrate this point by the following Operations Research application.

The Traveling Salesman Problem (TSP) is one example of a classical NP-complete problem, a problem that is inherently difficult and the time required to find an exact solution increases exponentially with the number of stops. Since optimal algorithms are highly unlikely to solve large problems, a heuristic is used by stopping with a suboptimal solution. We use a selection-insertion type of tour construction heuristic as reported by Golden and Stewart [4], in which stops not yet in the tour are selected according to some selection criterion (control rule1), and inserted into the growing tour in a position determined by an insertion criterion (control rule2). The algorithm repeats until all stops are in the tour. There are 6 selection rules and 6 insertion rules. For example, one could select the stop point closest to some stop already in the growing tour and insert in the position that cause the least increase in the total distance. This would be the nearest neighbor selection rule and cheapest insertion rule. Thus there are 36

different combinations of selection and insertion rules.

This TSP heuristic is static, meaning, the same predetermined combination of the selection-insertion rule remains in effect until all stops are in the tour and for all types of problems. The multiple neural network system described in this paper helps in the selection of the selection-insertion rule best suited to a given instance of the problem. This makes the TSP heuristic adaptive to the input problem instance. Manual or Automatic selection of these rules has been impossible as there is a very limited body of knowledge upon which to base the decision. In the following sections we describe this multiple neural network system and present the experimental results.

2 System Overview

The architecture of the complete system is shown in Figure 1. It basically consists of three stages which are described in more detail in following sections. The first stage serves to create and encode the domain specific data. The raw problem data consists of cartesian coordinates of stop location. This data is pre-processed to make it invariant to scaling and rotation, producing a 30 unit problem descriptor vector. This encoded data is used as input to the stop data feature extraction (SDFE) network . The raw performance data consists of 36 values of distances. These distances represent the total mileage of the tour, generated by the TSP heuristic for a given combination of the rule. The 36 values are normalized between 0 and 1.0 in the output encoding module which is used in the performance feature extraction (PFE) network. These are the performance descriptors the system has to categorize during on-line recall. The second stage extracts the features from the problem/performance descriptors and feeds the resulting feature vector train to the generalizing network. The SDFE vectors and the PFE vectors are stored in two separate files. These two files are then used to train the generalizing network. Thus, we have 3 off-line trained networks which we can now use for on-line recall. The third stage shows the on-line generalization run, resulting in the rule recommendation vector. The reconstruction network is shown in dotted lines because it is used only during the on-line recall phase, and no learning occurs here. The trained weights are copied from the PFE network. The reconstruction network is precisely the bottom half of the PFE network after training.

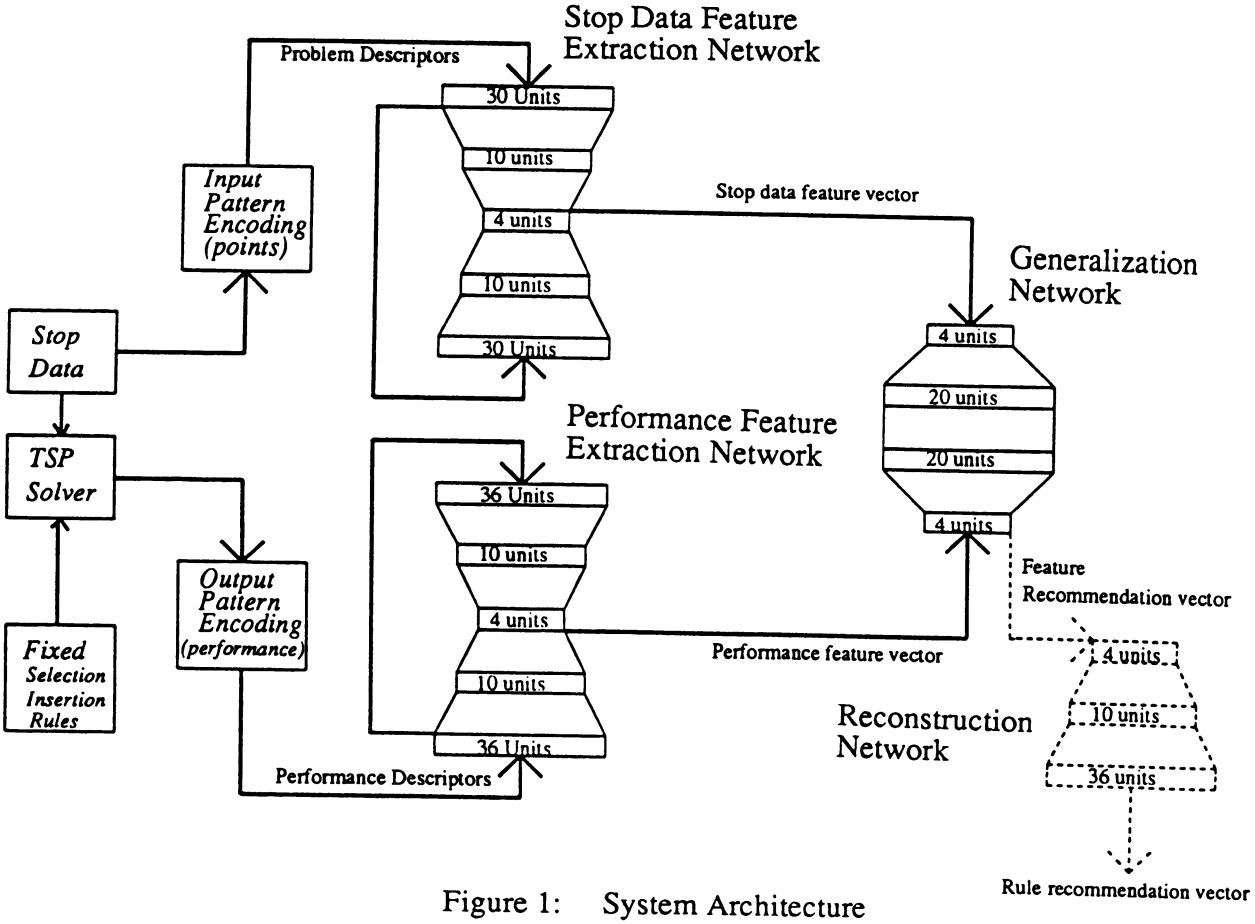


Figure 1: System Architecture

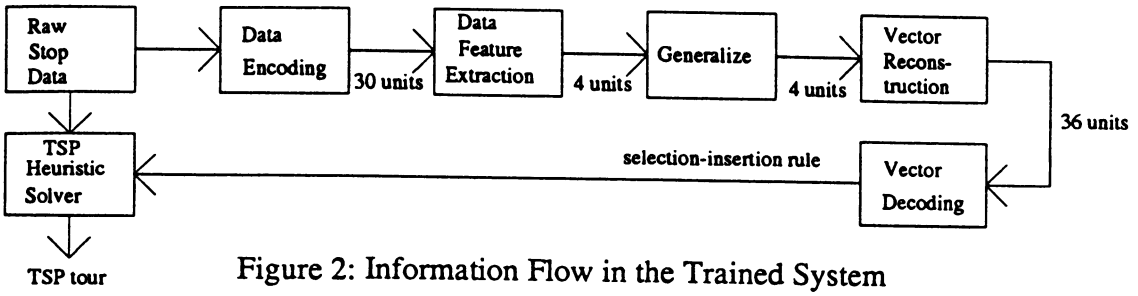


Figure 2: Information Flow in the Trained System

In Figure 2, we illustrate an information flow diagram during the on-line recall phase once all of the networks have been trained. The new testing stop data is presented to the data encoding module, which produces a 30 unit problem descriptor vector. This problem descriptor is exposed to the SDFE network which intern produces the 4 unit stop data feature vector. This stop data feature vector is then exposed to the trained generalization network. The generalizing network on recall, produces the 4 unit feature recommendation vector, which is presented to the reconstruction network. The original dimension of 36 unit performance vector is generalized by the reconstruction network. This vector is decoded by using a high value close to 1.0 to represent the best selection-insertion control rule combination for the given instance of the test stop data. Finally, the raw stop data and the best rule combination is presented to the TSP heuristic, which in turn produces a TSP tour.

3 Data Encoding

Neural networks have the inability to recognize shifted or rotated versions of patterns used in training. Our solution to this difficulty is to represent the input data in a way that is invariant to shifts and rotation. Therefore pattern encoding of the input data is carried out before presentation to the neural networks as shown in Fig. 1. The raw problem data consists of the stop point coordinates, their demands for service, and the number of available vehicles and their capacities. All of the problems had 100 stop points. To adequately represent the stop data we used a real valued vector of length 30 and a real valued vector of length 36 to represent the performance of the TSP heuristic.

Problem Descriptor 1. The geographical area is normalized into a 100 X 100 square area, then partitioned into frequency classes by concentric rings. We used 10 frequency classes and this 10 valued vector represented the percentage of the stop points observed in each of the 10 areas defined by the rings.

Problem Descriptor 2. To assure that tightly clustered groups of stops were adequately represented, we also calculated the mean distance of the stops within each area from the ring that forms the inner boundary of the region. The values were normalized into proportionate distances from the inner ring boundary, defining a corresponding vector of real values, each between zero and one.

Problem Descriptor 3. Angles between pairs of adjacent stops with respect to the center of the region were used as the third input descriptor. The purpose is to capture clustering in radial sense relative to the depot. Frequency counts within 10 classes were taken and a 10-unit vector of proportions within the classes was used as the descriptor. Again note that the descriptor is independent of the radial location of a cluster of stops, but does provide a measure of the relative dispersion of the stop points.

Performance Descriptor. The TSP heuristic were run 36 times with each combination of selection-insertion rule. Each of the 36 performance values produced by the TSP heuristic, represented in miles are then normalized with respect to the worst and the best rule combination during output pattern encoding. The value 0 representing the worst rule combination and value 1 representing the best rule combination. All the values between 0 and 1 represent the quality of the route length relative to the best combination, this is important information we wish to capture. We also realized that the relative performance of the rule combination (Rule1/Rule3, Rule2/Rule6 ..) was sensitive to the presence of the clustering in the location of the stop points.

4 Neural Network Architectures

We have used basically three types of neural networks, BP self-organizing feature extraction neural networks, a BP generalization neural network and a feed forward reconstruction network.

BP Self-organizing Neural Networks: As Figure 1 illustrates, the encoded problem data is presented to a SDFE network. The role of this network is to concentrate the problem descriptors into high order features that can later be used to train the generalization network. The self-organizing paradigm forms an compact internal representation of

its environment. In the TSP application, we choose to concentrate the data to a feature vector of 4 real numbers. The number of hidden units required to represent the data was determined empirically. We expect that 4 to 6 units would be adequate for many applications. The SDFE network has a 5-layer fan-in/fan-out architecture. The network is trained with back propagation in an unsupervised mode, accomplished by setting the 30 output unit values equal to the value of the 30 input units. Through this process of learning its own input, the weights are adjusted in such a way that the activations of the 4 units in the middle layer summarize high order features of the input problem that can be used in the subsequent classification. The PFE network is similar to the one explained above, except that the input/output layer has 36 units.

BP Generalization Neural Network: This is a typical feedforward network, trained with the error back propagation learning algorithm. The network has 4 unit input layer, two hidden layers of 20 units each and an output layer of 4 units. During off-line training, the generalization neural is trained using the 4 unit vector from the SDFE network as teaching input and the corresponding 4 unit vector from the PFE network as teaching output.

Feed-forward Reconstruction Network: As Figure 2 illustrates, the feature recommendation vector that is generalized from the trained generalization network is presented to the reconstruction network. No training occurs in this network, it is used only during the generalizing phase of the system. The reconstruction network has the same topology of the PFE network, but only the bottom half. Once the PFE network is trained, the weights and biases are transferred to the reconstruction network. So, when the the feature recommendation vector is presented, it remaps the reduced dimensional features back to the 36 unit original dimension with minimum average deviation from all the patterns presented to the system during training. The output layer uses a high value close to 1.0 to represent the best rule combination for a given instance of a problem.

5 Simulation Results and Conclusions

The purpose of the initial simulations was to determine if the BP generalization network would be capable of categorizing 36 distinct classes. The encoded stop location data and the encoded 36 relative performances of the TSP heuristic

were presented to a single BP generalization network. Several experiments were conducted on the single generalization with differing numbers of layers, topologies and learning parameters. These experiments typically recommended the best rule combination with only 10-20% accuracy. This led us to more experiments with the multiple neural network system.

We trained each of the networks with 50 problem patterns. The SDFE network was trained with 50 problem descriptors and the results of the run were stored in a file. This file contained 50 vectors of 4 units. The PFE network was trained with 50 performance descriptors and the results of the run were stored in a file. This file contained 50 vectors of 4 units. These two files were used as teaching input and teaching output for the generalizing network. All the networks were trained on an IBM 3090 computer. After the network system was trained the weights were downloaded to a SUN workstation. In testing the on-line recall of the multiple neural network system, we ran 50 problems that the system was not exposed to. Each problem was run with each of the 36 alternative TSP heuristic solvers, to provide an accurate measure of the multiple neural network system. Testing on the new 50 problems achieved an accuracy rate of 94%. This is in contrast to 10-20 % accuracy with single generalizing neural networks.

The results of the simulations indicate that the proposed method of using high order concentrated features to train BP generalizing networks, yields very high accuracy rates. We expect that the results of this research will apply to a wide variety of applications, to improve the accuracy of categorization in back propagation networks with large numbers of pattern choices.

References

[1] J.L. McClelland and D.E. Rumelhart, *Explorations in Parallel distributed Processing: A Handbook of Models, Programs, and Exercises*, MIT Press, Cambridge, MA, 1987.

[2] M. Myers, R. Kuczewski and W. Crawford, *Application of New Artificial Information Processing Principles to Pattern Classification*, Final Report, U.S. Army Research Office, Contract DAAG-29-85-C-0025, 1987.

[3] K. Fukushima and S. Miyaki, "Neocognitron: A new algorithm for pattern recoginition tolerant of deformation and shifts in position," *Pattern Recognition*, 15, (6), 455-469, 1982.

[4] Golden, Bruce L. and William R. Stewart, *"Empirical Analysis of heuristics" in the Traveling Salesman Problem*, ed. Lawer, Lenstra, Rinnooy Kan, and Shmoys, John Wiley and Sons, 1985.

NEURAL NETWORKS FOR ADDRESSING THE DECOMPOSITION PROBLEM IN TASK PLANNING*

Chandrashekar L. Masti and David L. Livingston
Department of Electrical and Computer Engineering
Old Dominion University
Norfolk VA 23529-0246.

ABSTRACT

An artificial neural approach to the machine decomposition problem is presented. Necessary conditions for the network processing capabilities are established. The algebraic structure inherent to the decomposition problem is exploited to enable formulation of a cost function with degenerate ground states. A third-order Boltzmann machine that meets the required conditions is developed and the results of simulations are discussed.

INTRODUCTION

We as humans seem to have a very natural and astounding capability to efficiently plan and execute a given task regardless of the complexity or level of the task description. The human brain has demonstrated, with as yet no parallel, an outstanding ability to identify existing structures or dependencies within a given task and formulate an efficient method of decomposing the task into sub-tasks. If there is a way of emulating the human brain's ability to decompose tasks, it is becoming increasingly clear that it must be through a neural approach to the problem.

Decomposition techniques for task planning and scheduling currently rely on symbolic and heuristic methods such as those employed in artificial intelligence. Livingston and Serpen have used lattice theory to examine the decomposibility of task structures modeled by finite state machines and demonstrated the use of Boltzmann machines to plan decomposed tasks [1][2]. However, they have also stated that a significant part of the effort of developing task structures consists of the combinatorially explosive problem of finding the elements of the lattice of substitution property (s. p.) partitions from which the decompositions are obtained. Since neural networks have shown great promise in their capabilities to solve problems that are combinatorially explosive such as the traveling salesman problem [3], we explore their use in our attempt to circumvent the intractability of the partition search problem.

NETWORK THEORY

Without detailing the algebraic theory of automata we define s. p. partitions on the state set of a finite state machine as collections of nonintersecting congruence classes collectively exhausting the entire state set [4]. Every s. p. partition defines a unique congruence relation between every state of the machine. A congruence relation posesses the properties of reflexivity, symmetry and transitivity and, because of satisfying the substitution property, also preserves the next-state mapping on every state of the machine under every input.

In order for a neural solution to the problem to have a direct and intelligible meaning upon convergence of the network dynamics, the aspect of a proper representation must be addressed first. To do so, we use an important concept from relational algebras: the relation matrix [5].

---

* This research was supported by the National Aeronautics and Space Administration under grant NAG-1-962.

A relation matrix describes a binary relation R between any n elements of a groupoid as follows

$m_{ij} = 1$, if iRj and

$= 0$, if $iR^{c}j$,

where $i,j \in R$ & $m_{ij} \in$ n-by-n relation matrix M.

Therefore, the representation must be an N×N neuronal grid such that

$V_{ij} = 1 \equiv$ "ON", if iRj

$= 0 \equiv$ "OFF", if $iR^{c}j$,

where $i,j \in R$ and V_{ij} is the activation of any neuron corresponding to the element $m_{ij} \in M$. However, the number of neurons in the network that actually take part in the dynamics need only be $N(N-1)/2$, since the properties of reflexivity and symmetry are implicitly encoded in such a representation.

Having addressed the aspect of a proper representation, what follows is the formulation of a cost or energy function that should reflect any violation of, or compliance to stipulated constraints derived from the substitution and transitivity properties.

To develop the energy function we must first establish the necessity of using a third-order network to find s. p. partitions. Thus we state the following theorem, omitting the proof for the sake of brevity.

Theorem: To determine the smallest transitive relation between the states of a finite state machine, it is necessary for a neural network to extract third-order correlational information from its input set.

Illustration

Consider a neuron "ON" in the position (0,2) and another "ON" at position (0,4). This directly reflects a proposed equivalence between states (0,2) and states (0,4). But since equivalence is transitive, the states (2,4) must also be shown as equivalent by the network. To verify this fact, it is now clear that the third neuron responsible for displaying the equivalence between states (2,4) must be "ON" whenever the other two are "ON". Thus the decision for any one neuron must be based upon the activations of pairs of other neurons.

Cost function formulation

Based on the theory developed in the preceeding material, the energy function is derived in parts by considering each factor responsible for reflecting the state of the network with respect to each constraint individually and then summing the parts.

The first term represents the transitivity constraint:

$$E_1 = \{K\_transitivity(\sum_i \sum_j \sum_k (V_{ij}\times V_{jk}+V_{ij}\times V_{ik}+V_{jk}\times V_{ik} - 3\times V_{ij}\times V_{jk}\times V_{ik}))\}.$$

his term equals zero only when the current partition reflected by the present state of the network does not violate the law of transitivity, but is positive in all other situations.

Next is a constraint which forces the substitution property:

$$E_2 = K\_next\_state(\sum_i \sum_j \sum_l (V_{ij} + V_{ei\,ej} - 2\times V_{ij}\times V_{ei\,ej}))\}.$$

This term is zero only when the next state function is preserved intact in the sense that state equivalence and image equivalence are both preserved in the current partition generated by the network.

The final term reduces the chance of finding trivial partitions:

$$E_3 = \{K\_trivial(\sum_i \sum_j V_{ij} - equal\_size\_blocks\_enforcer)^2\}.$$

It equals zero only when the total number of neurons "ON" in the network exactly equal the user specifiable parameter "equal_size_blocks_enforcer" and becomes positive for all other cases.

The total energy is therefore

$$E = E_1 + E_2 + E_3. \quad (1)$$

If the correct values for the parameter "equal_size_blocks_enforcer" are provided, the network can be influenced to relax to an s. p. partition that contains a small number of blocks. This is a particularly useful solution in the decomposition process, since what it essentially means is that a large number of states of the original machine model are equivalent, resulting in a highly refined decomposition.

The effective values for the gains, K_transitivity and K_next_state, may be set to unity because of the equal importance of both these constraints. The function of K_trivial is only to make sure the network avoids finding the two trivial s. p. partitions typically denoted by $\pi(I)$ and $\pi(0)$.

The ground state of E was kept degenerate meaning that all global minima in the landscape for E must occur at E = 0 and correspond to solution points for the problem. This offers the attractive advantage of the elimination of a search for the gain parameters.

The Hamiltonian function formulation for a third-order constraint satisfaction network is

$$E = -(1/3)\times[\sum_i \sum_j \sum_k W_{ijk} V_i V_j V_k]$$

which, on comparison with equation 1 yields an algorithm for computing the entries to the three-dimensional weight matrix of size $[N(N-1)/2]^3$. Imposing the following conditions:

$$W_{iii} = 0 \text{ and } W_{ijk} = W_{ijk} = W_{kji} = W_{jik}$$

to ensure symmetry and no self-feedback guarantees that the energy of the network is nonincreasing [6].

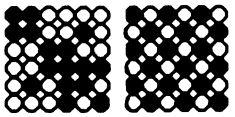
NETWORK SIMULATIONS AND RESULTS

In all our simulations, an attempt by a third-order Hopfield network to solve the problem failed. This is an inevitable consequence of allowing only downhill moves since the E function has multiple extrema in it's landscape. A third-order Boltzmann machine produced very good results. The artificial temperature was updated using the classical simulated annealing (CSA) schedule [7]

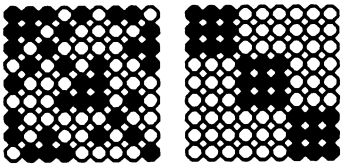
$$T_a(t) = T_{max}/\log(1+t).$$

When K_trivial is set to zero and K_transitivity and K_next_state are each set to unity, the network settles occasionaly to one of the trivial s. p. partitions $\pi(1)$ or $\pi(0)$. If it is necessary to avoid this, setting K_trivial to unity and using appropriate values for the parameter "equal_size_blocks_enforcer" will guarantee non-trivial s. p. partitions. It must be pointed out that this can sometimes lead to a parameter search in order to arrive at a value for "equal_size_blocks_enforcer" that will keep the ground state of E degenerate. Experience has shown that the number of times trivial s. p. partitions are generated is small relative to the number of occurences of non-trivial s. p. partitions. This indicates that such a parameter search can be eliminated by letting K_trivial be zero as suggested earlier.

Figures 1 and 2 show the initial and final states of a third-order network which finds the s. p. partitions. Figure 1.b represents the partition $\pi = \{\overline{0,2,4};\overline{1,3,5}\}$ on the state set of a modulo six counter which was acheieved in 30 time steps. The partition $\pi = \{\overline{0,1,2};\overline{3,4,5};\overline{6,7,8}\}$ on the state set of another machine with 9 states and 3 inputs as shown in Figure 2.b was obtained in about 500 time steps.



(a) (b)
Figure 1. 6x6 network.



(a) (b)
Figure 2. 9x9 network.

CONCLUSIONS

This work demonstrates that a neural approach to the decomposition problem offers a viable alternative to existing sequential search methods.

We have shown the necessity for the use of a third-order network for achieving a neural solution to the problem of task decomposition. A network with a deterministic update rule for it's neurons such as a Hopfield network fails to solve the decomposition problem because of its tendency to be trapped by local minima. A third-order Boltzmann machine which incorporates a stochastic element in its update rule is successful in addressing the same problem. An interesting aspect of the network is the absence of any "tuning". The implemented network scales favorably with the size of the state set of the finite state machine model.

The performance of the network model as implemented with the present architecture does, however, degrade with the size of the input set of the finite state machine. Real world problems in task decomposition can be expected to have large state sets as well as large input sets. A strategy based on the intersection of lattices of sub-groupoids for circumventing the input scaling problem is currently in progress and preliminary results are very encouraging.

REFERENCES

[1] D. L. Livingston and G. S. Serpen, "Evaluation of Lattice Theoretic Techniques for Task Decomposition Formalism", Proc. of IEEE SouthEastCon'89, vol. III, 1989.

[2] G. Serpen and D. L. Livingston, "A High Order Boltzmann Machine for Transfer Sequence Searches in Decomposed State Machines", Proc. of IJCNN Washington D. C., vol. II, pp. 598, 1989.

[3] J. J. Hopfield and D. W. Tank, "'Neural' Computation of Decisions in Optimization Problems", Biol. Cybern., Vol. 52, pp. 141-152, 1985.

[4] Abraham Ginzburg, "Algebraic Theory of Automata", ACM Monograph Series, Academic Press, 1968.

[5] J. P. Tremblay and R. Manohar, "Discrete Mathematical Structures with Applications to Computer Science", McGraw Hill Computer Science Series, 1975.

[6] T. J. Sejnowski, "Higher Order Boltzmann Machines", AIP Conference Proceedings 151, Snowbird, UT 1986.

[7] S. Geman and D. Geman, "Stochastic Relaxation, Gibbs Distributions and the Bayesian Restoration of Images", IEEE Trans. Pattern Anal. Machine Intell., vol. PAMI-6, pp. 721-741, 1984.

A Fault Tolerance Analysis of a Neocognitron Model

Qing Xu Charles Jurgens Begoña Arrue
Jay Minnix Barry Johnson R.M. Iñigo
Department of Electrical Engineering
University of Virginia
Charlottesville, VA 22901
804-924-6111

Abstract

Artificial neural networks (ANNs) are stated to be inherently fault tolerant by nature of design. However, little research has been done to substantiate this claim. In this paper, we will investigate the fault tolerant abilities of a forward connected neocognitron, a neural network designed for visual pattern recognition. Simulation results are presented which show the ability of the network to tolerate faults during operation and to overcome the faults via learning.

Introduction

Due to the massively parallel nature of ANNs, conventional evaluation techniques such as combinatorial or Markov modeling become computationally intractable. Consequently, an approach similar to that used by Belfore and Johnson [Belfore 88] is taken, in which qualitative analysis based on empirical data is used to determine the fault tolerance of the network rather than trying to set up a mathematical model. In conventional fault tolerance, the reliability of a system is in general, a primary design concern because these systems solve problems whose nature requires absolute solutions. On the other hand, ANNs solve optimization problems where the solutions required need only be sufficient. As a result, performance then becomes the primary metric of a neural network. In this paper, performability is defined as the probability that the system performance will be at or above some level given some number of faults injected into that same system. [Johnson 89]

The neocognitron is a hierarchial multilayered network of processing elements. Its design allows the network to select simple features from a stimulus pattern and then combine them to reconstruct the original pattern. The processing elements have inputs and outputs of non-negative analog values in order to correspond to the instantaneous firing frequencies of biological neurons. Variable and fixed connection weights in-between these cells are used to guide forward signal flow. For brevity, the reader is advised to consult the Fukushima papers for a more detailed discussion on the mechanics of this network. [Fukushima 88]

Analyical Methods

To commence with a fault tolerance analysis of a neocognitron model, we chose to use an existing computer simulation which we later modified for use as a test bed. In the simulation, parameters such as the number and size of the cell planes, the number of layers, the size of the connectable area and the size of the input pattern could be altered. The simulation parameters we used modeled a three-layer 7x7 neocognitron consisting of 2,882 elements and 77,835 connections.

Three relatively simple distinct binary geometric patterns were used to train the network. This minimized the number of identical features between the patterns. The reason for the training set was twofold. First, it allowed the neocognitron to easily learn and distinguish between the three patterns. Second, and more important was that if an injected fault was localized to a cell plane, then the network's performance should degrade for the one pattern associated with that cell plane. If multiple faults are introduced with each cell plane having an equally likely chance of being compromised, then an overall degradation in the performance of the network should be observed. In all, this eases the constraints on grading the performance of the network when faults occur.

In our study of the neocognitron we have concentrated on the effects of two types of permanent faults on two basic building blocks of the network: a) the connections, and b) the processing elements. Should a physical implementation of the neocognitron ever be attempted, it would be inevitable that component failures would exist. We can make no assertions as to the likelihood of which and how a component will fail in a physical implementation since no attempt has been made at analyzing the fault tolerant behavior of a hardware-based neocognitron. However, we do believe that the issue of faults stuck at zero and stuck at random are relevant for discussion. An analysis of connection and element faults was made but for the sake of brevity we will only discuss element faults in this paper.

In modeling stuck at zero and stuck at random faults in the processing elements, we assume that a fault occurs in the element's calculation of the activation function. With a stuck at zero fault, the cell is assumed to be functionally dead, producing zero output regardless of its inputs. A stuck at random fault

would indicate that the cell is producing a random output regardless of its inputs.

With the simulation code, a test shell was developed to inject faults into the model. Two phases of the neocognitron were analyzed: a) the performance phase, and b) the learning phase. The performance phase neocognitron is the "end result" of self-organization. In this phase, the network has completed its learning. There are no more weight modifications and, consequently, the internal variable weights are made permanent. The learning phase neocognitron describes the state of the network before the performance phase. Of interest from a fault tolerance viewpoint is the "winner take all" mechanism. Fukushima asserts that this mechanism provides the network with a self-repairing function. Specifically, Fukushima states, "If a cell that has responded strongly to a stimulus is damaged and ceases to respond, another cell, which happens to respond more strongly than others, starts to grow and substitutes for the damaged cell." [Fukushima 88] The purpose of this work is to test the validity of this hypothesis.

During the performance phase, element faults are introduced in the following manner: *a)* A type of element (u_s, u_c, u_v) is chosen to be faulty, *b)* The locations of all the faults are then chosen to be in a single sublayer, *c)* Locations for a chosen number of faults within the sublayers are randomly generated, *d)* The elements chosen to be faulty then have all of their outputs set at zero or at random.

Once a chosen number of faults has been introduced, the network is run to identify a pattern it has already learned. Of the eight output cells, a count is made of those cells whose outputs are greater than 10^{-1}. This is a heuristic value chosen with the knowledge that typical output values for cells not firing are in the range of 0 - 10^{-4}. Fifteen trials per chosen number of faults injected were made. Ideally only one cell will fire per trial. If this is the case a count value of one is given to that trial. If two cells fire, a count value of 0.5 is given to that trial. If three or more or no cells fire, we arbitrarily say that the network has failed and give it a count value of zero.

In the learning phase we concentrated our analysis on the elements stuck at zero. As before, we set each trial run such that faults occur in only one layer and with only one type of element. After the locations of the faults have been randomly generated, the elements specified to be faulty have their outputs set to zero during all iterations of the learning phase. Performance is measured after the learning phase is completed.

The neocognitron correlates input patterns with a unique u_c cell in the recall layer. However, the selection of the pattern $\Leftrightarrow$ u_c cell correlation is arbitrarily made by the network during the learning phase. Should an element's output be changed during the learning phase, there is no guarantee that the pattern $\Leftrightarrow$ u_c cell correlation will be the same as that for the fault free case. Regardless, our definition of performability remains intact, for as long as a correlation is made between an output cell and an input pattern, the network is said to work. As such, the methods used in determining the performability of the performance phase network will still apply to the learning phase. For our analysis, 10 trials per chosen number of faults injected were made. In addition, a performability analysis of the network with respect to shifted patterns was made.

Discussion of Results/Conclusions

Multiple runs of fault-injected neocognitron models were taken in an attempt to obtain "statistically" significant results. However, given the length of the simulation runs, limited CPU time restricted the amount of data that could be gathered. Nevertheless, the values that were used were then normalized to obtain performance and % damage values. % damage values were determined by dividing the number of faults injected by the total number of elements categorized to be faulty.

Performance Phase

For the performance phase neocognitron simulation, performance comparisons between layers were made for both stuck at zero (s-a-0) and stuck at random (s-a-r) fault models. Our results have shown that there is a distinct correlation between the type of fault injected and response in performance of the network. For the excitatory elements (u_s, u_c) in the first and second layers, s-a-r faults appear to have a much more damaging impact on network performance than s-a-0 faults. This is evidenced by the simulation results which for example show that with s-a-0 faults on s cells, the network can absorb as much as 20% damage to the elements and still maintain at least 80% performance, regardless of the layer attacked. On the other hand, u_s cell performance under s-a-r damage drops to zero with only 8% damage, regardless of the layer attacked. Similar observations can be said for the u_c cell response. To explain these results we offer this hypothesis. When s-a-0 faults are injected during the performance phase, most of the elements attacked are producing an output close to or at zero anyways. Consequently some of the s-a-0 faults injected will not yield errors. However with s-a-r faults injected, spurious information is added,

introducing errors which would prevent the network from functioning.

With the inhibitory elements (u_v) we see that the response of the system due to s-a-0 faults and s-a-r faults reverse roles, where s-a-0 faults now affect the network more adversely than s-a-r faults. This observation could be attributed to the fact that without any inhibitory output, there is no inhibition mechanism. On the other hand random outputs would provide the network a chance to remain functional. For both fault models the order of tolerance with respect to layers is 3v, 2v, 1v. From this observation we see that faults injected into earlier layers of inhibitory cells tend to wreck more havoc on network performance than in later layers. One possible explanation might be that errors generated at an earlier stage would propagate into a global effect, whereas errors in a later stage would be localized.

Learning Phase

Plots of performance comparisons between training set (gp) vs. shifted (sp) patterns presented to the faulty learning performance phase models were made. True to Fukushima's hypothesis, given fault free inhibitory elements, we have found the network to be exceptionally tolerant of faults injected into the excitatory elements. In one case we found that we could damage as much as 90% of the u_s cells in the second layer and still maintain full performance. This ability is attributed to the "winner take all" mechanism, which stipulates that if a maximum output cell is killed off during the learning phase, another cell with the next greatest output would have its input connections reinforced. However this feature has a cost which is paid for by the seed cell mechanism. When faults are introduced in the learning phase, the network diverts cells which would normally form redundant subnetworks in the cell planes to be "primary recognition" cells, i.e., the cells that get the pattern recognition mechanism to work. By taking away the redundant subnetworks, the network consequently loses its ability to recognize position shifted patterns. For the shifted patterns we observe again the greater decrease in performance from faults injected in earlier layers than in the later layers. This reconfirms our suspicions that errors generated at an earlier stage would propagate into a global effect.

The results gathered from the learning phase s-a-0 v cell model show the importance of the inhibitory cell mechanism to the operation of the network. With initial damage to the network, the inhibitory mechanism is completely shut down, allowing most if not all of the output cells to fire to a single pattern. However, this effect does not last forever and with more progressive damage the network fails to even converge to a solution, i.e. be able to make a pattern $\Leftrightarrow$ u_c cell correlation.

References

Fukushima, Kunihiko. "A Neural Network for Visual Pattern Recognition." *IEEE Computer*, March 1988, pp. 65-75.

Fukushima, Kunihiko. "Neocognitron: A Heirarchial Neural Network for Visual Pattern Recognition." *Neural Networks*, Vol. 1, 1988, pp. 119-130. Pergamon Press.

Lin, Linda. *Designing a Learning Neural Network for Pattern Recognition Based on the Neocognitron Model.* An Undergraduate Thesis for the School of Engineering and Applied Science, University of Virginia, Spring 1989.

Minnix, Jay. *Class Notes, Neural Networks.* Spring 1989.

Belfore II, Lee A. and Barry W. Johnson. "The Fault Tolerance of Neural Networks." *International Journal of Neural Networks Research and Applications*, Vol. 1, No. 1, January 1989, pp. 24-41.

Johnson, Barry W. *Design and Analysis of Fault Tolerant Digital Systems.* Addison-Wesley Publishing Company, Reading Massachusetts, 1989.

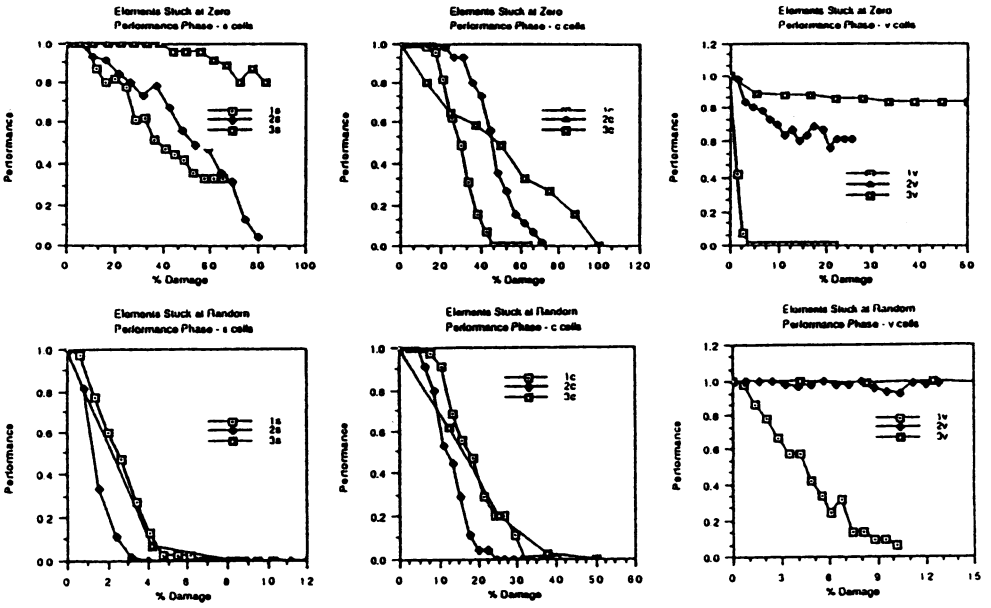


Figure 1 - Performance Phase Performance vs. % damage Results.

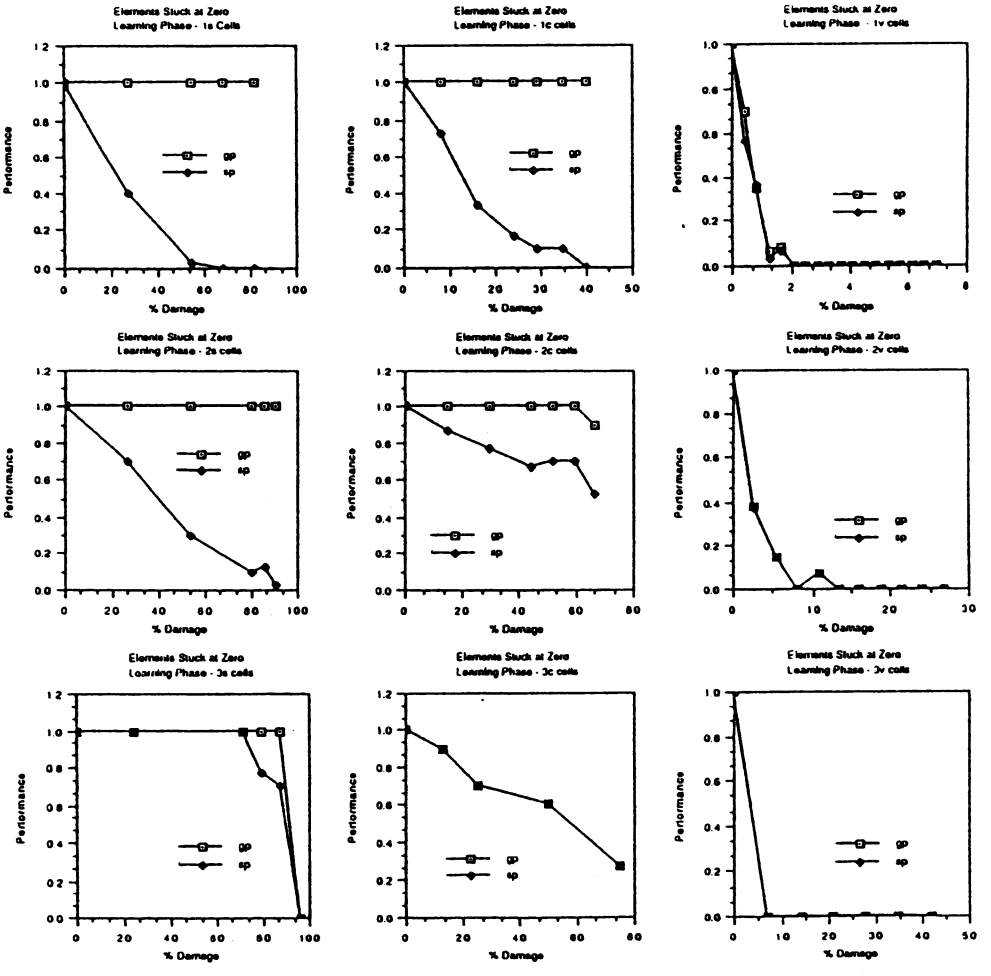


Figure 2 - Learning Phase Performance vs. % damage Results.

ROBUST TRACKING CONTROL OF DYNAMIC SYSTEMS WITH NEURAL NETWORKS

Stanislaw H. Żak
School of Electrical Engineering
Purdue University
West Lafayette, IN 47907

ABSTRACT

This paper is devoted to the problem of controlling a class of dynamic systems via controllers based on additive neural network models. In particular, the tracking and stabilization problems are considered. Some concepts from the variable structure control theory are utilized to construct stabilising controllers. In order to facilitate the analysis we employ a special state space transformation. This transformation allows us to reveal connections between the proposed controllers and the additive neural network models. We also present a procedure for designing neural networks based controllers.

1. INTRODUCTION

There are many neural network models ([4], [5], [7]). In this paper we will be concerned with the simpler additive model, also known as the Hopfield model, in the context of a control system tracking a reference signal. "The additive model has continued to be a cornerstone of neural network research to the present day... Some physicists unfamiliar with the classical status of the additive model in neural network theory erroneously called it the Hopfield model after they became acquainted with Hopfield's first application of the additive model in Hopfield (1984)" - see Grossberg ([4], p. 23).

The subject of this paper is an application of additive neural network models to the control of dynamic processes. We formulate the tracking problem and show how the additive neural network model can be used as a controller. A variable structure systems approach ([1], [2], [8]) is utilised to construct the proposed controllers. This approach allows us to circumvent analysis problems caused by the discontinuous nonlinearity which describe neurons.

2. FORMULATION OF THE TRACKING PROBLEM

Suppose we have a model of a dynamic process, given by the following equations

$$\left.\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \end{aligned}\right\} \tag{2.1}$$

where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, and $C \in \mathbb{R}^{m\times n}$. We wish to design a controller so that the closed-loop system can track a reference input with zero steady-state error (see the Fig.). Let the reference signals be

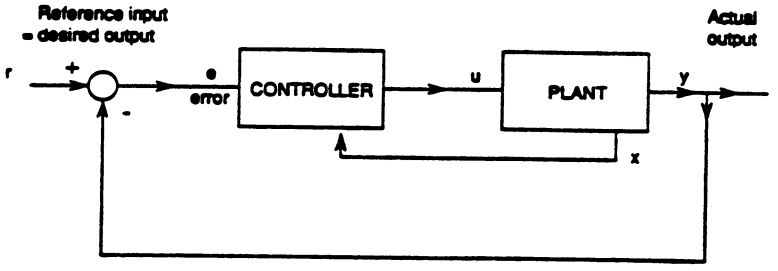


Fig. Tracking system structure.

described by the following differential equations

$$r_i^{(p)}(t) + \alpha_p r_i^{(p-1)}(t) + \ldots + \alpha_2 \dot{r}_i(t) + \alpha_1 r_i(t) = 0\,, \quad i = 1,2,\ldots,m$$

where the initial conditions $r_i(0)$, $\dot{r}_i(0)$, ..., $r_i^{(p-1)}(0)$ are specified.

The tracking error is defined as $e(t) = y(t) - r(t)$. The problem of tracking $r(t) = [r_1(t),\ldots,r_m(t)]^T$ can be viewed as a designing exercise of a control strategy which provides regulation of the error (Franklin et al. [3], p. 390), that is, the error $e(t)$ should tend to zero as time gets large. One way to tackle this

problem is to include the equations which are satisfied by the reference signal as a part of the control, (stabilization), problem in an error state space, thus reducing the tracking problem to the problem of stabilization of a class of systems modeled by (2.1). In what follows, we propose two new approaches to designing stabilizing control strategies for (2.1) involving the Hopfield type neural networks. Prior to presenting these control laws we will introduce the necessary apparatus for further analysis.

3. BACKGROUND RESULTS

In the following analysis we utilize certain concepts from the theory of variable structure control. This theory rests on the concept of changing the structure of the controller in response to the changing states of the system to obtain a desired response. This is accomplished by the use of a high speed switching control law which forces the trajectories of the system onto a chosen manifold, where they are maintained thereafter. The system is insensitive to certain parameter variations and disturbances while the trajectories are on the manifold. An important concept in variable structure control is that of an attractive manifold on which certain desired dynamical behavior is guaranteed. Trajectories of the system should be steered towards the manifold and subsequently constrained to remain on it.

Definition 3.1. ([8]). A domain Δ in the manifold $\{x \mid \sigma(x) = 0\}$ is a sliding mode domain if for each $\epsilon > 0$ there exists a $\delta > 0$ such that any trajectory starting in the n-dimensional δ-neighborhood of Δ may leave the n-dimensional ϵ-neighborhood of Δ only through the n-dimensional ϵ-neighborhood of the boundary of Δ.

We next describe the manifold which is used in this paper. Suppose

$$S = \begin{bmatrix} s_1 \\ \vdots \\ s_m \end{bmatrix} \in \mathbb{R}^{m\times n} ,$$

where $s_i \in \mathbb{R}^{1\times n}$. We assume that S is of full rank.

Let

$$\sigma(x) = [\sigma_1(x),...,\sigma_m(x)]^T = [s_1x,...,s_mx]^T = Sx ,$$

where $x \in \mathbb{R}^n$, and let $\Omega = \{x \mid \sigma(x) = 0\}$. Consider the system modeled by (2.1). We make the following assumptions:

Assumption 1: The matrix SB is nonsingular.

Assumption 2: The pair (A,B) is completely controllable.

Definition 3.2. The solution of the algebraic equation in u of $S\dot{x} = SAx + SBu = 0$ is called the equivalent control and denoted by u_{eq}, that is, $u_{eq} = -(SB)^{-1}SAx$.

Definition 3.3. The equivalent system is the system that is obtained when the original control u is replaced by the equivalent control u_{eq}, that is, $\dot{x} = [I_n - B(SB)^{-1}S]A\,x$.

We will now briefly discuss a method for designing of the switching surface. The method is based on that of El-Ghezawi et al. [2]. See also [6]. Certain relations which come out during the analysis of this method are instrumental in the construction of the state transformation discussed in the following Section.

Our goal is to choose S so that the nonzero eigenvalues of A_{eq} are prescribed negative real numbers and the corresponding eigenvectors $\{w_1,...,w_{n-m}\}$ are to be chosen. Let $W = [w_1...w_{n-m}]$; note that $W \in \mathbb{R}^{n\times(n-m)}$. In sliding mode, the system is described by $\dot{x} = A_{eq}x$, $\sigma(x) = Sx = 0$. The order of the system is n−m and the solution must be in the null space of S, that is, $SW = 0$.

Denote by R(T) the range of the operator T. Since we requires SB to be nonsingular and $SW = 0$, we must have $R(B) \cap R(W) = \{0\}$. It then follows that we should choose the generalized inverses B^g, W^g of B, W so that ([2])

$$B^gW = 0 \tag{3.1}$$

and

$$W^{\varepsilon}B = 0 \,. \tag{3.2}$$

We choose $\{w_1,...,w_{n-m}\}$ so that (3.2) holds. We can now construct S. Let $W^{\perp} \in \mathbb{R}^{m\times n}$ be any full rank annihalator of W, that is $W^{\perp} W = 0$. Since a necessary condition for $Sx = 0$ to be a switching surface is $SW = 0$, we see that $\Gamma W^{\perp}$ for any nonsingular $\Gamma \in \mathbb{R}^{m\times m}$ is a candidate. We also require that $SB = I_m$. We let $\Gamma = (W^{\perp} B)^{-1}$ and let $S = \Gamma W^{\perp}$. It is easy to see that $SB = I_m$ and hence $(W^{\perp} B)^{-1} W^{\perp}$ is a generalized inverse of B. If we let $B^{\varepsilon} = S$ in (3.1), the condition is satisfied.

We will utilize the results of this section to construct a state-space transformation which facilitates the design of neural controllers.

4. DESIGNING OF NEURAL CONTROLLERS

In this Section, we first introduce a transformation which brings the closed-loop system into the new coordinates in which the neural structure of the controller is revealed.

Let $M \in \mathbb{R}^{n\times n}$ be defined, as in Madani-Esfahani et al [6], by

$$M = \begin{bmatrix} W^{\varepsilon} \\ S \end{bmatrix},$$

where W^{ε} is defined by (3.2). Note that M is invertible with $M^{-1} = [W \;\; B]$. Introduce the new coordinates $\hat{x} = Mx$. Let $z = W^{\varepsilon}x$ and $\varsigma = Sx$. Then $\hat{x} = [z^T, \varsigma^T]^T$. In the new coordinates, the system becomes $\dot{\hat{x}} = MAM^{-1}\hat{x} + MBu$. We write

$$MAM^{-1} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix},$$

where $A_{11} \in \mathbb{R}^{(n-m)\times(n-m)}$, $A_{22} \in \mathbb{R}^{m\times m}$. Note that

$$MB = \begin{bmatrix} 0 \\ I_m \end{bmatrix},$$

where I_m is an m×m identity matrix. Hence

$$\begin{aligned} \dot{z} &= A_{11} z + A_{12}\varsigma \\ \dot{\varsigma} &= A_{21} z + A_{22}\varsigma + u \,. \end{aligned} \tag{4.1}$$

We analyze the closed-looped system (4.1) with the controller proposed by Madani-Esfahani et al [6]

$$u = -\,[\mu_1 \mathrm{sgn}\sigma_1,...,\mu_m \mathrm{sgn}\sigma_m]^T \,. \tag{4.2}$$

where

$$\mathrm{sgn}\,\sigma_i = \begin{cases} 1 & \text{if } \sigma_i > 0 \\ 0 & \text{if } \sigma_i = 0 \\ -1 & \text{if } \sigma < 0 \,, \end{cases}$$

and $\mu_i > 0$. For convenience, we let $D = \mathrm{diag}[\mu_1,...,\mu_m]$ and $\mathrm{sgn}\sigma = [\mathrm{sgn}\sigma_1,...,\mathrm{sgn}\sigma_m]^T$. We can now write (4.2) as

$$u = -\,D\,\mathrm{sgn}\sigma = -D\,\mathrm{sgn}\varsigma \,. \tag{4.3}$$

Combining (4.1) and (4.3) yields

$$\left.\begin{aligned} \dot{z} &= A_{11} z + A_{12}\varsigma \\ \dot{\varsigma} &= A_{21} z + A_{22}\varsigma - D\,\mathrm{sgn}\,\varsigma \end{aligned}\right\} \tag{4.4}$$

Note that the subsystem

$$\dot{\varsigma} = A_{21} z + A_{22}\varsigma - D\,\mathrm{sgn}\,\varsigma$$

which can be interpreted as a dynamic controller driving the dynamic system $\dot{z} = A_{11} z + A_{12}\varsigma$ has a

structure of an additive neural network model. Although we arrived at (4.4) starting with the controller (4.3) whose structure does not correspond to an additive neural network model, we can utilize the above analysis in the case when we explicitly apply a neural control strategy. We then proceed as follows. Suppose we are given a dynamic system model $\dot{x} = Ax + Bu$. We apply an additive neural network control law

$$\dot{u} = Fu - D \text{ sgn } \sigma(x,u) + Gx , \tag{4.5}$$

where $F \in \mathbb{R}^{m\times m}$, $G \in \mathbb{R}^{m\times n}$ are design parameters. The closed-loop system is

$$\begin{bmatrix} \dot{x} \\ \dot{u} \end{bmatrix} = \begin{bmatrix} A & B \\ G & F \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} + \begin{bmatrix} 0 \\ I_m \end{bmatrix} (-D \text{ sgn } \sigma(x,u)), \tag{4.6}$$

where $[x^T,u^T]^T \in \mathbb{R}^{n+m}$, and $\sigma(x,u)$ is a switching surface to be chosen. Using the approach presented in Section 3 we design the switching hyperplane $\sigma(x,u)$ and then construct the transformation M. In the new coordinates (4.6) will have the form (4.4), where now $A_{11} \in \mathbb{R}^{n\times n}$, $A_{22} \in \mathbb{R}^{m\times m}$, $z \in \mathbb{R}^n$ and $\varsigma \in \mathbb{R}^m$. We know that the above procedure yields a stable closed-loop system. However, we are also interested in the extent of the stability properties of the closed-loop system. The issue of the estimation of stability regions of dynamic systems driven by the neural network controllers is discussed in [9].

5. CONCLUDING REMARKS

In this paper we investigated viability of employing controllers based on additive neural network models to the problem of stabilization (tracking) of a class of dynamic systems. Two approaches to designing stabilizing controllers were proposed. Elements of the variable structure control theory were utilized to construct such controllers. The proposed controllers are characterized by robustness property which is inherent in the variable structure controllers. An important role in the analysis was played by a special state space transformation. This transformation helped us to utilize additive neural network models in designing stabilizing controllers. The proposed approach is promising in two ways. First, it results in robust controllers. Second, it has a potential to be employed in constructing fault tolerant controllers. Also generalizations to the control of a more general class of dynamic systems are feasible.

REFERENCES

[1] R. A. DeCarlo, S. H. Żak, and G. P. Matthews, "*Variable structure control of nonlinear multivariable systems: A tutorial,*" Proceedings of the IEEE, Vol. 76, No. 3, pp. 212-232, March 1988.

[2] O. M. E. El-Ghezawi, A. S. I. Zinober, and S. A. Billings, "*Analysis and design of variable structure systems using a geometric approach,*" Int. J. Control, Vol. 38, No. 3, pp. 657-671, 1983.

[3] G. F. Franklin, J. D. Powell, and A. Emani-Naeini, "*Feedback Control of Dynamic Systems,*" Addison-Wesley, Reading, Massachusetts, 1986.

[4] S. Grossberg, "*Nonlinear neural networks: Principles, mechanisms, and architectures,*" Neural Networks, Vol. 1, No. 1, pp. 17-61, 1988.

[5] J. J. Hopfield, "*Neurons with graded response have collective computational properties like those of two-state neurons,*" Proc. Natl. Acad. Sci. USA, Vol. 81, pp. 3088-3092, May 1984.

[6] S. M. Madani-Esfahani, S. Hui, and S. H. Żak, "*On the estimation of sliding domains and stability regions of variable structure control systems,*" Proc. 26th Annual Allerton Conf. Communication, Control, and Computing, Monticello, Illinois, pp. 518-527, Sept. 28-30, 1988.

[7] 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. Circuits Syst., Vol. CAS-33, No. 5, pp. 533-541, May 1986.

[8] V. I. Utkin, "*Variable structure systems with sliding modes,*" IEEE Trans. Automat. Contr., Vol. AC-22, No. 2, pp. 212-222, April 1977.

[9] S. H. Żak, S. M. Madani-Esfahani, and S. Hui, "*Control of Dynamic Systems via Neural Networks,*" Technical Report TR-EE 89-33, School of Electrical Eng., Purdue Univ., W. Lafayette, IN, July 1989.

Dynamic Digital Satellite Communication Network Management by Self-Organization [1]

Nirwan Ansari and Yizhong Chen

Center for Communications and Signal Processing
Electrical and Computer Engineering Department
New Jersey Institute of Technology
Newark, New Jersey 07102, USA

Abstract: Telecommunication network management has several tasks. Among which traffic management is one of the most important and difficult tasks. New attempts to automate this task such as expert system based approach are yet to be perfected. In this paper, we propose a neural network model to realize this task. This model is a derivation of Kohonen's self-organization model. Experimental results support the feasibility of our approach to automate the task.

1. Introduction

In a telecommunication network, traffic management is concerned with maximizing network efficiency when the network is under stress either due to overload or failure. The purpose of traffic management is to best meet the communication requirement of users under the constraint of a fixed network capacity. In other words, the probability that users cannot go through should be the lowest. The response and execution time is a crucial factor to prevent traffic lose and degradation of the network. The communication traffic operator (human or machine) must analyze and monitor the network status constantly, and react promptly to configure the network to meet various network requirements.

A typical communication network consists of thousands of channels. Managing such a network is a very difficult task. In our study, we only consider Time Division Multiple Access (*TDMA*) digital satellite communication network [1,2] because such network is more easily configurable as opposed to terrestrial communication network.

Followed by a brief description of TDMA digital satellite communication network, we will discuss our Neural Network model for traffic management in Section 3. We will present experimental results in Section 4, and finally conclude with a discussion of future work in Section 5.

2. TDMA Digital Satellite Communication Network

Time Division Multiple Access (*TDMA*) is a multiple access technique used in digital satellite communication. It enables a number of earth stations to share a satellite's capacity by allocating each earth station a time slot to communicate. The number of channels assigned to a station defines the capacity of the station. The TDMA digital satellite communication network has a mesh topology which allows direct station-to-station communication. The communication established between two stations is known as a *communication link*. The capacity of each link is determined by the number of channels assigned to the link. The total number of channels allocated to each link in the network thus defines the capacity of the network.

We define the load and demand of a network as the state of the network. The state of the network is quantified by the number of channels allocated to each link of the network. The capacity of the whole network is a constant, but the capacity of each link should vary according to the state of the network. An assignment of channels to links within a network topology is called a *map*. Currently, the TDMA digital satellite communication network (without Demand Assignment Multiple Access device) is configured to different maps either according to a programmed time table or via an experienced human operator monitoring the state of the network. We will describe a neural network which will adaptively automate the " channel assignment " of the TDMA digital satellite communication network.

We pose the " channel assignment " problem of the TDMA digital satellite communication network as a pattern recognition and generation problem. During the network operation, the state of the network fluctuates constantly. A pattern which reflects the state of the network for a period of time is derived from the Common Signaling Channel(*CSC*) of the satellite system. Based on the statistical analysis of the service requirement of the network, we initially select N maps to which the network can configure. An exemplar pattern is associated with each map. The exemplar pattern of a map represents the typical state of the network for a period of time during which the map is used. Using a neural network, each pattern derived from the CSC at a given time is compared to each of the exemplar patterns. Depending on how close the pattern is matched to each of the exemplar pattern, one of the following three actions will be taken:

1. The original map is used if the input pattern is close to the exemplar pattern associated with the original map.
2. A map corresponding to a different exemplar pattern which resembles the input pattern is used.

[1]This work has been partially supported by the New Jersey Department of Higher Education through NJIT Separately Budgeted Research.

3. A new map is generated to replace one of the existing maps if the input pattern is remotely different from any of the existing exemplar patterns.

In this paper, we assume that N is fixed.

3. The Neural Network Model

The Neural Network model (N-N model) consists of three layers as shown in Figure 1. The first layer performs the pattern recognition task. In this layer, the N-N model determines the exemplar pattern (map) which resembles most closely to the input data. The second layer analyzes the discrepancy between the chosen exemplar pattern and the input data, and generates intermediate maps which gradually deviate from the exemplar pattern and converge closely to the input data pattern. A new map is finally generated by the third layer.
Denote:

L as the total number of links in a network,

CA as the total number of channels in a network,

$R_i(t)$ as the number of channels in link i required by users at time t, and

$C_{ij}(t)$ as the total number of channels assigned to link i of the jth exemplar map at time t. Here, the parameter t indicates that exemplar map j is time varying.

In this study, we assume that the original N exemplar maps are given. We also assume that the state of the network is updated constantly from the Common Signaling Channel (CSC) of the satellite system, and converted to numerical values denoted by $R_i(t)$. To perform the recognition task, the state of the network, $R_i(t)$, is updated as input data to the 1st layer of the N-N model. We compute the distance (metric) between the input data, $R_i(t)$, and each of the exemplar maps as follows:

$$D_j(t) = \sum_{i=1}^{L} |D_{ij}(t)| \quad (j = 1, 2, ..., N), \tag{1}$$

where $D_{ij}(t) = R_i(t) - C_{ij}(t) \quad (i = 1, 2, ..., L \text{ and } j = 1, 2, ..., N)$.

$D_{ij}(t)$ indicates the busyness of link i in the network if exemplar map j is used. $Dij(t) > 0$ implies that link i of the network is overload when map j is used; i.e., channels allocated to link i of map j do not meet those demanded by the users in link i. $D_{ij}(t) \leq 0$ means that link i of map j provides more than enough channels required by the users in link i. $D_j(t)$ thus indicates the resemblance between the input data $R_i(t)$ and exemplar map j. In our N-N model, instead of using $D_j(t)$ which is used in the self-organization model [3,4], we use the normalized distance, $D'_j(t)$, as our metric, where

$$D'_j(t) = \sum_{i=1}^{L} \frac{|D_{ij}(t)|}{R_i(t)} \quad (j = 1, 2, ..., N). \tag{2}$$

The normalized distance is used so that the pattern recognition task will select the exemplar map which distributes the load of network most evenly. The exemplar map that best meets the requirement of the network is the one that yields the smallest normalized distance, $D'_j(t)$.

The exemplar map selected by the first layer of the N-N model may not satisfy the requirement of the network. It is the task of the second layer to modify the exemplar map to better meet the requirement of the network. Note that only the exemplar map selected by layer 1 is modified. We call the resulting modified map produced by the second layer an *intermediate* map. This process is governed by the following equation:

$$C'_{ij}(t) = C_{ij}(t) + \beta(n(j))D_{ij}(t) \quad (i = 1, 2, ..., L), \tag{3}$$

where $C'_{ij}(t)$ is the number of channels of link i of the modified map j, and $\beta(n(j)), 0 < \beta(n(j)) < 1$, is a gain factor. $n(j)$ is the number of occurrence of exemplar map j being selected. When a new exemplar map j is generated by the third layer, this number is reset to 0. $\beta(n(j))$ is a decreasing function. We use

$$\beta(n(j)) = k_1^{-(n(j)+k2)}. \tag{4}$$

The gain factor $\beta(n(j))$ has three effects. It affects the drift of map j being modified to converge to the input pattern. It also determines whether a new map j will be generated in the third layer, which will be discussed shortly. Finally, it affects how fast a new map is generated.

In the third layer of our N-N model, we compute a convergence factor, $E(j)$, to decide whether a new map should be generated to replace the exemplar map. We define

$$E(j) = Max\{|\beta(n(j))D_{ij}(t)|\} \qquad (i = 1, 2, ..., L). \tag{5}$$

From Equation 5, $E(j)$ is getting smaller when the jth map has been selected more often, indicating that the modified (intermediate) map j is reaching more closely to the network requirement. A new map is thus generated based on the following conditions:

If $E \geq r$,

$$C_{ij}(t+1) = C_{ij}(t) \quad (i = 1, 2, ..., L). \tag{6}$$

Else

$$\begin{aligned} C_{ij}(t+1) &= C'_{ij}(t) \quad B \leq CA \quad (i = 1, 2, ..., L), or \\ C_{ij}(t+1) &= \frac{CA}{B} C'_{ij} \quad B > CA \quad (i = 1, 2, ..., L), \end{aligned} \tag{7}$$

where $B = \sum_{i=1}^{L} C'_{ij}(t)$.

From Equations 6 and 7, if the jth map has been selected often enough (i.e.,$\beta(n(j))$ small), we generate a new map j to replace the original chosen exemplar map. If the capacity of the modified map exceeds the capacity of the network, the modified map is normalized as shown in Equation 7.

4. Experimental Results

We simulate a simple TDMA digital satellite communication network with 10 links and a capacity of 1000 channels. Ten original exemplar maps are given and shown in Table 1. The simulation result is shown in Table 2. In Table 2, the network requirement fluctuates slightly (random data) , from $t = 1$ to $t = 199$, and changes drastically at transition $t = 200$, and then varies slowly until $t = 1000$. Note that it takes approximately 150 iterations to generate a new map which converges closely to the requirement. Starting from transition $t = 200$, it takes less iterations to converge closely to the new requirement. This is because the difference between the requirement and the closest exemplar map in the latter case is small while the difference in the former case is large. For example, at t=1, the required channels in link 4 and link 5 are respectively 140 and 100 while those in the chosen exemplar map are 70 and 130, respectively. At $t = 200$, however, the number of channels required and the number of channels assigned in the chosen map are close.

5. Discussion and Future Directions

We have presented a simple example which demonstrates the feasibility of our algorithm for network management. We will further study the selection of some of the parameters such as k1, k2 and r used in our algorithm. These parameters affect the rate of convergence of new maps to the actual requirement. Effort will be made to test and improve our algorithm on real data which reflect actual activity of a network.

6. References

[1] Kamilo Feher, *Satellite/Earth Station Engineering*, Prentice-Hall, Englewood Cliffs, 1983.

[2] Spar Aerospace Limited, Communication Group, " Voice, Video, and Data Small Aperture Earth Terminals Using TDMA Technology," in *EXPO COMM '88 CHINA*, Beijing, China, October, 26-31, 1988

[3] T. Kohonen, *Self-Organization and Associative Memory*, Springer-Verlag, Berlin, 1984.

[4] Richard P. Lippmann, "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, pp.4-22, 1987.

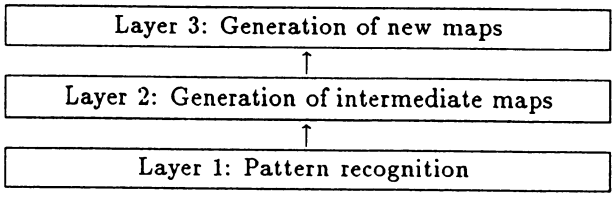


Figure 1: The Neural Networl Model. The task of Layer 1 is to select an exemplar map which resembles the input data most. Layer 2 modifies the chosen exemplar map. Layer 3 decides whether a map based on the intermediate map is generated.

Map No.	Link									
	1	2	3	4	5	6	7	8	9	10
Map 1	100	100	100	100	100	100	100	100	100	100
Map 2	120	80	120	80	120	80	120	80	120	80
Map 3	130	70	130	70	130	70	130	70	130	70
Map 4	150	50	150	50	150	50	150	50	150	50
Map 5	120	110	100	120	110	100	80	90	100	80
Map 6	80	90	100	80	90	100	120	110	100	130
Map 7	70	85	135	70	85	135	110	100	90	110
Map 8	60	140	100	60	140	70	110	110	120	100
Map 9	40	60	200	40	60	100	100	100	200	100
Map 10	140	60	140	60	140	60	200	0	200	0

Table 1: 10 Exemplar Maps, each with 10 links and a capacity of 1000 channels.

Network Activity and Map Selection		Link									
		1	2	3	4	5	6	7	8	9	10
t=1	Required	130	70	130	140	100	70	130	70	130	70
	Chosen (new) Map=3	130	70	130	70	130	70	130	70	130	70
	Intermediate Map	130	70	130	105	114	70	130	70	130	70
t=26	Required	131	69	132	140	100	69	130	69	131	71
	Chosen (new) Map=3	129	70	129	81	123	70	129	70	129	70
	Intermediate Map	129	70	129	88	120	70	129	70	129	70
t=150	Required	130	70	130	140	100	70	130	70	130	70
	Chosen (new) Map=3	125	68	125	133	96	68	125	68	125	68
	Intermediate Map	127	69	127	136	98	69	127	69	127	69
t=200	Required	120	130	100	120	110	100	80	110	100	60
	Chosen (new) Map=5	120	130	110	100	120	110	80	90	100	60
	Intermediate Map	120	120	100	120	110	100	80	100	100	60
t=213	Required	121	129	100	121	110	100	79	111	100	60
	Chosen (new) Map=5	119	118	99	119	109	99	79	98	99	59
	Intermediate Map	119	121	99	119	109	99	79	101	99	59
t=250	Required	120	130	100	120	110	100	80	110	100	60
	Chosen (new) Map=5	116	125	98	116	106	98	78	105	98	58
	Intermediate Map	118	127	99	118	108	99	79	107	99	59
t=1000	Required	120	130	100	120	110	100	80	110	100	60
	Chosen (new) Map=5	116	125	98	116	106	98	78	105	98	58
	Intermediate Map	118	127	99	118	108	99	79	107	99	59

Table 2: The results of applying our algorithm to a simulated network at various time. At each instance, it shows the activity of the network (the number of channels requested by the users), an intermediate map produced by Layer 2 of our model, and the (new) map chosen.

Diagnosis of Epilepsy via Backpropagation

B. APOLLONI[(o)], G. AVANZINI[(•)],
N. CESA-BIANCHI [(o)], G. RONCHINI[(o)]

[(o)] *Laboratorio di Reti Neuronali, Dip. di Scienze dell'Informazione*
Università di Milano, V. Moretto da Brescia 9, 20133 Milano (Italy)
[(•)] *Istituto Neurologico C. Besta, V. Celoria 11, 20133 Milano (Italy)*

1 Introduction

The term epilepsy designates a group of neurological disorders characterized by the recurrence of epileptic seizures. The category is by no means nosographically homogeneous since it includes pathological situations different as ethiology, natural history, prognosis and associated neurological signs. The diagnosis of epileptic syndromes via computer-based systems appears thus a challenging task because of the still many unknown relations among symptoms and their ethiological factors.

This study summarizes the outcomes of a thesis project [1] carried out by one of the authors and investigates the problem of epilepsy classification via Multi-Layer Perceptrons (MLP's) trained with the backpropagation algorithm [2]. We describe a MLP able to learn the diagnoses of a set of patients on the basis of a suitable coding of their case sheets. Since the diagnosis can be grouped into clusters depending on their reciprocal similarities, they have been coded so that similar diagnosis have similar codings. The trained network provides plausible generalizations (in a sense to be specified later) on unseen patterns in the 87% of cases. A simple technique for pruning redundant input units has also been developed. We observed that a pruned network with only the 10% of its initial input units could achieve the 95% of correct classifications on diagnosis clusters.

2 Epilepsy and its Classification

The need for a common international terminology and classification for epilepsy was finally met in 1985 by the Commission on Classification and Terminology of the International League Against Epilepsy (ILAE) [3].

The classification is aimed to identify clinical entities, i.e. epileptic syndromes, characterized by clusters of signs and symptoms customarily occurring together. In contradistinction to a disease, a syndrome does not necessarily have a defined ethiology. Some epileptic syndromes, however, are thought to be related to a specific ethiological factor (e.g. genetic) altough not yet defined. A more strict specificity is attributable to the different epileptic syndromes in terms of both natural history and prognosis: two clinical categories which are particularly relevant to the medical practice.

The ILAE classification has been acknowledged with interest by physicians and scientists working in this field who find it useful for evaluating the comparability of results and therapies. However, several drawbacks have to be pointed out. Syndromes may belong to different orders. Some represent rather broad concepts, others are much more specific. Overlapped syndromes and inclusion of one syndrome within another occur frequently.

For these reasons the ILAE classification is not intended as conclusive, but rather as a proposal to be tested for its usefulness. Along this way, at the Istituto Neurologico C. Besta, a computerized case sheet form has been developed to collect comprehensively all the clinical and laboratory informations which the ILAE Classification is based upon.

Data from 158 consecutive patients presenting epileptic syndromes dating back no more than 6 months and not associated with any detectable progressive brain process have been included in the present study. The aim of the present work is to compare the syndromic classification based on clinical criteria with the categorization achieved with a backpropagation trained MLP.

3 Network Topology and Training

The case sheet form was structured as a list of questions and has been initially coded with 814 bits. We gave a binary coding to the set of possible answers to each question in the form. Questions judged not relevant for epilepsy (e.g. some anagraphic informations) were excluded. According to the ILAE classification we considered 31 possible diagnosis. Since the diagnosis can be clustered into 7 groups by a relation of similarity we have coded each diagnosis so that the Hamming distance between the coding of two classes belonging to the same cluster is always lesser than the coding of two classes belonging to different clusters. After further selections on the questions in the form we ended up with 724 units and 31 units for input and output layer respectively.

Different activation functions have been experimented following [4] but best results have been reported when using the classical sigmoidal of the form: $f(x) = 1/(1 + e^{-ax})$. The setting of the parameters for the backpropagation is the result of an experimental tuning, the actual values are reported in Table 1.

Learning Rate	0.1
Momentum	0.1
Slope	1
Initial Weights Range	$-0.3 \div 0.3$
Error Threshold	0.3

Table 1

The set of 156 case sheets was divided in two subsets: 134 corresponding to reliable diagnoses and 22 corresponding to uncertain or "fuzzy" diagnoses. In a fuzzy diagnosis the doctor proposes a class but he gives also a low "membership degree" to that class. The reliable diagnoses have been used as training set while the fuzzy ones as test set for checking the generalization capabilities of the trained network. The training set was normalized in order to present with the same frequency each diagnosis to the network. Then the network has been trained until the differences between the expected and the current output for each output unit and on each element of the set were lesser than the error threshold.

In Figure 1 the graphs of the error function and of its gradient's modulo during the learning process are reported for a MLP with 50 hidden units.

4 Learning and Generalization Results

We considered MLP's with a variable number of hidden units: best performances were obtained with 50 units. Using a significantly lesser number of units (say 30) or greater (say 100) originated different problems.

The MLP with 30 hidden units could not learn the whole training set within the limits mentioned above. The two other MLP's (50 and 100 hidden units) could learn it but gave different results in the generalization test. In particular, we considered three possible outcomes for this test.

1. The network answer agrees with the fuzzy diagnosis of the doctor
2. The network gives a different hypothesis which is accepted by the doctor as a plausible alternative hypothesis
3. The network answer is either not plausible or "confuse".

Membership to a particular class was decreed by measuring the distance, with respect to a suitable metric, between the real vector $(x_1, x_2, \ldots, x_n)$ of output unit values and each binary vector representing the coding of the classes. The metric we used is the so-called city-block distance [5] (equivalent to a Minkowsky metric with $\lambda = 1$). We call "confuse" a configuration

of output units which does not match, within a small approximation, any coding of a class. Considering as valid generalizations any outcome of type 1 or 2, we observed that the MLP with 50 hidden units gave 20 valid generalizations (11 of type 1 and 9 of type 2) versus a total of 11 for the MLP with 100 hidden units.

5 Data Compression

An interesting problem related to the above work is the determination of which parts of the case sheet are really important for carrying out a correct diagnosis. Namely, we would like to select a subset of the questions in the case sheet form such that a MLP with a shorter input layer could obtain satisfying results too.

Figure 2 shows the distributions of weights of the full MLP with 50 hidden units after the learning is terminated. Interpretating the weights with a value close to zero as noise produced by the learning process, we have pruned the network by eliminating all the input units having no output weights larger than 0.5 in modulo. This amounted to a reduction of almost the 90% of input data and gave rise to a network with 74 input units. We remark that among the corresponding questions eliminated in the case sheet form there was no one judged essential for the diagnosis of epilepsy.

We have trained this pruned MLP on the same training set restarting the learning process with a new set of random weights. The generalization test gave approximately 80% of valid results on the single diagnosis and 95% on the clusters of diagnosis. This clearly shows the possibility, in this case, of working with very small networks without loosing too much in precision.

6 Conclusions

This work can be seen as an attempt to study neural networks as a tool able to validate the robustness of epilepsy classifications. The hypotheses to ascertain in this case were two. First, if the case sheet form contained all the relevant information for accomplishing the task. Second, if the classes of diagnoses proposed by ILAE were characterized strongly enough or some split and/or merge among classes was in order. The encouraging generalization results seem to indicate that the information contained in the case sheet and coded with the 724 bits was enough. On the other hand, the feasibility of data compression (as showed in the previous Section) may be accounted for by the presence of unimportant questions. An indicator of how strong a class gets characterized by a given example set could be the generalization test made on a set of (almost) sure diagnoses. If the network answers on inputs belonging to a particular class are always very close (in terms of the city-block metric mentioned before) to the coding of the class, then the characterization of that class is satisfying. A better assessment of this method could require a larger sample: we are currently gathering new case sheets on a national scale for improving the results.

Neural-based expert systems [6] [7] seem to overcome some of the drawbacks encountered in medical applications of rule-based expert systems. Beside these advantages, they are criticized mainly for their infeasibility with respect to the so-called explanation facility which is common in rule-based systems. We would like to point out how this feature is not essential for the study of medical diagnoses with neural networks since they involve other forms of knowledge representations not necessarily aimed to the extraction of rules.

REFERENCES

[1] G. Ronchini, "Classificazione dell'Epilessia tramite Reti Neuronali", Thesis, Università di Milano, 1989.

[2] D.E. Rumelhart, G.E. Hinton and R.J. Williams, "Learning Representations by Back-Propagating Errors", Nature, Vol. 323, 1986.

[3] Commission on Classification and Terminology of ILAE, "Proposal for Classification of Epilepsies and Epileptic Syndromes", Epilepsia, Vol. 26, pp. 268–278, Raven Press, New York, 1985.

[4] W.S. Stornetta and B.A Huberman, "An Improved Three-layer Backpropagation Algorithm", ICNN-87, Vol. 2, pp. 637–643, 1987.

[5] T. Kohonen, "Self-Organization and Associative Memory", 2nd edition, Springer Verlag, Berlin, 1988.

[6] D.G. Bounds and P.J. Lloyd, A Multi Layer Perceptron for the Diagnosis of Low Back Pain, ICNN-88, Vol. 2, pp. 481–489, 1988.

[7] K. Saito and R. Nakano, "Medical Diagnostic Expert System Based on PDP Model", ICNN-88, Vol. 1, pp. 255-262, 1988.

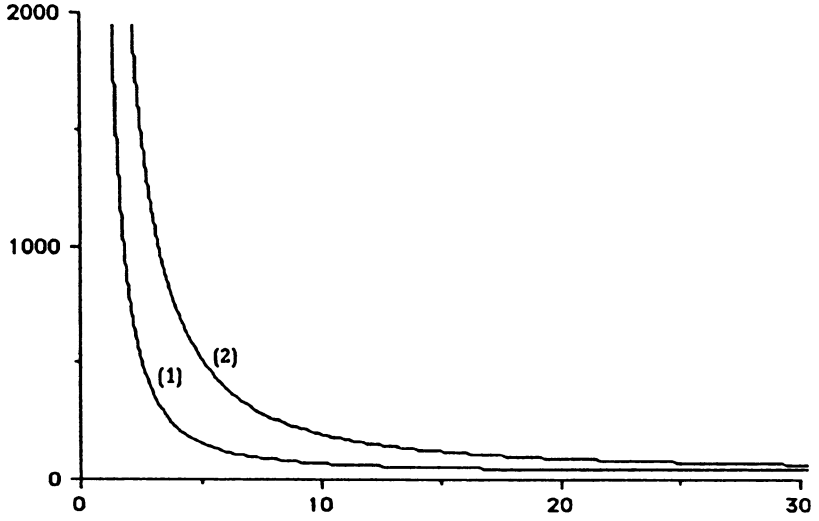


Figure 1.— Graphs of (1) error function and of (2) sum of absolute value weight changes.

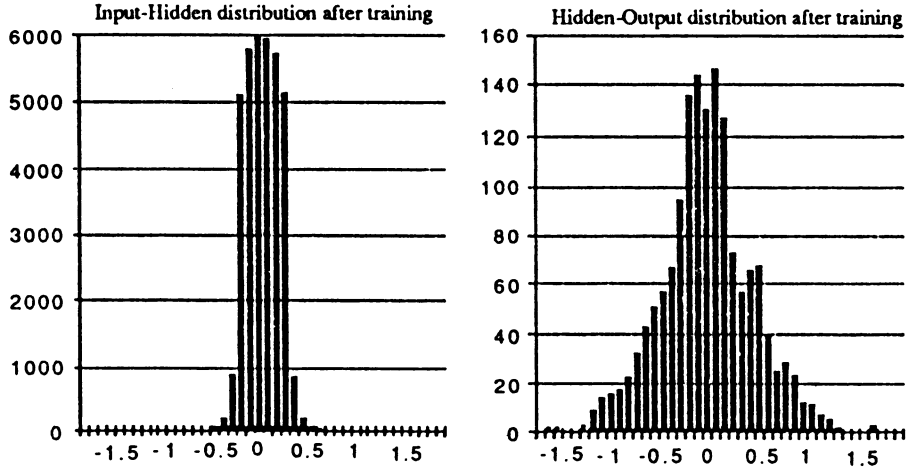


Figure 2.— Distribution of weight matrices after training.

A Connectionist Approach To The Processing Of Time Dependent Medical Parameters

Barry Blumenfeld, MD

Section of Medical Informatics
Department of Medicine
University of Pittsburgh
B50A Lothrop Hall
Pittsburgh, Pennsylvania 15261
Phone: (412) 648-3190

The representation of temporal knowledge, and reasoning with temporal information presents a continuing challenge to researchers in the field of Medicine. Several connectionist architectures specifically designed to deal with temporal information have been developed. Some of these paradigms treat time explicitly, representing flow either with an additional "time" parameter, or in the ordering of input vectors. Examples of networks of this type include Time Delay Networks (Sejnowski &Rosenburg,1986), Back-propagation in time (Rumelhart, Hinton & Williams), and image compression in a back propagation network (Cottrell, Munroe,Zipser,1987). A different approach has been to represent time implicitly in the functioning of the network itself rather than as an additional dimension of the input data (Jordan, 1986),(Elman,1988), (Mozer, 1988).

Networks that do not explicitly represent time confer several advantages over explicit representations. There is no limit to the number of intervals that can be examined sequentially, nor is there a need to explicitly specify the duration of each interval. Further, since a linear vector is typically used to represent a sequence of events in networks that model time explicitly, patterns that resemble each other but occur at different times cannot be easily recognized. This paper examines examine the performance of a particular type of network, called a Simple Recurrent Net (SRN) first introduced by Elman (Elman, 1988). He demonstrated that such a network was capable of mastering several problems involving the prediction of elements in a sequence, including the solution to a sequential version of the XOR problem, and the discovery of syntactic structure in natural language data. A network with the same architecture has since been trained as a finite state recognizer for a small grammar (Servan-Schreiber, Cleeremans, &McClelland,1988). Despite the simplicity of the architecture, these networks are capable of interpreting data in a context which can be quite complex. To make this point more clearly, and to demonstrate the power of this paradigm in a medical domain, the remainder of this paper will examine the performance of a recurrent net on a simple medical example.

The Domain

Assume that a diabetic patient is receiving an intravenous insulin infusion. The only information available about the patient is a stream of serial blood glucose levels taken at discrete intervals, and the current status of the insulin infusion. At each point in time, the physician (or network) is faced with the decision of either increasing the infusion, decreasing the infusion, or keeping it the same. At any point in time, the decision is based not only on the current blood glucose level, but also on the patient's past response to therapy. The absolute glucose level, its recent trends, and the trend as a function of the insulin dosage are all considered in adjusting the insulin infusion rate. We are given just two constraints:

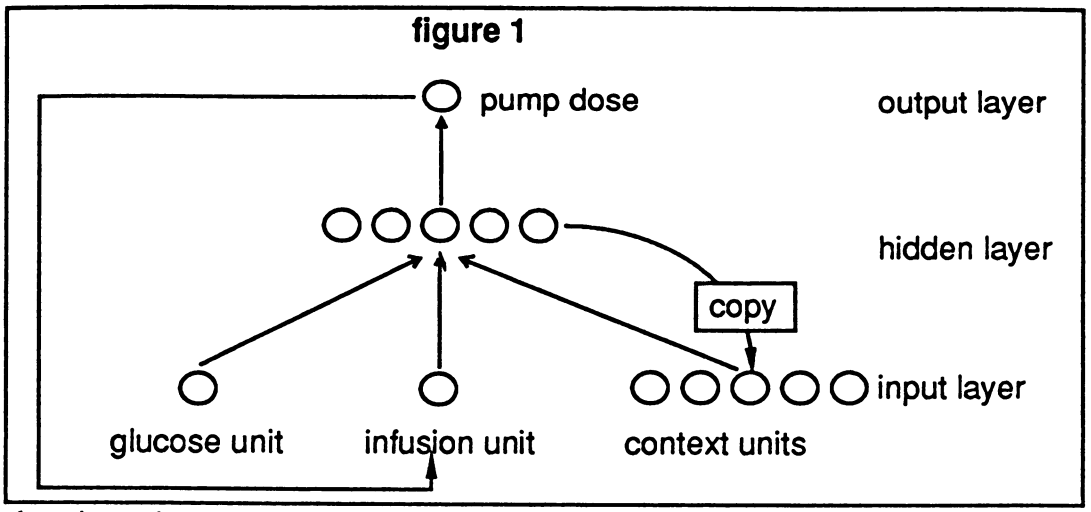
1. The blood sugar should fall at a rate of approximately 50-100 mg/dl/hour.
2. The blood sugar should never be allowed to fall below 100-150mg/dl.

The problem can be formulated geometrically as the task of finding a suitable trajectory through time while staying within these constraints. There exists an "ideal" trajectory, where, with a steady infusion rate, the blood sugar steadily declines at 50-100 mg/dl/hour starting at time zero.

The task of the network is defined as follows; when the sequence of blood sugars deviates from the ideal trajectory, the network should at each point propose a change in the insulin infusion dosage rate that will return the trajectory to a path parallel to the ideal, while staying within the limits of global constraints. Problems of this type have proved exceedingly difficult for conventional systems to solve, since they require the ability to recognize trend spanning variable periods of time. This situation has been modelled in the network detailed below

Network Description

The network used here, a back-propagation network with recurrent connections is a variation on a class collectively referred to as Simple Recurrent Networks (SRNs), first proposed by Elman(1988). It is shown in Figure one.



The input layer is supplemented by a group of units called "context" units. Actually, these units are also "hidden" in a sense, since they send and receive activations from the units in the hidden layer in the absence of external input. The Input Layer contains 7 units. The first two units are used to represent values of the glucose level and current pump dosage on an analog scale. The remaining five units are context units. The Hidden Layer contains five units with recurrent one to one connections to the context units. The Output Layer consists of a single unit representing a proposed pump dosage for the next time interval. All links between units in the network are trainable and weighted with real number values, with the exception of the recurrent links from the hidden units to the context units which are always set to 1. These values of the weights are set at the time of training and remain fixed during normal processing. All units have activations in the range [0,1]. In the input layer the activation of the first two units is set by the input to the network. The remaining five units, the context units, can either have their activations set directly, or receive input from the hidden layer.

Network Function

Processing proceeds as follows. The input layer is presented with sequences of vectors that represent a serum glucose and current pump setting at intervals as determined by an external clock. The output is a pump dosage in scaled absolute terms, that is proposed to keep the glucose falling at the proper rate over the next time interval. Note that there is no unit for representing the glucose in the output layer. The information is superfluous, since the "expected" glucose at the next step in time will always be approximately 100 mg/dl less than the current value. The interval occurring between each pattern is fixed, and is not specified by the network. All predictions made by the network are valid in reference to the interval with which it has been trained. To apply the output of the network to some other interval would require that all output be scaled appropriately.

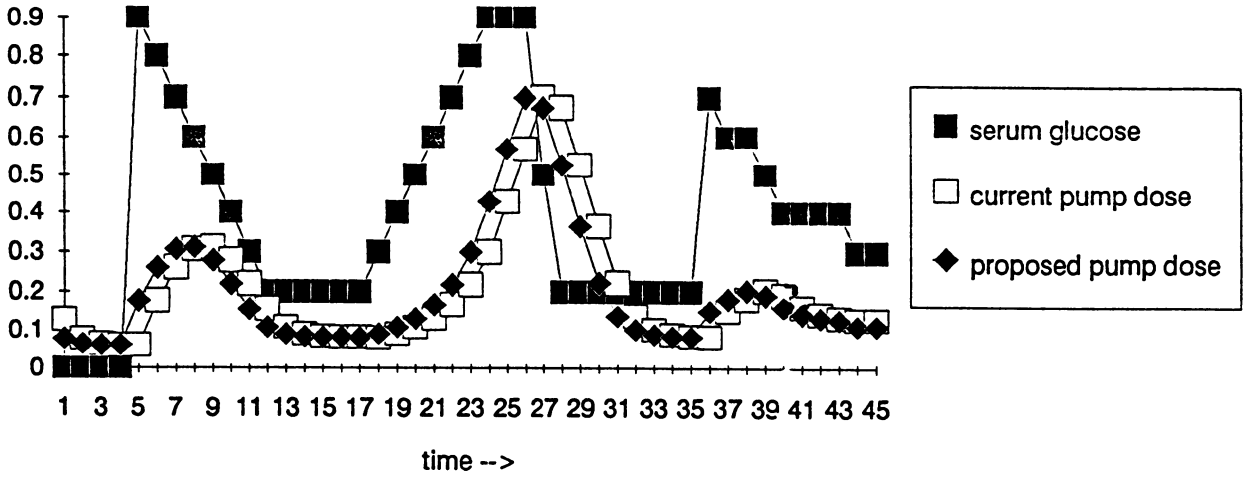
The Training Set

There are a potentially infinite number of sequences that might be presented to the network. However, it is not necessary to train the network on each of these possible sequences. What is really desired is to be able to respond to certain types, or classes, of state changes that represent "trends". Five prototypical types of changes were identified. Each of these prototypes represents a type of change, i.e. a rapid fall from high to low, or a steady high serum glucose. The network was trained by presenting each of the prototypes sequentially, with a "clear" sequence consisting of all zeros presented presented between each prototypical sequence. A total of two hundred epochs was required for adequate training.

Results

At any point in time, the network should output a pump dosage that will result in the glucose falling at the proper rate over the next time interval. The dosage suggested is influenced by both prior glucose levels and prior pump settings. To test the network, a sequence of 30 time steps during which the glucose was constantly varied was fed to the network. The network was also modified by adding a recurrent connection from the output node to the input node for current pump dosage. This simulates the closed loop situation were the network is really controlling the pump; the insulin dosage at every point in time is the dosage proposed by the network on the prior interval. The results of the test sequence are shown in figure two.

figure 2



The network suggested dosages properly throughout the sequence. It should be noted the the network was never trained on any part of the test sequence, rather, its performance represents a generalization of the behaviors learned on the training prototypes. During steps four to six, the glucose rises rapidly, and the network suggests a corresponding rapid rise in pump dose. When the glucose falls at a desirable rate in steps seven to twelve, the suggested dose is slowly stepped down, but when it falls too rapidly as in steps 26 to 28, a much sharper fall off in dose is suggested by the network. When the glucose plateaus at a desirable level of 200, the suggested pump dose plateaus as well. This is in contrast to a plateau at a high glucose value such as occurs in steps 23 to 27, where markedly higher pump doses are appropriately suggested. Testing the network on other lengthy sequences different from those it trained on always produced similar results.

Discussion

Many theoretical and practical obstacles remain before a network of this type can be developed that is clinically useful. Some of the problems relate to the way time is represented. Ideally, a network used in this setting should interpret all data as a continuous stream, with allowances made for the interval between the last measurement and the current data. Because the network architecture used represents time as a series of fixed intervals, all measurements must be taken at fixed intervals.

Although the performance of this network has been impressive thus far, there are also patterns that a network of this type simply cannot learn. An example of such a trajectory would be the case in which a patient's response to small changes in the insulin dose result in large swings in blood sugar. This will result in oscillating behavior by the network. A human observer would learn to underestimate the required insulin dosage after several cycles. The network will never learn to make this sort of compensatory behavior unless it is specifically trained on a such a sequence, and then only with great difficulty if the cycle occurs over a period several intervals.

Lastly, there is a problem that arises as the number of relevant parameters increases. The task of choosing an adequate training set can become intractable. Identifying a set of prototypes becomes more difficult with each variable that is

added. Alternatively, the network could be trained on real data, but this requires a huge corpus of cases to adequately cover the range of behavior that will be encountered. This is potentially the most serious challenge to further use of networks of this type.

Current Efforts

Several directions are currently being explored:

1. A larger, more robust set of prototypes is being developed in order to elicit more complex behaviors from the network.

2. A network with additional units that encode the interval between each time step is being tested. This would allow the network to modify its output based on the interval between each measurement.

3. A network model for a more complex domain is being developed.(controlling a ventilator in an ICU setting.)

REFERENCES

2.Cottrell GW, Munro PW, Zipser D. Image compression by back propagation: A demonstration of extensional programming. In N.E. Sharkey (Ed.), Advancesin cognitive science (Vol. 2). Chichester, England: Ellis Horwood, 1987.

3.Jordan MI. Serial order: A parallel distributed processing approach. Institutefor Cognitive Science Report 8604. University of California, San Diego, 1986.

4.Elman JL. Finding structure in time. CRL Technical report 9901. Center forResearch in Language, University of California, San Diego, 1988.

5.Rumelhart DE, Hinton GE, Williams RJ. Learning internal representations bybackpropagating errors. Nature 323:533-536.

6.Sejnowski TJ, Rosenberg C. NETalk: A parallel network that learns to readaloud. Technical Report, Johns Hopkins University JHU-EECS-86-01, 1986.

9.Mozer MC. A focused back-propagation algorithm for temporal patternrecognition. (Unpublished manuscript), 1988

Extraction of Semantic Features and Logical Rules from a Multilayer Neural Network

Laurent BOCHEREAU and Paul BOURGINE
CEMAGREF, BP 121
92164 Antony, France
Tel. (1) 40 96 61 21

1. Introduction

Neural networks are often considered as black boxes, in which knowledge is distributed and implicit in order to perform an associative memory. The purpose of this paper is to show how explicit knowledge such as logical rules can be extracted from a neural network. Previous works on this topic include [4, 9, 10]. After extraction of logical rules, a neural network becomes capable of explaining its reasoning as a classical expert system.

We consider multilayer neural networks where input and output neurons are booleans. We first study the semantic interpretation of a single neuron (§3) and then show how to build an "equivalent" network of logical and arithmetical rules (§4). An example concerning the first bid choice when playing bridge is presented afterwards (§5).

2. Multilayer Neural Networks

We are considering here feed-forward type multilayer networks with complete connections between adjoining layers [2, 3, 7]. The number of hidden layers between the input and output layer is arbitrary. The input-output relationship of each neuron (except for those belonging to the input layer) is given by :

$$y = f [net] \qquad \text{where} \quad net = \sum_{i=1}^{n} w_i x_i - \theta$$

where y represents the output of the neuron, n the number of inputs x_i, w_i the connection weights and θ the threshold. f is chosen to be the sigmoid function for each neuron's output function. We are given a set of exemples $\{ X_i, Y_i \}$ where X_i takes its values in the hypercube $\{0,1\}^n$ and Y_i the hypercube $\{0,1\}^m$. The network calculates a continuous mapping F from $\{0,1\}^n$ to $[0,1]^m$ by using the back propagation algorithm [6, 8] that consists of a gradient descent method to modify weights and thresholds in order to minimize the error between the desired output and the output signal of the network.

We now define smooth and strong neurons. A neuron is said to be smooth for the learning set if the number of input values belonging to the interval [-2,+2] is greater than or equal to the number of input values in $] -\infty , -2 [\cup] +2 , +\infty [$. A neuron is said to be strong if it is not smooth. An equivalent definition for a smooth neuron would be to consider the number of activation values in the interval $[f(-2) , f(2)]$; this allows us to consider the input neurons as strong neurons and also boolean variables. When applying neural networks with a limited number of hidden neurons to various problems, we observed that the hidden neurons tended to behave as very strong or very smooth. This led us to develop a method for designing networks such that the resulting hidden neurons discriminate in very strong or very smooth. The details of this method are out of the scope of this paper. One of the advantages of this method is to allow one, after the learning phase, to force the very strong neurons to become threshold neurons so that they can be interpreted as boolean variables. In the following, we will assume that we have threshold neurons and smooth neurons, which is not restrictive.

3. Semantic Interpretation of a Neuron

We first consider transformations of a neuron $f [w_0 + \Sigma_j w_j x_j]$ that keep its transfer function invariant, but allow better semantic properties. We will then explain in §4 transformations of the overall network.

3.1. Smoothing of the weights

When the number of x_js is large, it is possible to perform a histogram of the weights w_j and cluster the weights in several classes. When the classes are difficult to distinguish, one can use statistical methods such as discriminant analysis. By choosing an average coefficient for each class, we can derive an equivalent neuron $f [w_0 + \Sigma_k w_k (\Sigma_{jk} x_{jk})]$ where x_{jk} belongs to class k. In order to ensure that the error introduced by the averaging operation is negligeable, one can choose :

$$w_k = \frac{\Sigma_{jk} \frac{\delta E}{\delta w_{jk}} w_{jk}}{\Sigma_{jk} \frac{\delta E}{\delta w_{jk}}} \quad \Rightarrow \quad \Delta E \approx 0$$

under the conditions that :

$$w_k \in [\min_{jk} \{ w_{jk} \}, \max_{jk} \{ w_{jk} \}]$$

3.2. Appearance of intermediate numerical variables

If the neurons x_{jk}s of a same class k are strong neurons (this is always the case if they belong to the input layer), then the summation $\Sigma_{jk} x_{jk}$ can be interpreted as the number of x_{jk} equal to 1. We thus can introduce an intermediate numerical variable that will ease the further interpretation. An other case consists of performing a linear transformation on neurons belonging to the same class if we compensate by similarly introducing an intermediate numerical variable.

4. Extraction of Logical Rules Distributed on the Network

When considering an output neuron or a hidden neuron, one can extract an equivalent logical rule. If we assume that the neuron is activated at level s, we can write :

$$w_0 + \Sigma_k w_k (\Sigma_{jk} x_{jk}) \geq f^{-1} [s] \Leftrightarrow \Sigma_k w_k (\Sigma_{jk} x_{jk}) \geq f^{-1} [s] - w_0 \qquad (1)$$

If we assume that most of the x_{jk} are strong neurons and thus boolean (this is always the case when considering the first hidden layer), we can easily, by enumeration, extract a disjonctive normal form, by superposing all the cases with the OR connector : the derived logical formula is equivalent (at level s) to (1). We can afford to use an explicit enumeration algorithm because the number of variables has been greatly reduced as explained in §3.2. In any case, it is always possible to improve the implicit enumeration by using heuristic methods such as propagation/extraction techniques [1].

The validity domain of the methods presented above is limited to networks whose hidden layers consist of less than 10 neurons. The limitation on the number of input neurons is related to the existence of regularities in the coefficients (we considered networks with input layers containing until 52 neurons).

5. Application

We are considering the problem of the first bid choice when playing bridge. A method called "the new fith major" has been widely used [5] in order to determine the right first bid out of 11 candidates (from "pass" to "2 without trump"). This method consists of a large number of logical rules that are applied on several decision elements such as the number of honor points or the number of cards of the same suit. An expert system based on this method has been implemented and has provided us with 50 hands per bid. We considered a neural network with three layers : the first layer consisted of 52 neurons corresponding to a deck of cards ; the second layer was chosen to have 5 hidden neurons since there are five predominant features when estimating the value of an hand, i.e. the number of honor points and the number of cards in each of the four suits ; finally, the third and last layer contained 11 neurons corresponding to the possible bids.

After the learning phase, we observed that only one hidden neuron (N1) was performing as a smooth neuron whereas the four other neurons were strong neurons. Table 1 shows the results of the semantic interpretation of the five hidden neurons. The smoothing of the coefficients as presented in §3.1 allows one to cluster the input neurons in several classes. By considering the rank and the suit of each card, we can easily give a name to each of these classes : P_H = number of honor points ; N_T = number of clubs ; N_K = number of diamonds ; N_C = number of hearts ; N_P = number of spades. It is very interesting to notice that the network has been able to retrieve by itself the concept of points as well as the concept of suits - even though the neurons N2, N3 et N4 are superposing a concept of suit with the concept of points.

Neuron	Type	Input interpretation	Activation condition at level 0.5
N1	Soft	$-0.2\,P_H + 3.2$	$P_H < 16$
N2	Strong	$-2\,P_H + 5.4\,N_K$	$N_K > 0.37\,P_H$
N3	Strong	$P_H + 9\,N_C - 39$	$N_C > 4.3 - 0.11\,P_H$
N4	Strong	$-2\,P_H + 5.4\,N_P$	$N_P > 0.37\,P_H$
N5	Strong	$7.5\,N_T - 34$	$N_T > 4.8$

Table 1 : Activation conditions of the five hidden neurons at level 0.5

By a similar smoothing operation, we can derive the formulas giving the activation of the neurons "bid" with respect to the activation of the hidden neurons :

Pass = f [3.5 N1 + 0.5 (N2 + N3 + N4 + N5) - 4]
1 Club = f [2 N1 + 1.5 N5 - 0.5 (N2 + N3 + N4) - 2]
1 Diamond = f [N2 - N4 - 0.5 (N3 + N5) - 1]
1 Heart = f [3 N1 + N3 - 0.5 (N2 + N4 + N5) - 3]
1 Spade = f [N1 + N4 - N2 - 0.5 (N3 + N5) - 1.5]
1 Without Trump = f [8 N1 - 2 (N2 + N3 + N4 + N5) - 4]
2 Clubs = f [-7 N1 + 1.5 (N2 + N3 + N4 + N5) + 1.5]
2 Diamonds = f [-2 N1 + N2 - 0.5 (N3 + N4 + N5)]
2 Hearts = f [0.5 N1 + N3 - 1.5]
2 Spades = f [- 2.5 N1 + 1.5 N4]
2 Without Trump = f [- 3 N1 - (N2 + N3 + N4 + N5) + 1.5]

At this point, it is possible to interpret the overall network and extract logical rules. Let us take the exemple of the "2 Spades" bid. By using the result given above, we can write that the activation condition at level 0.5 is :

$$-2.5\ N1 + 1.5\ N4 \geq 0 \qquad (2)$$

Since N1 is a smooth neuron and N4 is a threshold neuron, we know that $N1 > 0$ and thus N4 must be equal to 1. Thus (2) becomes $N1 \leq 0.6 \Leftrightarrow P_H \geq 15$. This condition can be combined with the result N4 = 1 to extract from the condition given in table 1 the logical formula : "$P_H \geq 15$ and $N_P \geq 6$". It is important to notice that the logical rule is a function of the desired level of activation s (here 0.5). We, thus, can derive rules as functions of s. This allows us to extract the exact frontiers between bids by considering competitions between them at a given activation level. A systematical way of retrieving the rules consists of simulating the equivalent network by varying all the numerical variables and extract the frontiers from the results.

6. Summary

We have shown that some types of neural networks can be represented by "equivalent" logical and arithmetical formulas. Derived from an analogic machine working as a black box, a logical machine can sometimes be built that is able to explain its reasoning. The neural network answers as an expert and the equivalent network can be used to provide explanations as an expert system would do.

Moreover, logical and arithmetical formulas extraction provides one with an extra advantage: the formulas are not very sensitive to slight variations of the numerical coefficients; the calculation allows a better reproductibility. Thus, two similar neural networks learning from the same examples set will reach different numerical coefficients; however, the extracted formulas will be similar.

Acknowledgement

The authors thank Benoit Bonatre for providing the hands used in the training set.

References

[1] Bourgine P., "Propagation et extraction en résolution de problème", Colloque Intelligence Artificielle, Strasbourg, France, 15-19 september, 1986.
[2] Feldman S.E. and Ballard D.H., "Connectionist models and their properties", Cognitive Science, 1982, 6, 205-254.
[3] Fogelman-Soulie F., Gallinari P., Lecun Y., Thiria S., "Automata networks and artificial intelligence" in Automata Networks in Computer Science", F. Fogelman-Soulié, Y. Robert, M. Tchiente eds, Manchester University Press, 1987, 133-186.
[4] Gorman R. P. and Sejnowski T. J., " Analysis of hidden units in a layered network trained to classify sonar targets", Neural Networks, 1, 1988, 75-89.
[5] Jais P. and Lebel M., La Nouvelle Majeure Cinquième, Editions du rocher, 1982.
[6] Le Cun Y., "A learning procedure for asymmetric networks", Proceedings of Cognitiva 85, Paris, 1985, 599-604.
[7] Personnaz L., Guyon I., Dreyfus G., "Information storage and retrieval in spin-glass like neural networks", J. Physical letters, 1985, 46, L359- L365.
[8] Rumelhart D.E., Hinton G.E., Williams R.J., "Learning internal representations by error propagation" in Parallel Distributed Processing, D.E. Rumelhart & J.L. MacClelland eds, MIT Press, Volume 1: 318-369, 1986.
[9] Rumelhart D.E., Hinton G.E., Williams R.J., "Learning representations by back propagating errors", Nature, Volume 323, 9, october 1986, 533-536.
[10] Smolensky P., "Information processing in dynamical systems : foundations of harmony theory" in Parallel Distributed Processing, D.E. Rumelhart & J.L. MacClelland eds, MIT Press, Volume 1, Chapter 6, 1986.

PROBLEM-SOLVING BY USING REINFORCEMENT LEARNING NEURAL NETS

Victor C. Chen
VITRO Corporation
14000 Georgia Avenue, Silver Spring, MD 20906-2972

Abstract The traditional artificial intelligence approach to problem-solving is to design programs that will work out a step-by-step method, then carry out the steps. Without programming, neural networks are able to solve problems by learning. In this paper, we discuss a neural network approach to problem-solving and an appropriate neural network architecture. We chose the Tower-of-Hanoi puzzle as an example to demonstrate the problem-solving ability of reinforcement learning neural networks. We also propose a search approach which combines stochastic search with simulated annealing to significantly reduce the instability in the learning curve.

THE STATE REPRESENTATION

There are three important features common to problem-solving : states, operations, and goals. We chose the Tower-of-Hanoi puzzle as our example to illustrate our neural network problem-solving scheme. In the Tower-of-Hanoi puzzle as shown in Fig.1, there are three pegs, numbered 1,2,and 3. Initially, on peg 1 there are N disks, each one smaller than the one below it. The problem is to move all the disks from peg 1 to peg 3 by moving disks one at a time. Only the top disk on a peg can be moved, but it can never be placed on top of a smaller disk.

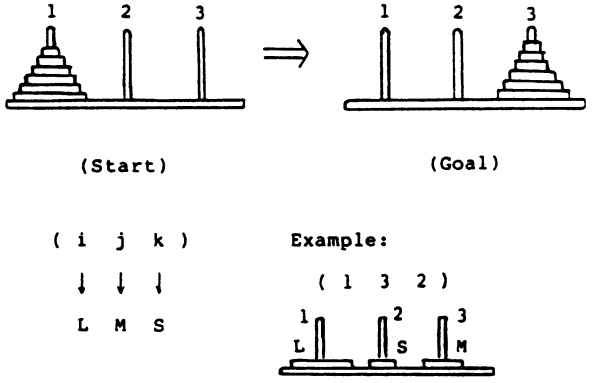


Fig. 1 The Tower-of-Hanoi Puzzle

Fig.1 also illustrates the three-disk problem. Let S be the small disk, M the medium disk, and L the large disk; then we can describe the state of the Tower-of-Hanoi by specifying which peg disk S is on, which peg disk M is on, and which peg disk L is on. We can then describe any state using three numbers: for example, (132). The first number tells which peg L is on; the second tells which peg M is on; and the third tells which peg S is on. The operation changes one of the numbers in the state description to reflect the moving. However, the operations cannot change the numbers arbitrarily. There are constraints: (1) Only the top-most disk can be moved from one peg to another, (2) A larger disk may never be placed on the top of a smaller disk, (3) Only one disk can be moved at a time.

SEARCHING FOR A SOLUTION

Heuristic search is a technique for searching the state space efficiently. It usually requires information about the properties of the specific problem domain. In heuristic search, decisions must be made as to which state is the most promising to expand next what operation to apply next to a given state.

The cost of a solution path in the state space is defined as the number of operations required to transform from the initial state into the goal state. Unfortunately, when a state is encountered during a search, we do not know if the lowest-cost solution path goes through the state. Usually an evaluation function is needed in order to estimate the promise of a state.

Strategy learning conducted under the influence of the evaluation is called reinforcement learning. This type of learning assigns credit to the system performance so as to increase the probability of receiving high evaluations. The learning procedure must : (1) generate operations, (2) assign credit to operations, and (3) modify the operation-selection process.

Operations can be generated by a searching procedure, such as stochastic search, to observe relationships among states, operations, and degree of success in achieving a goal. According to the evaluation at each operation step and using the heuristic rule, credit can be assigned to the searching process to indicate degrees of success. Guided by the assigned credit or blame, the operation-selection process will be modified. Langley [1985] discussed some heuristics for the assignment of credit to operations.

THE SYSTEM MODEL FOR PROBLEM-SOLVING

The general model of the problem-solving system consists of two parts : the evaluation part and the performance part. The evaluation part outputs an evaluation signal which criticizes the operation taken according to the search heuristics. The performance part consists of three elements : learning, the knowledge base, and the searching operation or action. The learning element has two inputs : the states of the environment, including the current state as well as previous states, and the evaluation value. According to the inputs, the learning element adjusts the knowledge base, and based on the adjusted knowledge, the searching operation element executes the next operation on the environment.

IMPLEMENTATION OF THE PROBLEM-SOLVING SYSTEM WITH NEURAL NETS

Both the performance part and the evaluation part can be implemented with neural networks. Following Barto *et al.* [1983] and Anderson [1987], one is called the action network and the other is called the evaluation network. The evaluation network will learn an evaluation function. According to the heuristics, credit or blame can be assigned to operations through an external critic signal, r(t). In order to get a continuous evaluation function value, the adaptive heuristic critic (AHC) algorithm [Sutton, 1984] can be employed. The evaluation net will perform the AHC algorithm to find the evaluation function, $\hat{r}(t)$. The most significant external critic signal, r(t)=1.0 (reward), is assigned whenever the goal state is reached. A large negative primary critic signal, r(t)=-1.0 (penalty) is assigned whenever a loop occurs. For all other non-goal states, the primary critic signal is

assigned to a small negative value, r(t)=-0.05 — -0.1 (a little penalty), to ensure the shortest path heuristic. The action network performs searching, and learning will be based on the estimated, time-varying, and delayed evaluation value from the evaluation network. Random noise, n_k, is added to each output $y_k(t)$ in order to perform stochastic search. As learning progresses, the action probabilities will be updated and become biased in favor of more successful actions. The difference between the actual action, act(t), and the expected action value, E{act(t)}, can be used as an error signal. Fig.2 shows the neural network architecture for problem-solving. In order to avoid re-visiting a previously visited state, the previous two actions should also be input to the action net in addition to the current environment state-vector.

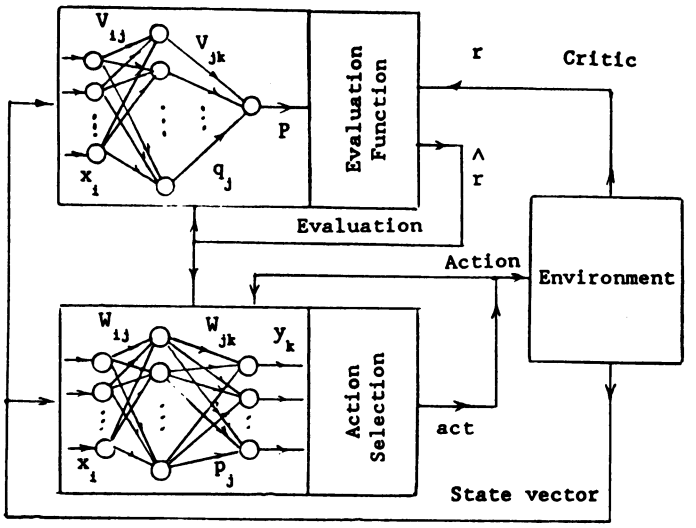


Fig.2 Neural Network Architecture for Problem-solving.

SIMULATION RESULTS

Fig.3(a) shows the learning curve of the system. Generally speaking, after 3,000 trials the system will learn the optimal solution path and the learning curve will converge to 7 steps.

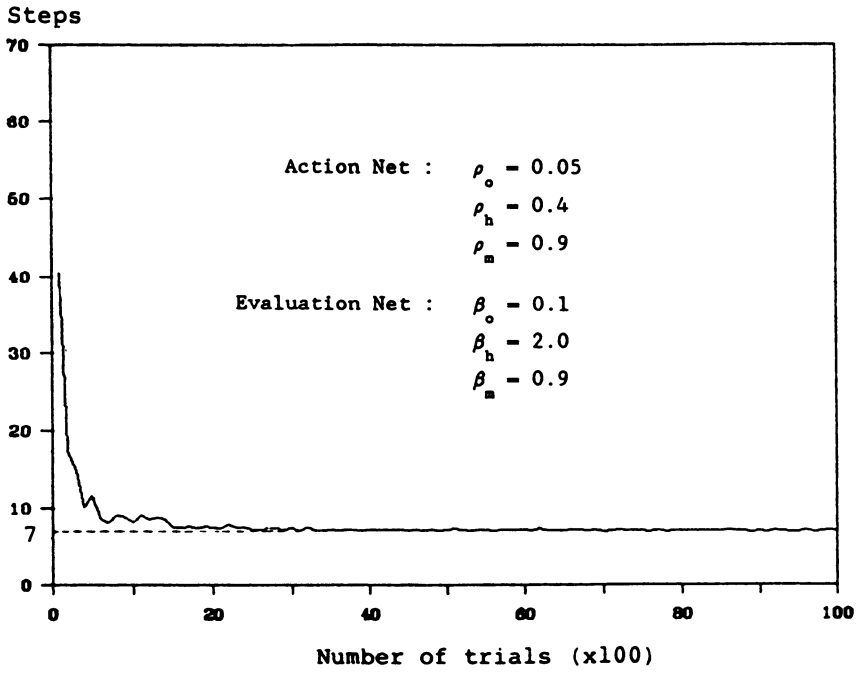
Because random noise is added to the outputs of the action network, it is still possible that the system will bias from the optimal solution path even after it learns the path. Fig.3(b) illustrates learning curves for solving a simple two-disk Tower-of-Hanoi problem. The dashed line indicates the learning curve using conventional stochastic searching. From the dashed line we can see that peaks can still take place even after the learning curve converges to the shortest step. However, when we combine the stochastic search with simulated annealing (i.e., gradually reducing temperature, therefore, the noise intensity), the learning curve will converge to the shortest steps constantly as indicated by the solid line in Fig.3(b).

REFERENCES

1. Langley, P., " Learning to search:from weak methods to domain-specific heuristics ", Cognitive Science, 9, pp.217-260, 1985.

2. Barto, A.G., Sutton, R.S., and Anderson, C.W., " Neuronlike elements that can solve difficult learning control problems ", IEEE Trans. on System, Man, and Cybernetics, 13, pp.835-846, 1983.
3. Anderson, C.W., " Strategy learning with multilayer connectionist representations ", Proceedings of the 4th International Workshop on Machine Learning, pp.103-114, 1987.
4. Sutton, R.S., " Temporal aspects of credit assignment in reinforcement learning ", Ph.D. Dissertation, University of Massachusetts, 1984.

(a)



(b)

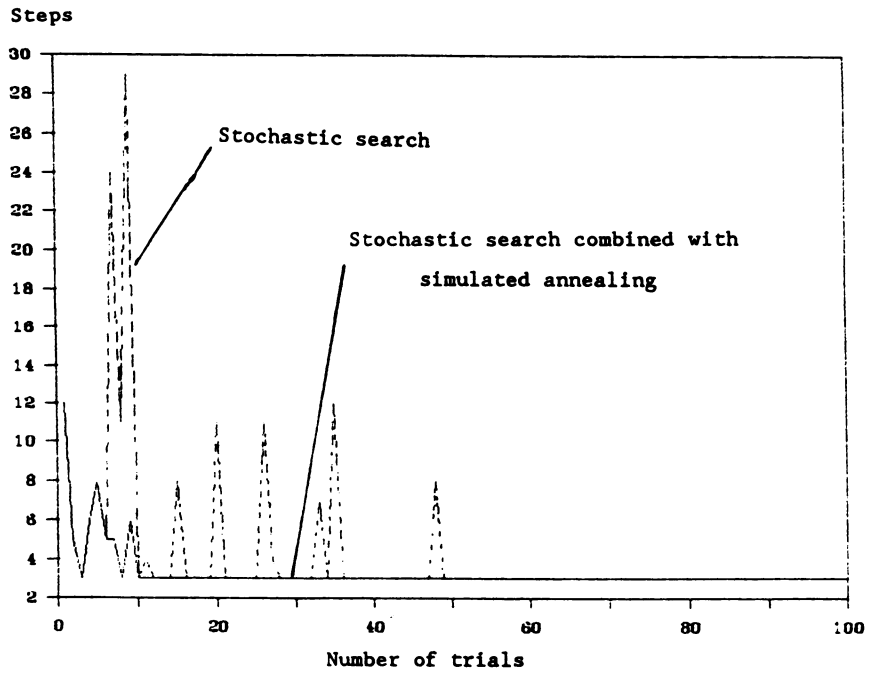


Fig. 3 (a) The Learning Curve of the Learning System.
(b) Learning Curves for Solving 2-desk Tower-of-Hanoi Puzzle.

COMPOSITE STOCK CUTTING PATTERN CLASSIFICATION THROUGH NEOCOGNITRON

Cihan H. Dagli M. Reza Ashouri Gary Leininger Bruce McMillin
Intelligent Systems Center, University of Missouri-Rolla
Rolla, Missouri 65401-0249

ABSTRACT

At present, the majority of manufacturing of aerospace composite laminates and the resulting cutting process is done manually often resulting in high scrap percentages. Automation of this process using artificial intelligence technology can reduce production lead time and manufacturing costs due to the high costs of composite materials. In this paper the use of a neocognitron system as a pattern classifier for a composite stock cutting process that integrates neural networks with the more traditional expert systems technology is discussed. The neocognitron architecture is extended to cover translation, rotation and scale invariant pattern recognition without additional processing to meet the requirements imposed by the composite stock cutting process.

INTRODUCTION

Currently, a large majority of firms have or are in the process of transferring their composite product design process into an electronic environment. In this area where computer graphics is extensive, there is a need for a system that can retrieve part drawings from the electronic environment and generate the best allocation of patterns to be cut from a specific plate of composite material. Once the cutting configuration is found it can be transferred to a computer integrated manufacturing environment in which cutting operations can be performed. Automating the pattern generation process is essential in reducing manufacturing lead time to provide the ability to meet frequent product design changes and short product life cycles. This necessitates an integrated approach to stock cutting problems. There is a need for an advanced decision support system that has the capability to process patterns in an electronic environment, select appropriate optimization algorithms and determine applicable allocation heuristics based on pattern configurations and stock sheets. Composite stock cutting is not an exception of this trend. The need for an automated composite stock cutting system becomes more important as the use of composite materials has increased dramatically over the past decade and will continue to increase in the military research and development areas. The drive to higher performance specifications and better efficiency in airframe structures will have a significant impact on the grouth of composite materials research and its acceptance in the manufacturing domain..

Dimensionality, shape of patterns, and number of plates are the basic attributes generally used to classify stock cutting problems. Even the two dimensional single plate rectangular pattern problem is NP-complete as shown by Garey (1979). The problem becomes more complicated for other combinations of attributes. Thus, instead of trying to reach an optimal layout pattern most researchers have turned to heuristic solutions that generated the above classification. Optimization is a critical elements in the solution of these problems. The methods proposed are summarized in survey articles by Hinxman (1980) and Golden (1976). Current techniques have restrictions in application due to the NP-complete nature of the problem. To overcome this difficulty various heuristic techniques and integrated artificial intelligence optimization approaches are proposed in Dagli (1990). Cutting pattern generation is an essential issue in developing a solution to this problem. Neural networks could contribute greatly to the design of such a system through their ability to identify various pattern configurations in the process of cutting pattern generation.

Most of the well known neural networks paradigms including backpropagation, counter-propagation, Hamming, etc. are based on a pattern matching criteria. These networks, when trained, form a number of decision regions. Each decision corresponds to one specific class. In the recall process, input patterns are classfied based on their distance from these decision regions. These pattern matching neural networks are to some extent tolerant of noise but if the input pattern is slightly shifted, scaled or rotated, they are unable to properly identify the reformed patterns. This is an important issue to be considered in cutting pattern generation for composites as patterns need to be shaped, scaled and rotated in the process of generating nests on the stock sheet. The neocognitron developed by Fukushima (1982) is a pattern recognizer based on feature extraction and feature matching. This network imitates the human visual system.

A neocognitron system consists of a number of modular stages. At the first stage, local features of the input pattern are extracted and at each succeeding stage they are gradually converted to the more global features. Finally at the last stage each cell represents a feature pattern which has global characteristics and is invariant to shift, scale and deformation, Fukushima (1988). A distinguishing characteristic of neocognitron system over the backpropagation and similar networks is the fast learning ability. It can learn a pattern in a few iterations and therefore is capable of real time learning. In this study the neocognitron is selected as a network paradigm and extended to meet the requirements imposed by composite stock cutting. The extended neocognitron architecture is demonstrated on example patterns incorperating essential stock cutting problem features.

NEOCOGNITRON

The basic structure of the neocognitron system is shown in Figure 1. This multistage network consists of many stages. Each stage contains a layer of S-cells followed by a layer of C-cells. The input pattern Uo is applied to the first stage.

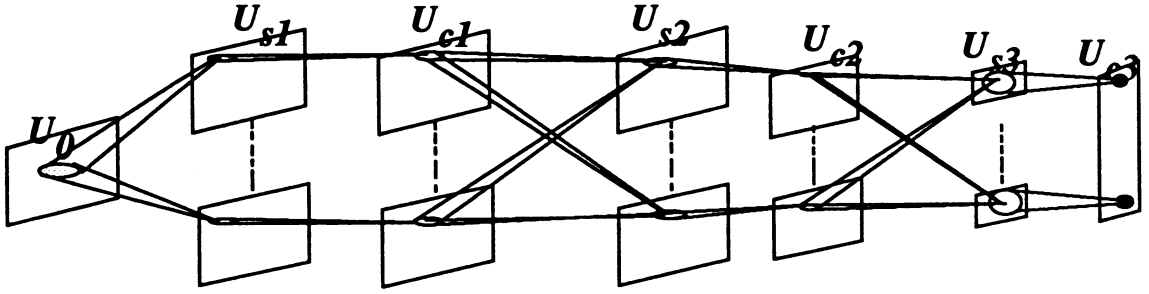


Figure 1. Structure of the neocognitron

S-cells are feature extracting cells. In other words, an S-cell is activated only when a particular feature is presented at a certain position in the input layer. The feature which the S-cells extract is determined during the learning process. In lower stages, local features such as a line at a particular orientation are extracted. In higher stages, more global features, such as a part of a training pattern are extracted. Figure 2 shows a S-cell input to output characteristic employed in the neocognitron.

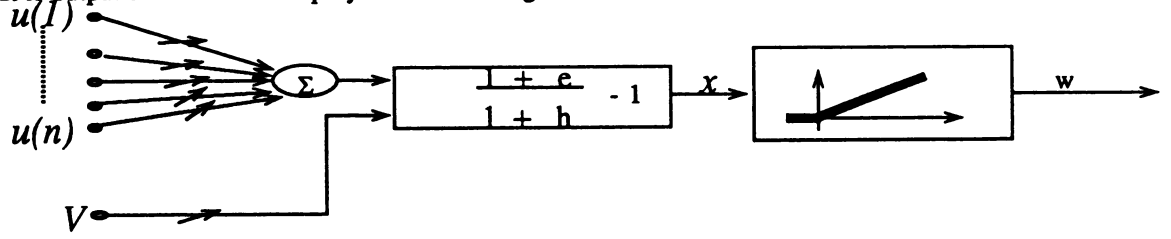


Figure 2 . Structure of a S-cell

An S-cell has an inhibitory input which causes a shunting effect. Let u(1), u(2), . . , u(N) be the excitory inputs and v be inhibitory input. Let e be the weighted sum of all excitory inputs and h the weighted sum of inhibitory input.

$$e = \Sigma a(v) * u(v) \qquad (1)$$

$$h = b * v \qquad (2)$$

where a(v) are the excitory and b is the inhibitory weights. e and h are inputs of the S-cell and w is the output of the S-cell.

$$w = \varphi\left[\frac{1 + e}{1 + h} - 1\right] = \varphi\left[\frac{e - h}{1 + h}\right] \qquad (3)$$

The nonlinear function φ is defined by the following equation.

$$\varphi[X] = \begin{bmatrix} X & (X>0) \\ 0 & (X<0) \end{bmatrix} \qquad (4)$$

The a(v) weights are variable and are reinforced during the learning process. The inhibitory input b is obtained from the output of a subsidiary inhibitory cell, which is called a V-cell. The input of the S-cell and V-cell are the same but V-cell weights are fixed. The output of the V-cell satisfies equation 5.

$$v = \sqrt{\sum c(v) * u(v)} \qquad (5)$$

The C-cells are employed in the network for positional shift and feature deformation. Connections from S-cells to C-cells are fixed and do not change during the learning process. These weights are called the d weights. The input-to-output characteristic of a C-cell is similar to that of the S-cell except the function φ is replaced by function ψ which is a saturation function defined by

$$\psi(x) = \begin{bmatrix} \frac{x}{\alpha + x} & (x>0) \\ 0 & (x<0) \end{bmatrix} \qquad (6)$$

The neocognitron is capable of training with both unsupervised learning (or learning without a teacher) and supervised learning (or learning with a teacher). In unsupervised learning Fukushima (1982), the repeated presentation of a set of training patterns is employed for the self-organization of the network and it is not necessary to give any information about the categories in which these patterns should be classified. The unsupervised learning of the neocognitron is similar to Kohonen's learning rule with the difference that S-cells of a S-plane compete for learning a feature rather than the whole pattern. Among the S-cells of a certain S-plane, only the one which is responding the strongest to a feature has its interconnection reinforced. The amount of reinforcement of each interconnection to this maximum output cell is proportional to the intensity of the response of the cell. The reinforcement is only applied to variable weights a(v) which are excitory and b(v) which is inhibitory. This reinforcement of the variable connection a(v) and b(v) is based on the following equations

$$\Delta a(v) = g * c(v) * V(v), \qquad \Delta b(v) = g * V(v) \qquad (7)$$

where g is a positive constant which determines the speed of increment.

The neocognitron may also be trained by a supervised learning rule. This learning method is advantageous for recognition of patterns which are similar in shape but belong to the two different classes. An example of this is the degree of similarity between O and 0 which belong to the same category and that of O and Q which are similar but belong to two different categories. The main difference between the supervised and the unsupervised learning is that in supervised learning each layer is trained at one iteration with a teacher. The features taught to the first layer are primitive features of the pattern like horizontal, vertical and oriented lines. The next stage is trained by more sophisticated features obtained from the combination of the features of the first layer. Each cell of the last stage corresponds to the complete global feature of a trained pattern. Supervised learning is a useful technique when the application is limited to a certain type of pattern. The extended neocognitron, proposed here, has the ability of recognizing patterns in different orintations. This new feature of the neocognitron is obtained by a generalization of the position invariant characteristic of this network. In the neocognitron, cells of a S-plane are responsible for extraction of the same feature at different locations. The cells of each row look for a specific feature along the X axis while cells of each column look for the same feature along the Y axis. This is the reason for having two dimensional S and C planes. The idea of position invariance of the neocognitron is generalized to augment the rotation invariance property. Thus the extended neocognitron is capable of recognizing patterns in different positions, orientations and scale with no preprocessing stage. It is also sufficient to train the network with patterns in one position, orientation and scale and the trained network can recall the same patterns from a different position, orientation and scale.

CUTTING PATTERN RECOGNITION USING EXTENDED NEOCOGNITRON

Stock patterns to be cut from composite piles contain features that can be classified. It is not difficult to come up with feature names as; rectangular multiple bosses, convex multiple groves, symmetric bosses. Some of the features are the terms used in the shop floor for identifying various components of the composite structure. A simple composite structure is shown in Figure 3 is differentiated from the other patterns as rectangular convex single boss (the second feature in the stage three). Sample composite structure recognition of this shape using the extended neocognitron is demonstrated in the figure based on three stage network.

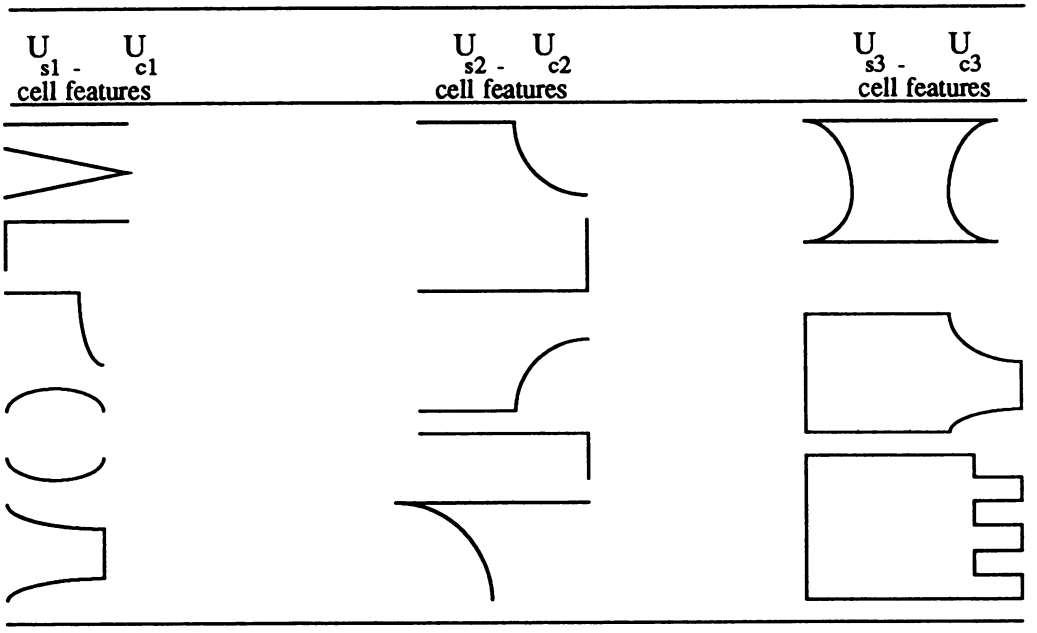


Figure 3 Sequence of feature extraction

Once the features of the patterns are identified it becomes possible to select an appropriate pattern sequence for creating a pattern nest using an expert system. This extended neocognitron architecture can then work as a pattern classifier for composite stock cutting.

CONCLUDING REMARKS

In this study a neocognitron architecture is extracted for composite stock cutting classification. Based on initial tests, the new paradigm looks promising. It can identify scrap patterns generated from the combination of various composite stock patterns. The adopted architecture of neocognitron is coded for a sequential machine and efforts are underway to implement the network on a parallel machine.

REFERENCES

C. H. Dagli, "Knowledge-based Systems for Cutting Stock Problem", European Journals of Operational Research, (In Print), 1990.

M. R. Gary and D. S. John, "Computers and Intractability: A Guide to the Theory of NP-completeness.", San Fransisco, Freeman, 1979.

B. L. Golden, "Approaches to Cutting Stock Problem", AIEE Transactions 8. 2, 1976, pp 256-274.

A. I. Hinxman, "The Trim Loss and Assortment Problems: A survey", European Journals of the Operational Research 5, 1980, pp. 8-18.

Fukushima, K. and Miyake, S. "Neocognitron: A Neural Network Model for Pattern recognition Tolerant of Deformation and shift and in position", Pattern Recognition 15(6), 1982.

Fukushima, K. "Neocognitron: A Hierachial Neural Network capable of visual Pattern Recognition", Neural Networks, vilume 1, 1988.

Switch Pattern Planning in Electric Power Distribution Systems by Hopfield-type Neural Network

Chihiro Fukui and Junzo Kawakami

Hitachi Research Laboratory, Hitachi, Ltd.
4096 Kuji-cho Hitachi, Ibaraki, 319-12 Japan
E-mail fukui%hrl.hitachi.junet@uunet.uu.net

Abstract

An application of the Hopfield-type neural network is presented in the switch pattern planning problem of electric power distribution systems. This problem requires consideration of non-equality constraints. To represent these constraints in the energy function of the neural network, a new technique was developed which introduces special neurons into the network. These neurons are expected to converge to intermediate values between 0 and 1. Simulation results show that this method is quite encouraging.

1. Introduction

Since Hopfield showed the neural network technique could solve combinational problems such as the Traveling Salesman Problem[1], many researchers have attempted to apply the neural network to various optimization problems. The constraints of almost all of those problems are combinational constraints. However, in most actual applications, non-equality equations have to be taken into consideration. This paper deals with the switch pattern planning problem which contains the the non-equality equation constraints.

2. Switch Pattern Planning Problem

The switch pattern planning problem involves determination of the on/off states of switches in electric power distribution systems. Changing their on/off states allows control of the energy flow from energy sources to consuming points such as factories and houses. A distribution system can be represented by a graph such as Fig. 1. In this figure, branches correspond to switches, and nodes correspond to consuming points and energy sources. Usually an energy source is called a "Feeder" and an energy consuming point is called a "Load". Switch pattern planning sometimes proceeds as restoration steps after an emergency situation such as a black out. If the power supply through *Feeder-0* is interrupted, *Load-1* to *Load-10* are blacked out. The restoration finds available back-up feeders and emergency supplying routes from back-up feeders to blacked out loads. Rerouting is realized by changing the on/off states of switches located in the distribution systems. Then rerouting can be considered as an allocation problem among back-up feeders and blacked out loads.

The aim of the switch pattern planning is to find the best allocation under some constraints and objective functions.

The constraints and the optimization targets shown in Fig. 2 are as follows.

(1) The total energy value from a feeder must be smaller than its capacity.
(2) Any blacked out load should not receive energy from two or more feeders.
(3) The optimizing target is to minimize the amount of black out power.

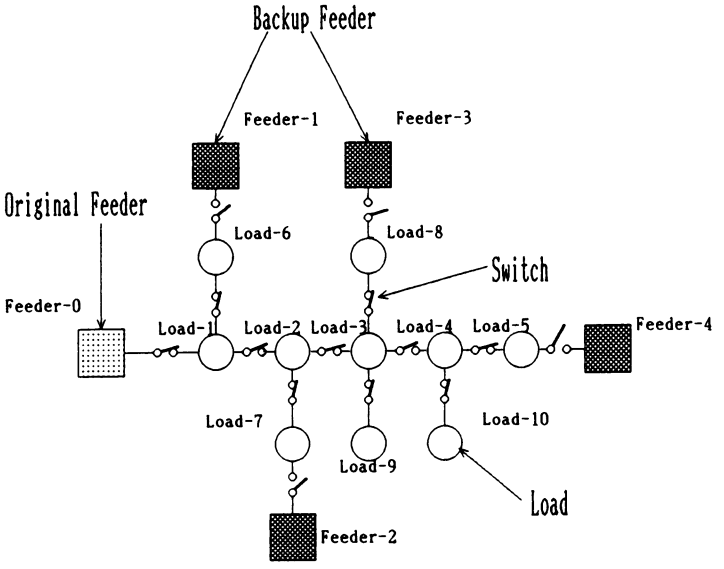


Fig. 1 Graph representation of a distribution system

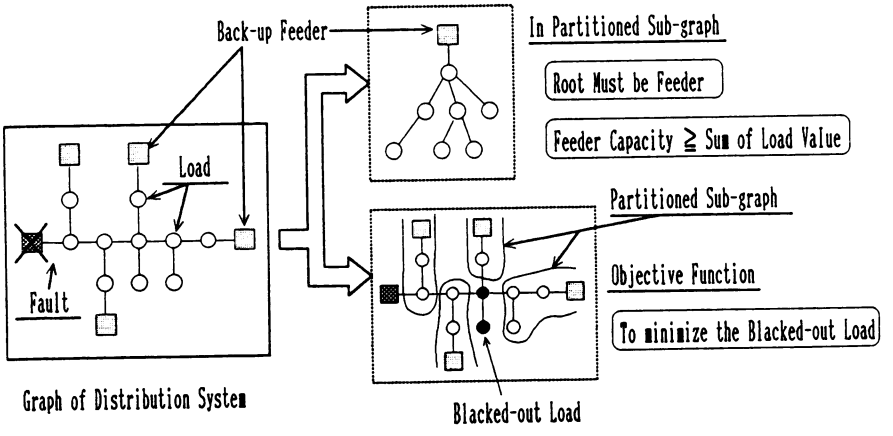


Fig. 2 Constraints of the problem

$$\begin{pmatrix} Y_1 & X_{1,1} & X_{1,2} & \cdots & X_{1,10} \\ Y_2 & X_{2,1} & X_{2,2} & \cdots & X_{2,10} \\ Y_3 & X_{3,1} & X_{3,2} & \cdots & X_{3,10} \\ Y_4 & X_{4,1} & X_{4,2} & \cdots & X_{4,10} \end{pmatrix}$$

Fig. 3 Neuron matrix

3. Formulation of the problem

With the constraints mentioned above, the switch pattern planning problem can be mapped onto the graph partitioning problem. As Fig. 2 shows, a partitioned sub-graph should be a tree which has a feeder as a root, and the sum of the required energy value in the sub-graph should be smaller than the capacity of the feeder.

The state variables *Xij* (neurons) shown in Fig. 3 are defined to represent the allocation state between feeders and loads. *Xij*=1 means that *Feeder-i* supplies power to *Load-j* and *Xij*=0 means otherwise. Constraint (2) does not allow $0<Xij<1$.

In this case, the capacity constraint is represented by the following non-equality equation.

$$\sum_{j}^{Nn} Lj \cdot Xij \leqq Fi \qquad (1)$$

where, Nn is the number of loads, Lj is the energy value which *Load-j* requires, and Fi is the capacity of the *Feeder-i*.

A new method is proposed to represent these constraints in the energy function for the Hopfield-type neural network. The fundamental concept of this method is to introduce the neurons which converge to intermediate values between 0 and 1. This special neurons Yi are multiplied to the upper limits Fi as shown in Eq. 2. The final equation for energy function $E1$ is described in Eq. 3.

$$\sum_{j}^{Nn} Lj \cdot Xij = Fi \cdot Yi \qquad (2)$$

$$E1 = \left(\sum_{j}^{Nn} Lj \cdot Xij - Fi \cdot Yi \right)^2 \qquad (3)$$

3. Energy functions

The other terms of the energy function are described here.

(1) Path constraint $E2$

From a *Feeder-i* to *Load-j*, there must be a path to transmit the energy. To represent this constraint, the following term is defined.

$$E2 = \sum_{i}^{Nf} \sum_{j \neq g(i)}^{Nn} Xij \left(1 - Xik \right)_{k=k(i,j)} \qquad (4)$$

where, Nf is the number of feeders, $g(i)$ is the load to which *Feeder-i* is directly connected, $k(i,j)$ is the load that is in the path from *Feeder-i* to *Load-j* and directly connected to *Load-j*.

(2) Tree constraint $E3$

The partitioned sub-graph should be a tree. This constraint means that the number of neurons whose values are 1 in each column of matrix $\{Xij\}$ should be 1 or 0, and defined in Eq. 5.

$$E3 = \sum_{j}^{Nn} \sum_{i}^{Nf} \sum_{k \neq i}^{Nf} Xij \cdot Xkj \qquad (5)$$

(3) Objective function $E4$

The optimization target is to minimize the number of the blacked out loads. This is defined in the objective functions $E4$ in Eq. 6.

$$E4 = \left\{ \sum_{j}^{Nn} Lj \cdot \left(1 - \sum_{i}^{Nf} Xij \right) \right\} \Big/ 2 \sum_{j}^{Nn} Lj \qquad (6)$$

The total energy function is defined in Eq. 7.

$$E = \omega 1 \cdot E1 + \omega 2 \cdot E2 + \omega 3 \cdot E3 + \omega 4 \cdot E4 \quad (7)$$

where, $\omega 1, \omega 2, \omega 3$, and $\omega 4$ are weights.

4. Simulation Results

All the neurons including Xij and Yi are calculated using the following equations.

$$dUij/dt = -\partial E/\partial Xij \quad (8)$$
$$dUi/dt = -\partial E/\partial Yi \quad (9)$$
$$Xij = 1 / \{ 1 + exp(-Uij) \} \quad (10)$$
$$Yi = 1 / \{ 1 + exp(-Ui) \} \quad (11)$$

We investigated 100 cases with various values of Fi and Lj on a model distribution system with 4 feeders and 10 loads. Figure 4 shows an example. The values of $Lj, Fi, \omega 1, \omega 2, \omega 3$, and $\omega 4$ are shown in the figure.

We got valid solutions in almost all of the cases. However, in the cases under the condition of $\Sigma Lj > \Sigma Fi$, some Xij converged to the intermediate values between 0 and 1.

5. Conclusion

A new method for describing the non-equality equation in the energy function of the neural network was investigated. The simulation results are quite encouraging although it does not always give valid solutions as the original Hopfield-type model.

6. References

[1] Hopfield, J. J.: Neurons with Graded Response Have Collective Computational Properties Like Those of Two-State Neurons, Proc. National Acad. Sci. USA, Vol. 81 pp3088-3092, 1984

[2] Abe, S.: Theories on the Hopfield Neural Networks, IJCNN-89, June 1989

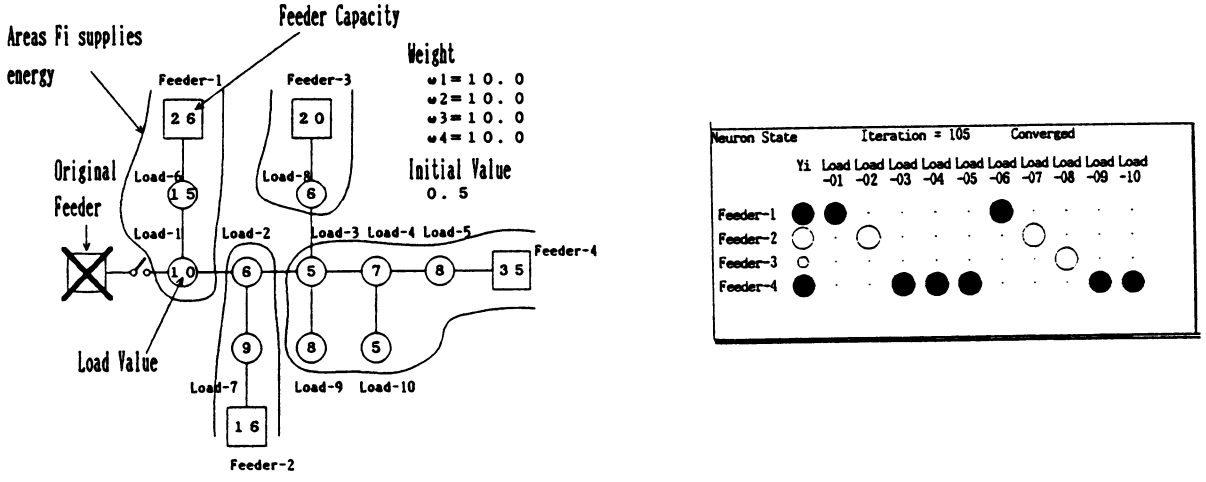


Fig. 4 Simulation results

INVARIANT TARGET RECOGNITION USING FEATURE EXTRACTION

J. Joseph Fuller, Department of Computer Science
WV Institute of Technology, Montgomery WV 25136

Ali Farsaie, Naval Surface Warfare Center
10901 New Hampshire Ave, Silver Spring, MD 20903-5000

ABSTRACT

A target recognition system using an artificial neural network was developed and trained by the Learning Vector Quantization (LVQ) method. The system distinguished among three targets regardless of their spatial orientation. It was demonstrated that recognition was invariant under the affine transformations of rotation, scaling, and translation.

Invariance is built into the network by the extraction of unique features from the various targets. This method overcame the two shortcomings of long training times and combinatorial explosion of terms often present in other networks. Two important elements of this work are the feature vectors used to represent the targets, and the recognition technique for identifying the observed targets.

INTRODUCTION

Target recognition can be defined as choosing the proper model structure to characterize a given target, or identifying the target as belonging to a certain parameterization. Some criteria used for target identification are explained by Larimore (1977), Chellapa and Kashyap (1982). Researchers have attempted the identification of unknown objects, using their 2-D silhouettes, by comparing them with models in a library. Edges composed of straight line segments have been used as features by many researchers in this context (Faugeras and Toscani 1986, Lowe 1986), and satisfactory results have been obtained.

Artificial neural networks have been trained to distinguish among digitized images of several targets presented in fixed positions with a high degree of success (Farsaie 1989). The success rate is much lower, however, when the targets are rotated, translated, or scaled in size. Previous techniques have included training the network on the images presented at different angles as well as designing networks so that the synaptic weights have affine invariance incorporated into their calculation (Giles, Griffin, and Maxwell 1987, and Reid,Spirkovska, and Ochoa 1989). This paper presents a new approach to affine invariant target recognition using artificial neural networks.

APPROACH

For the purpose of recognizing larger images of targets (100x100), an attempt was made to include the idea of information being derived from correlations between and among pixels as described by Giles et.al., but also to avoid the combinatorial explosion of terms and connections that occur as the image sizes increase. Furthermore, a neural network model was developed which accounts for aliasing and "jaggies".

Testing and training were performed on three targets shown in Figure 1. Images were 100 x 100 pixels in size. Three sets of images of these targets were generated: silhouette, thin edge (with boundary line drawn one pixel thick), and thick edge (with boundary line drawn two pixels thick). Each data set was presented to the neural network separately during training and testing.

FEATURE EXTRACTION: A procedure for selection and extraction of features was developed. This provided a set of feature vectors for use with the neural network model. Components of vector X represent the features extracted from an image, where X is defined as follows:

- X [1] = the total number of pixels in the image with value 1.
- X [2] = the sum of all of the products of pixels with the pixels which are the same distance from a designated origin but 90 degrees apart.
- X [3] = the same as x[2] except for pixels 180 degrees apart.

At first glance, the above features appeared to be invariant - at least under translation and rotation - if the proper origin is selected. However, close inspection of these features revealed that aliasing may cause any or all of the features to fail to be invariant.

To remedy the problems described above, an average number of pixels representing a target was chosen rather than an absolute number of pixels. Deviation from this assumed average is a function of the rotation angle as well as the inherent problem of aliasing. Using this approach, a learning paradigm was selected which accounted for "average behavior" in a system. It appeared that although the features described above are not "truly" rotationally invariant when a transformation is applied to a digitized target, they are "nearly" rotationally invariant and a paradigm which accounted for fluctuations about an assumed mean would work well. This led to the choice of the Improved LVQ method described by Kohonen (1987).

After several training sessions were attempted to implement rotational invariance, it became apparent that aliasing and "jaggies" caused the features extracted at the mid angles (25 degrees to 65 degrees) to differ substantially from those from the same target extracted at angles near zero degree and 90 degrees. Therefore it was decided to use two feature vectors to represent each target. The first vector was composed of features initialized with the target at standard position (at 90 degree angle), as shown in Figure 1, and the second vector used the same features initialized at mid angles.

Translation invariance was easily incorporated into the network by computing features with respect to the center of gravity of the target. Thus the "designated origin" referred to in the definitions of features X[2] and X[3] is the center of gravity of the target. Features were adjusted for scale invariance by a simple ratio and proportion scheme. The radius (R) of the smallest circle which encloses all of the pixels in the target was found. The center of the circle was at the center of gravity of the target. Each of the features was then multiplied by 144/R**2 ,where 144 is simply a base value indicating the square of the radius of the circle which encloses the training targets.

TARGET RECOGNITION: After the features have been extracted for the targets of interest, feature vectors were used for training and testing. Kohonen's LVQ2 technique was used to perform recognition. LVQ2 refers to the Improved Learning Vector Quantization technique described by Kohonen (1987). In this paradigm, codebook vectors representing known classes are initialized - either randomly or by a mentor - and training proceeds with all of the codebook vectors competing for the input vectors. If the winning codebook vector correctly classifies the input, the codebook vector is rotated toward the input vector. If the winning codebook vector incorrectly classifies the input vector, the winning codebook vector is rotated away from the input vector and the codebook vector which should have won is rotated toward the input vector.

A methodology has been formulated and algorithms were developed for training process (Figure 2). A target was randomly selected and then rotated at a random angle between zero and 180 degrees. Targets were kept fixed at one location (no translation) with their standard size (i.e., scale factor of 1:1). Features were extracted and used to generate two feature vectors as described before. Then these feature vectors were introduced to the neural network model where codebook vectors are computed and updated. During training the network assigned two codebook vectors to each target. This process was repeated for 1000 iterations, which would result in rotating the two codebook vectors toward the center of the cluster that represents the target. Iterations of 200, 400, and 600 were also used.

During testing, one of the three targets was randomly selected and rotated at a random angle. Then for any selected scale factor the target was translated by x,y. Feature vectors were extracted and then presented to the network for recognition. Training and

testing procedures were repeated for all of three sets of images (i.e., silhouette, thin edge, thick edge).

RESULTS

The recognition network was tested for all combinations of orientation angles and scale factors. Angles were varied from zero to 90 degrees, and scale factors ranged from 0.49 to 1.7. Translations ranged from -15 to 15 in both the x and y directions.

Performance of the network did not vary substantially as a function of orientation angle when thick edge images were used (Figure 3). For the case of silhouette images, the percentage of correctly recognized targets was reduced by 10 at mid angles (i.e., 30 and 60 degrees). The network performed poorly at angles near 45 degrees in the case of thin edge images.

Figure 4 shows the percent correctly classified as a function of scale factor. It appeared that in the case of thin edge images the network did well for scale factors from 0.81 to 1.2, but the performance fell off sharply as the scale factor increased or decreased beyond this range. When thick edge images were used the network performed very well for scale factors from 0.81 to 1.44 with a correct recognition rate of 97 percent. In the case of silhouette images recognition rates as high as 100 percent were achieved. When the target size was reduced the accuracy dropped by three percent.

As expected, the network trained on silhouette images exhibited the best overall performance, achieving an accuracy of 99 percent when trained for 1000 iterations (Table 1). In general thick edge images were superior to thin edge images for identifying targets (90.0 vs 70.0 percent). As Table 1 also shows, no appreciable improvement was obtained by increasing the number of iterations for either the thick edge or thin edge images.

CONCLUSIONS AND SUGGESTION FOR FURTHER STUDY

The test results clearly indicated that the feature extraction technique described here, along with LVQ2 training process represented an efficient and effective method for target recognition which is invariant under any of the affine transformations of scaling, translating, and rotating. Using only two codebook vectors for each target and only three features per vector, a high rate of success was achieved in distinguishing among three targets presented at varying spatial orientations. In addition to the rapid training process and efficient identification, a new set of features was introduced which may be easily generalized to account for more complex structured targets.

Work is underway in researching the problem of identifying targets by feature extraction with Kohonen style network. Emphasis will be placed on establishing criteria on what is the adequate training time, and incorporation of statistical measures to present probabilities of detection, as well as assigning measures of confidence to the response of the network. Other problem areas such as more efficient scale invariant algorithms, and perhaps most significantly, designing the networks to work in the presence of high signal to noise ratio are to be studied.

REFERENCES

Chellapa, R. and R.L. Kashyap "Statistical Inference in Gaussian Markov Random Field Models", Proc. Conf. PRIP, pp77-80, 1982.

Farsaie, A., "An Artificial Neural System For Target Identification Using IR Imagery Data", Internal Report, NSWC-WO, 1989.

Faugers, O.D. and G. Toscani, "The Calibration Problem for Stereo", Proc. CVPR, pp15-20, 1986.

Giles C., Griffin R, and Maxwell T., "Encoding Geometric Invariances In Higher Order Neural Networks", Proc. IEEE, 1987.

Kashyap, R.L. and R. Chellapa, "Estimation and Choice of Neighbors in Spatial Interaction models of images", IEEE Trans. Inform. Theory, vol. 1T-29, pp60-72, 1983.

Kohonen,T., "Self-Organization and Associative Memory", Springer-Verlag Co., 1987.

Larimore, W.E., "Statistical inference on stationary random fields", Proc. IEEE, vol. 65, pp961-970, 1977.

Lowe, D.G., "Three-Dimensional Object Recognition From Single 2-D Images", Comp. Sci. Division, New York Univ., TR-202, 1986.

Reid M., Spirkovska L., and Ochoa E., "Rapid Training of Higher Order Neural Networks For Invariant Pattern Recognition", Proc IJCNN, 1989.

Figure 1. Targets used for training and testing.

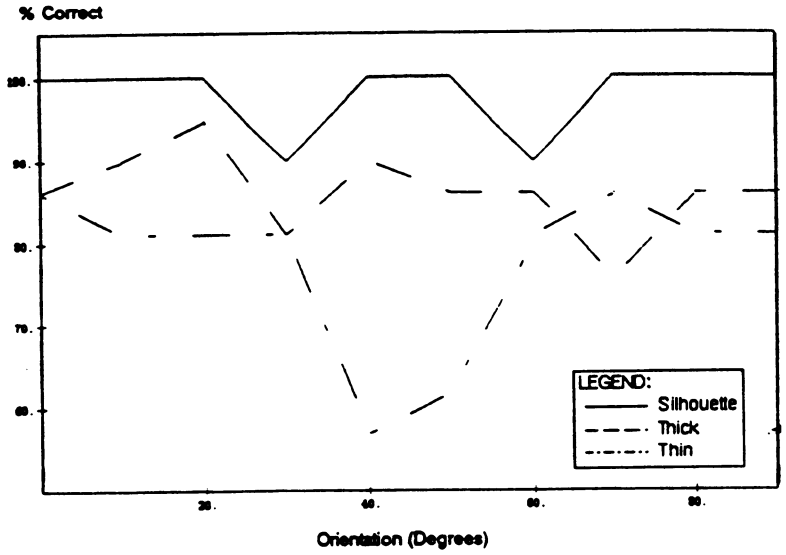


Figure 3. Performance of the network as a function of orientation angle.

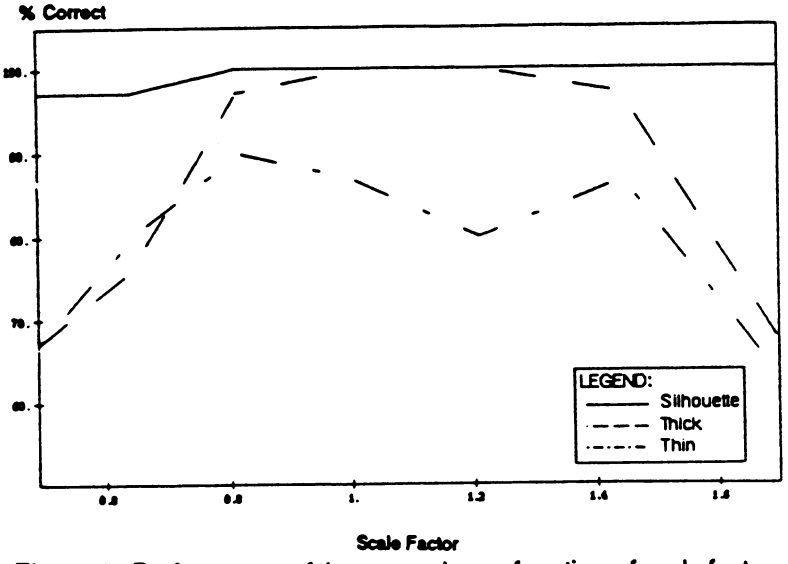


Figure 4. Performance of the network as a function of scale factor.

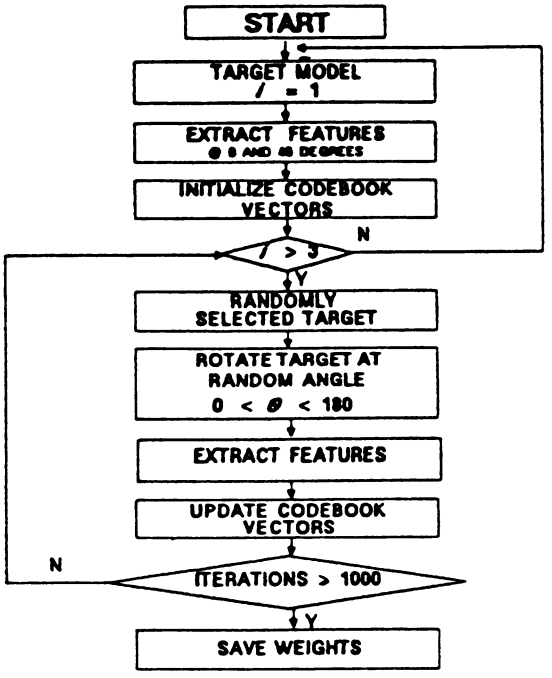


Figure 2. Methodology for training the network.

IMAGE TYPE	NO. OF ITERATIONS	% CORRECT
SILHOUETTE	200	95.2
	1000	99.0
THICK EDGE	200	83.0
	300	88.0
	400	90.0
	1000	84.0
THIN EDGE	400	63.0
	600	70.0
	1000	63.0

Table 1. Overall performance of the network using three types of images.

FUZZY LOGIC IN CONNECTIONISTS' EXPERT SYSTEMS

L. S. Hsu, H. H. Teh, S. C. Chan and K. F. Loe
National University of Singapore
Kent Ridge, Singapore 0511

KEYWORDS

Neural Network, Connectionists' Expert system, Fuzzy logic

ABSTRACT

The use of the certainty factor formalism in connectionists' expert systems has three disadvantages:

1. The certainty factors of the "AND" and "OR" operations are taken to be the maximum and minimum of the certainty factors of the premises respectively. This means that some information are completely ignored in the process.
2. The uncertainties in the facts, rules and logical operations are not treated in a unified way

In this paper, we define a new logic that weighs all available information and implemented it using an emulated neuronet.

1. INTRODUCTION

One way of handling uncertainties in an expert system is to introduce Fuzzy logic [1]. The idea of Fuzzy logic originated from the concept of Fuzzy set and inherited certain limitations. The most serious one is that not all the strengths of the conditions are taken into consideration in determining the strength needed for the corresponding action. Consider the following rule:

If A (a) and B (b) and C (c) then D (d)

where the upper case letters represent conditions and action and the corresponding lower case letters represent the strengths of the conditions and the strength of the action. Fuzzy logic would take d to be the minimum of a, b and c. If we replace the "and" in the above rule by "or", the maximum of a, b and c would be used as the value for d. There are many situations where this is not a good model of the real world.

Suppose we are building an expert system for buying of properties. The expert may say that

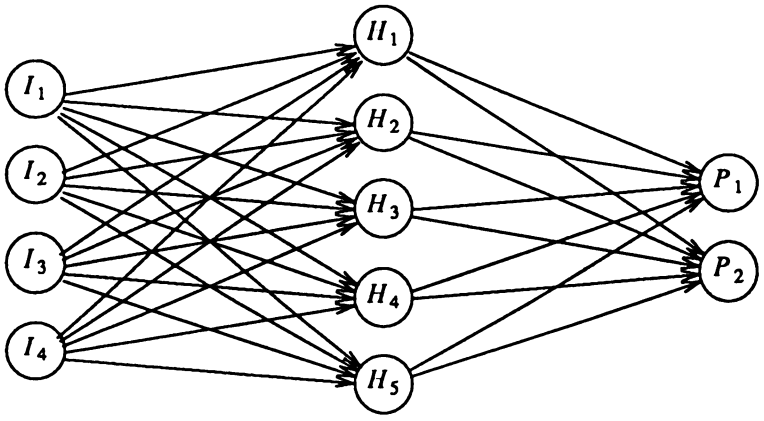
If		
	the location is good	(a)
and		
	the price is reasonable	(b)
and		
	the property tax is not too high	(c)
Then		
	buy the house	(d)

Surely the case where the values of (a, b, c) are (0.8, 0.8, 0.5) and the case where the values are (0.5, 0.5, 0.5) merit different consideration. In most decision making process, it is more realistic to take an weighted average of the strengths of the conditions rather then just the maximum or the minimum.

The following sections described a method of combining the strengths of the conditions and pointed out how it can be naturally implemented in a neural network.

2. NETWORK STRUCTURE

The neural network used to implement the decision making expert system consists of n input nodes I, m output nodes O, and p hidden nodes H. A simple case in which n=4, m=2 and p=5 is shown in the following diagram:



Each of the I nodes corresponds to a primitive condition in the system. The strength of a node is denoted by an ordered pair of real, non-negative numbers (a, b). These numbers are normalized in such a way that a+b <= 1. The numbers are assigned by the user and they have the following meaning:

1. The quantity "a" denotes the amount of evidence that is for the condition represented by the node,
2. The quantity "b" denotes the amount of evidence against the condition, and
3. The expression "1-a-b" express the lack of evidence regarding the condition.

For example, if we ask the opinion of 100 experts, 70 of them think that the condition is true, 20 of them consider it untrue, 10 of them do not know the answer, then (a, b) = (0.7, 0.2).

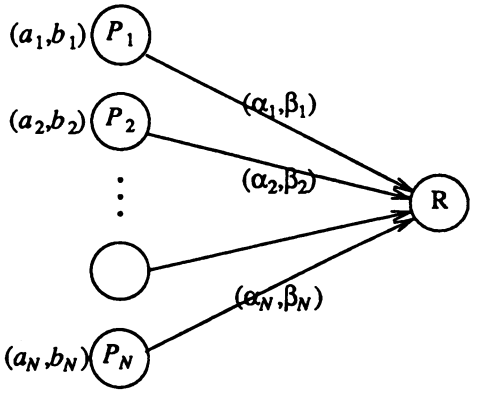
Each hidden node H represent a logical operation. It has a strength that is denoted by an ordered paired of real, non-negative numbers (c,d). These numbers are computed from methods to be described in the next section, and they satisfy the normalizing condition c+d <= 1. There is also a threshold value θ attached to each hidden node. When θ is less than 1, the system performs a threshold operation after the strength is calculated. If its value is 1, no threshold operation is needed, and the calculated strength is stored at the node right away.

Each output node O represents a possible recommended action. The strength of each node is given by an ordered pair of real, non-negative numbers (e,f). The values are computed from the weights of the internode connections as well as the strengths of the input and hidden nodes. The rule of computation make sure that the normalization condition e+f <= 1 is satisfied. The values of e and f represents the eagerness with which the system recommend the action to be or not to be taken. A large 1-e-f shows that the system does not know whether to recommend for or against the action.

The three types of nodes described above are connected by a set of arrows to form a directed graph. A weight is assigned to each connecting arrow. Each weight is an ordered pair of real numbers (α, β). Unlike the strengths, the weights are not normalized, and negative value is allowed. The actual values depend on the number of conditions used in the rule and the type of logical operations involved.

3. PROPAGATION OF STRENGTHS

To show the propagation of strengths, we shall focus our attention on one of the nodes R. Let there be n incoming arrows connecting it to n other nodes P_i whose strengths are (a_i, b_i). This portion of the network is shown in the following diagram:



The strength of the node R is calculated by the following steps:

1. Compute the total uncertainty of the incoming nodes: $uncert = \sum_{1 \le i \le n} 1 - a_i - b_i$
2. Compute the products $a_i \alpha_i$ and $b_i \beta_i$ for all i.
3. Let pos be the sum all the positive terms and neg be the absolute value of the sum of all the negative terms.
4. The strength of the node R is given by (a_r, b_r), where $a_r = pos/(pos+neg+uncert)$ and $b_r = neg/(pos+neg+uncert)$. Note that $a_r + b_r \le 1$ just as it should be.
5. Perform threshold calculation. If $a_r - b_r \ge \theta_r$ then we set $a_r = 1$ and $b_r = 0$. For nodes that does not reach the threshold, the values of a_r and b_r are not changed.

4. DETERMINATION OF WEIGHTS

The computation described in the last paragraph made use of strengths of incoming nodes as well as weights of the connecting lines. The weights depends on two factors: the number of incoming nodes and the type of logical operation. We shall describe the determination of weights for a few logical operations.

The criteria used is very simple. From the definition of the logical operation, we know whether R is true or false for a given set of inputs. We choose a set of weights so that R agrees with this result. For this purpose, we note that R is true if $a_r - b_r >= 1$. It is false if $a_r - b_r <= -1$.

4.1 The 'AND' operation

Let

$$R = P_1 \text{ "AND" } P_2 \text{ "AND" } \ldots\ldots \text{ "AND" } P_n$$

then we set

$$\alpha_i = 1/n$$

and

β_i = n The following table from Kleene's strong three-valued logic [2] gives a definition of the binary "AND" operation.

"AND"	true	false	unknown
true	true	false	unknown
false	false	false	false
unknown	unknown	false	unknown

It can be easily verified that the above assignment of weights does produce the desired result when we note that

1. For the input nodes, the strengths are (1,0) for true, (0,1) for false and (0,0) for unknown.
2. Let (a, b) be the strength of node R. R is true if a-b>= 1. R is false if a-b <= -1. R is unknown otherwise.

Inductive argument shows that the assignment is valid for the case with n incoming nodes.

To model different worlds, other types of three-valued logic may be defined. This results in a different table of definition and consequently a different set of weights for the lines connecting nodes to an "AND" node. We have developed a program which produces the weights by reading in a table of definition.

4.2 The 'OR' operation

Similarly, it can be verified that for the "OR" operation, whose definition is given in the following table:

"OR"	true	false	unknown
true	true	true	true
false	true	false	unknown
unknown	true	unknown	unknown

the weight assignment is:

$$\alpha_i = n$$

and

$$\beta_i = 1/n$$

4.3 The negation operation

The negation operation is a unary operation. The table of definition is very simple:

P	"NOT" P
true	false
false	true
unknown	unknown

and the weight is:

$$\alpha_i = -1$$

and

$$\beta_i = -1$$

5. Conclusion

The neural network suggested above is useful in developing expert systems that helps in decision making. It allows for fuzziness in the facts as well as rules in a natural way. It is more realistic then the classical fuzzy logic because for both "and" and "or" operations, all evidences are weighed and taken into account.

The expert system can operate in two modes. In the normal mode, rules are given by experts and weights are assigned values given in the last section. In the learning mode, weights are allowed to vary while the system is fed with examples.

REFERENCES

1. Zadeh L. A., "The Role of Fuzzy Logic in the Management of Uncertainty in Expert Systems," *Fussy Sets and Systems*, vol. 11, pp. 199-227, 1983.
2. Turner R., "Logics for Artificial Intelligence," , Ellis Horwood, 1984.

ARCHITECTURAL ISOMORPHISMS IN NEURAL NETWORK APPLICATIONS

JIM HUFFMAN
JOHN SCOGGINS

MOTOROLA INC.
MICROPROCESSOR PRODUCTS GROUP
6501 WEST WILLIAM CANNON DRIVE
AUSTIN, TEXAS 78735

ABSTRACT: It is the purpose of our research to determine whether or not linking serial and neural architectures functionally would map to a model of a corpus collosum*which would provide insight to optimal matching in a hardware hybrid system. Given the current static neural network models, it is difficult to determine isomorphisms between computer based functions and their biological counterparts. As we progress more toward dynamic networks, we anticipate using systemic approaches that more closely map to the physiology of the network rather than make the division of processes at a procedural level. We have created a road map for a particular application which commences with a procedural level division of processing, using conventional static neural network technology, and will evolve to a powerful workstation based on systemic models.*

I. INTRODUCTION

Our goal in this experiment has been to evaluate certain aspects of optimization in the interfaces between conventional computing architectures and architectures based on Neural Networks which are, functionally more biologically authentic . We wished to create a practical, operational program to help locate functional areas for optimal interconnection. We also wished to provide research on the practical use of these combined systems in areas where artificial neural networks could supplant or duplicate human functions.

We have sought our modeling examples from the related fields of neurophysiology and computer science with a bent to the practical application in a working computer program. This meant defining a systemic[1] architecture that would provide stepped functionality culminating in a future system based on alternate neural networks which more closely parallel the biological in nature and function. The first step, is a simple application available to any interested users at no cost. The application is based on the Lüscher Color Psychology Tests.

From the field of biology, the macro structure that provides inspiration for our model is the left and right cerebral cortex with specialization to "left hemisphere" rules and algorithms cooperating with "right hemisphere" heuristics and "fuzzy" matches. The outcome of our research would be insight into the functional equivalent of the *corpus collosum*, providing an optimized bridging of left-right functionality. We expected to be able to find areas for the optimal distribution of system functionality from examples in neuropsychology.

From computer science, we find combined artificial intelligence (AI) and neural network (NN) techniques[2] that use rule based AI analysis where expedient and otherwise allow the NN to provide the subjective/objective information handling. A synthesis of these examples has led us to investigate expert system technology which uses NN to provide the inference mechanism.

For an initial application, psychological/behavioral modeling seemed an interesting direction to pursue for several reasons. On one hand, starting with an interactive counseling advisor would be feasible. Self-evaluation conditions and the need to provide subjectivity for self-analysis fit well within our proposed technology. Furthermore, this kind of computer application could evolve as a future tool for psy-

1. *See* [Olson 1989].
2. *See* [Rumelhart 1986].

choanalysis, taking advantage, in its simplest form, of its: data collection capabilities, comparison of analysis against objective system, and the flexibility to alter the application based on personal experience of the professional.

We decided to base our application on the Lüscher Color Test[3], expanded to include a more modern paradigm for interpretation. Why choose Lüscher? Consider that the Color Test presents a well documented and popularly accepted self-analysis methodology that is constrained to a reasonable number of possible diagnosis/prognosis. Also it maps well to our duality requirement, i.e. using a rule based approach based on Lüscher's specific criterion and examples, cooperatively with "fuzzy" analysis providing a graded prognosis using NN.

II. Implementation

A computer program for implementing the Lüscher Color Test can be neatly structured into three phases: *Test*, *Analysis*, and *Prognosis*. The first phase, *Test*, essentially concerns the human interface for administering the Color Test. The metaphor of selecting from a small number of cards in a specific order, suggests a simple user interface based on direct manipulation. For instance: randomly arrange all the cards on the screen and instruct the user to "pick" cards using a mouse, starting with the most favorite color on through to the least favorite color.

The second phase, *Analysis*, addresses the data structures and algorithms used for scoring test results. Since the Color Test consists of two iterations of an eight color test, the data structures are simply a set of 2 x 8 arrays representing: selected order, color pairings, analytical marks, anxiety/compensation, and stress. This data is determined after the second test according to the Lüscher rules.[4]

The final phase, *Prognosis*, uses the derived data to condition the NN input, then activates the NN to infer possible behavioral patterns from a set of learned archetypical test scores. The inferred results could be either presented as the most likely match, i.e. found by using some "clustering" function, or they could be a set of partial matches along with associated "certainty" values.

1. Criteria for the Application

The architecture of the application must allow for the use of alternative NN types as the research progresses. It was determined that the first generation of the application would have a standard three-layer perceptron with back propagation learning.[5] This was a "public domain" architecture which would allow us to distribute the program freely.

The application required an intuitive human interface targeted to technically unsophisticated users. We have targeted for individual desktop computer users in two categories: *users* interested in the application as it pertains to practical NN technology or interested in the self-analysis and self-improvement aspect, and *professional researchers* engaged in personality analysis. The design principles of flexibility for researchers and ease of use for target users would drive architectural and implementation considerations.

Flexibility in working with the data required the ability to modify previously collected diagnostic information, and the text of the prognosis as well. These features would be important for researchers although perhaps of little benefit to individual users. So a "user mode" was defined to distinguish between operating the program and modifying its means of diagnosis.

Data collection/entry capabilities were required in two methods: *ease of use* by providing mouse support for "point and click" operations, and *efficiency* by using an editor to allow clerical input of data. This also provided flexibility to alter the application as much as possible, depending on the personal experience of the professional user. It was intended that statistics could be gathered at a central point or group statistics could be distributed to desktop systems for incorporation into their individual databases,

3. *See* [Scott 1969] *which is an English translation of the original work in* [Lüscher 1948].
4. *The Lüscher rules are described in the appendix.*
5. *See* [King 1989] *for an example of this NN type, including C source listings.*

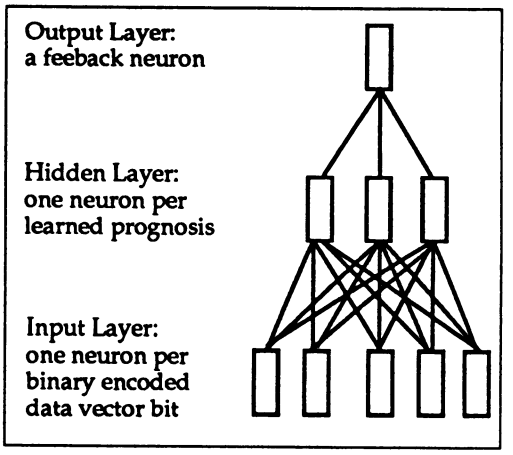
in order to allow cross correlation of Lüscher test outcomes to other tests or to cross correlate major environmental factors. As an example, we would use this facility to update the Lüscher prognosis to include more modern philosophical approaches.

Finally, the specific equipment needed would be a Macintosh[6] with a color display. Additionally a black and white option would be provided to allow demonstration of the application on monochrome computers.

2. Features of the finished *Lüscher* application

The Macintosh software development has resulted in a reasonably small application, under 75K bytes in size. The Macintosh operating systems allows the prognosis and associated data patterns to be stored and manipulated as *data resources*. This provides a convenient database mechanism for both the set of standard behavioral patterns as well as for collected data.

When the program starts, it searches for available prognosis resources and then dynamically allocates a NN large enough for the known set of patterns. This means that the program is essentially bounded by the memory requirement of one neuron in the hidden layer per learned prognosis (see Figure 1). The program then learns each data pattern as a data input vector associated with a resource index for the corresponding output value.



The human interface for the typical user consists of a window with eight color tiles, which the user clicks in succession using the mouse. After two passes, the program runs the rule based analysis and then infers a match from the known prognosis, which is presented to the user on another window. Using our current set of 23 prognoses, this entire transaction requires an amount of processing time on a Macintosh IIx that is negligible to the user. Overall, the user interface is reminiscent of many card based video games.

A major tradeoff of the entire approach is that a true biological model is not available. Thus the interface work done on this application was only good from a functional standpoint, since we were not able to create a more architecturally realistic *corpus collosum* model.

III. Futures: Toward a true Behaviorist's Workstation

It is our intention that by using a neural network that more closely models the biological we can provide the kind of functionality that could create a Behaviorist's Workstation, or Psychological Workstation. Current research leads us to believe that this is possible. We also believe there is at least another step that is possible using more conventional, biologically inspired networks.

One option would be to have the computer act as the analyst and/or the subject. This operating mode bears semblance to the proverbial *Turing Test* for computer intelligence.[7] Building on this analogy, we are pursuing the implementation of a natural language interface where the computer uses prognosis from the user's test scores as a basis for dialogue and further questioning, reminiscent of the *Eliza* program.[8]

From our current perspectives in using this application, we can see yet anther possibility. That is of actually allowing the NN to synthesize the behavioral input to the program. For instance, from the knowledge base of diagnostic data provided, the NN could make the Color Test selections. Thus the

6. *Macintosh® is a registered trademark of Apple Computer, Inc.*
7. *As described in* [Hofstadter 1980].
8. *As described in* [Weizenbaum 1965].

workstation could be used to teach the evaluation skills in pinpointing particular neuroses.

Of course, a neural network closely modeling the biological could be artificially given certain neuroses or psychoses and when used in such a workstation could help the behaviorist or psychologist devise still more refined tests, possible cures, and evaluate the outcome of the application of these hypotheses. All this is quite impossible to do on humans at this time, but it would make the Behaviorist's Workstation a powerful instrument for the benefit of anyone involved in human behavioral studies such as the aforementioned behaviorists or psychologists, or even marketers and politicians. In those cases aberrant behavior might be considered as stress conditioning caused by environmental influences such as impact on local communities of individuals.

IV. Conclusions

Up to the point of this writing, we have investigated various possibilities for the architecture of this program in light of an overriding set of requirements that it potentially evolve into a Behaviorist's Workstation. We have come up with a plan and timetable for our activities, and we have developed a computer program which we are offering to all interested parties.

We are pleased with the application as it applies to the concept of a workstation for behaviorists or psychologists and we are pleased to offer this first application to those generally interested in practical application of NN or people who are interested in self-help systems.

In a hardware system, we would have been more cognizant of specific data paths and how they pertained to overall performance as well as finding optimal placement of data handling interfaces as in this simple application. From a standpoint of function splitting and scheduling parallelism, this simple example did not provide the breadth of a platform we would need.

Lüscher claims that his colors were carefully selected with just the right tinting within each to elicit the appropriate responses. We carefully matched his colors using an 8-bit color table on the Macintosh, only to find unit to unit variations that distorted our original palettes. We were forced to compromise Lüscher's colors in favor of more distinct hues, however we intend to provide a color name facility to allow individual users adjustment the application's palette to suite their particular desktop system.

Note that the simple perceptron model requires binary input to its neurons, so the derived Color Test data in our application must be binary encoded. At one point we accidentally left a zero input vector in the database, i.e. all zero bits, and found that the otherwise functional NN produced extreme outputs regardless of the NN input. This led us to revise our binary encoding with attention paid to Hamming distance analysis[9], which led to more consistent output. The question of data encoding appears to be a problem for almost any digital neuron model, so we consider research into analog models of neural networks a potentially more profitable pursuit.

We are convinced that neural networks can find great applications in the Behavioral/Psychological Workstation areas, working in concert with conventional architectures. We are looking forward to our future research work in this direction.

9. *See* [Hamming 1980].

A Neural Lexicon in a Hopfield-style Network

Arun Jagota *(jagota@cs.buffalo.edu) , Yat-Sang Hung (hungys@cs.buffalo.edu)
Dept Of Computer Science, Bell 226
State University Of New York Buffalo, NY 14260

Abstract

A Hopfield-style network model was first developed in [2] and studied in detail in [3]. This model has the advantage of higher storage capacity and less interference between stored memories than the classical discrete Hopfield network [1]. In this paper, we consider one application of this model, namely, a Neural lexicon. We also describe a new *learning* rule that further improves the capacity of the model.

Introduction

A Lexicon can be formulated as a content-addressible memory with words in the lexicon being memories to be stored and the primary task being to retrieve the *best* stored word given an input (mis-spelt, incomplete or noisy word). Such a lexicon can be used as a component in cursive script (or other hand-written/machine written text) recognition systems because it can tolerate noisy or incomplete input. It can also be used for spelling correction of electronic text.

A detailed description of our Hopfield-style model and its computational properties can be found in [3]. In this paper, we will focus only on those aspects relevant to the lexicon application. Compared to the classical discrete Hopfield network [1] the learning rule described in [2, 3] considerably mitigates the problem of interference, resulting in high network capacity. However, experiments (described in this paper), show that spurious minima become a serious factor when the number of stored memories is between n and n^2, n being the number of units. This number of memories would be required for the Lexicon application. A new learning rule is developed to attempt to unlearn *spurious* minima in such a case. Our experiments show that the new rule performs moderately better (and much faster) in unlearning spurious minima words.

Our activation rule is also different than that of the classical discrete Hopfield network in that it performs *steepest descent* in the energy landscape [2, 3].

Model for Spatial Memories

Let the environment be modelled as a domain D of symbols. A unit is associated with each symbol. The network is fully connected. The number of symbols in D is not fixed. New symbols can be added any time, which means the network can *grow* new units and new connections. Here, however, we assume a fixed size domain D on n symbols, s_1, s_2, ..., s_n. Let k be the maximum size of any stored set, that is $k = max\ (|S_j|)$ for all stored sets S_j. Then k more symbols are made available by this model, named 1 through k. These symbols *represent* sizes of stored sets and are essential for storing subsets of other stored memories [3]. A unit is associated with each *size* symbol and such units are called *size* units. p_i denotes the i_{th} unit. w_{ij} denotes the *symmetric* constraint between units p_i and p_j. In a stable state (local energy minimum), each unit, p_i, has the value 0 or 1. Initially, units may have values in the interval $[0,1]$. For the time being, however, let initial p_i also be $\{0,1\}$. A spatial memory is represented by the set

$$S_m = \{s_{i_1}, s_{i_2}, ..., s_{i_j}\},\quad S_m \subset D$$

The network implicitly includes the symbol representing $|S_m|$, in this case, j, during training. Each spatial memory is represented by a particular set, S_i. Symbols in that set may either be clamped or unclamped. s_i^{*1} denotes a symbol clamped to 1. s_i^{*0} denotes a symbol clamped to 0.

The network has the following initial state.

$$w_{ij}(0) \;=\; \rho\ ,\ \ \rho < -(n+k)\ \ \forall i,j\ \ i \neq j \qquad (1)$$

Learning and Energy Descent Rules

The precise formulation of this learning rule (LR-I) can be found in [2, 3]. Very briefly, pair-wise constraints are learnt between all symbols in the test input S_j and the symbol representing $|S_j|$. Let $p_i, p_j \in S_j$ and both be clamped. $w_{ij}(t+1) = \lambda$ *if* $w_{ij}(t) = \rho$. $w_{ij} \uparrow$ if p_i and p_j are clamped to 1. $w_{ij} \downarrow$ if one of them is clamped to 0. The non-linear learning rate (λ) bounds w_{ij} between $[-1, 1]$ unless it has the value ρ. This learning rule allows the development of local minima separated from each other via strong lateral inhibitions (ρ) thus increasing capacity and minimizing interference.

As an alternative, the following non-incremental Learning Rule (LR-II) was formulated specifically for unlearning spurious memories without forgetting real ones. It was expected to perform better than LR-I for the number of stored memories, $M, >> n$. Let m_s be the maximum size of any set (memory) to be stored. The network has the same initial state as in 1) except that $\rho << -(n+k)$. Pair-wise constraints are learnt between all symbols in the test input S_j and the symbol representing $|S_j|$ according to the following rule..

$$w_{ij}(t+1) = -w_{ij}(t)/m_s \quad \textit{if } (p_i^{*1} \textit{ and } p_j^{*1} \textit{ and } w_{ij}(t) < 0) \textit{ or } (p_i \textit{ and } p_j \textit{ are clamped and } p_i \oplus p_j \textit{ and } w_{ij}(t) > 0) \tag{2}$$

w_{ij} is unchanged otherwise.

Our energy descent rule switches the unit that reduces the energy the most, thus performing *steepest descent* in energy space [2]. The local energy each unit can lower by switching is calculated as follows

$$\Delta E_i = -V_i * \textstyle\sum_j w_{ij} * p_j + r$$

where V_i is the direction of switch (1 or -1) unless no switch is possible in which case V_i is 0. r is a global resistance parameter. For the time being, we'll assume r is 0. *Clamped* units cannot switch. Let $p_{i_{min}}$ be the unit with minimum ΔE. If $\Delta E_{i_{min}} < 0$ then $p_{i_{min}}$ is switched in the direction of $V_{i_{min}}$. The advantages of steepest descent are discussed in [2].

Lexicon application mapped onto this model

We wish to store words. The symbols that make up words are letters at particular positions. Hence our domain D has symbols $s_1, ..., s_n$ where $s_i = (l_i, p_i)$, l_i is the letter at position p_i. For notational convenience, (l_i, p_i) is replaced by $l_i.p_i$, the "." denoting concatenation. Particular words are then stored as sets on D.

Example: Store three words: *cat, car , dim.* Then $D = \{c1, a2, t3, d1, i2, m3\}$. The inputs to the model during training are the sets: { **c1 a2 r3** }, { **c1 a2 t3** }, { **d1 i2 m3** }. For notational convenience, during training, all symbols are assumed clamped to 1 unless followed by the suffix "*0" indicating that they are clamped to 0. The network that results after training is shown in Fig 1.1. A test input can be any set on D, for eg, { **c1 a2 r3 t3 m3** }. Observe that it represents three hypotheses at position 3. The network trained with only three words would then settle into *one* of { **c1 a2 r3** } { **c1 a2 t3** }, representing *cat* and *car* respectively.

This model has a fair number of advantages. It's a good content-addressible memory. It is easy to show that it can fill in missing information and clean up noise. Also, if we employ LR-I and make the assumption that the only negative weights are of size ρ, it has the following property. Let s_i be a test input set. Then if $\exists s_j, s_i \subseteq s_j$ and s_j is a stored memory, no unit in s_i can switch off. If s_j is the only stored superset of s_i, then the network will terminate at s_j, regardless of $|s_j|$. It is possible that $\exists s_k, s_i$ is not a subset of s_k but $|s_k| - |s_i| < |s_j| - |s_k|$. A Hamming distance minimizer would reach s_k, our network will reach s_j. An example illustrating this effect is as follows. Consider two stored words (among others), *plug* and *plagiarize.* Consider the following input: { **p1 l2 g4 z9** }. A Hamming distance minimizer would retrieve *plug.* Our network will retrieve *plagiarize.* A proof of this effect can be found in [3].

An architectural advantage of this model is that it is single-layer. To store M words, we need n units and $O(n^2)$ connections, $n << M$ for large M (see Table 1.1). Retrieval time in our network is guaranteed to be within n network steps [3]. In fact, it is typically within w network steps, w being the maximum word length. The network architecture (single layer), learning rule (I) and energy-descent rule seem *feasible* to implement in analog vlsi. Implementing the wide range of values for weights could be a problem. Steepest descent could be approximated by decaying an initially high global threshold signal sent to each unit.

One other advantage is that the learning rule automatically guarantees that units representing different letters at the same position are mutually inhibitory (since the network will only be trained with sets representing words).

The main problem with this model is that *spurious memories* can develop. Consider using the model to store the following words. *cat,con,rot,car,for* From Fig 1.2, it is clear that two spurious words get stored, *cot* and *cor.*

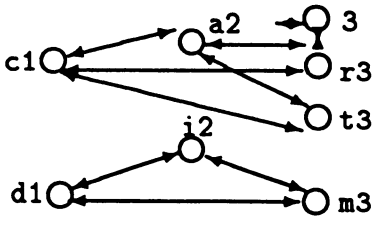


Fig 1.1

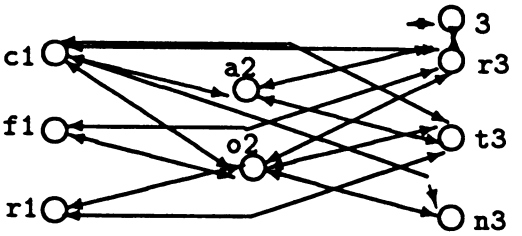


Fig 1.2

It is hard to imagine how the development of *spurious* memories could be avoided within a single-layer network architecture. In order to stick with a single-layer architecture, such memories must be *unlearnt*. Very roughly, the idea is to train the network, detect spurious memories (not words in the training set) and decrease the strengths of the connections *responsible* for reaching the spurious memory from a test input. One approach is to decrease their strengths *incrementally* in order to keep them low $+ve$ numbers (to avoid forgetting any real memories) and assume that *steepest descent* will usually avoid these spurious memories. Another approach is to make these connections $-ve$ in one step removing a *spurious* memory in one shot, but it has the danger of suppressing *real* stored memories. The problem with incremental decrements, on the other hand, is that *spurious memories* are never fully unlearnt, only suppressed (hopefully) usually. LR-II is an attempt to remove *spurious memories* without also removing (forgetting) some *real* memories. The principle underlying it can be stated as follows. A configuration of weights (parameters) is desired such that

1. If s_j is a real memory then for every $p_i \in s_j$, $\sum_{p_k \in s_j - \{p_i\}} w_{ik} > 0$. Moreover, for every $p_k / \in s_j$, $\sum_{p_i \in s_j} w_{ik} < 0$.
2. If s_j was a spurious memory then, $\exists p_i \in s_j$, $\sum_{p_k \in s_j - \{p_i\}} w_{ik} < 0$ OR $\exists p_i \in D - s_j$, $\sum_{p_k \in s_j} w_{ik} > 0$.

Strong lateral inhibition (ρ) ensures the second condition of item 1. LR-II comes close to satisfying the first condition in item 1 and the condition in item 2 by appropriately choosing m_s, the weight division factor.

Table 1.1 shows how the number of units and the number of spurious memories (before unlearning and reached after first letter in each word is removed) scale with the number of stored words. It suggests that network complexity scales well for this application since complexity is a function of n ($O(n^2)$ and $O(n)$ for space and time respectively), not the number of words. The number of spurious memories increases more than linearly with the number of stored words making the training process difficult.

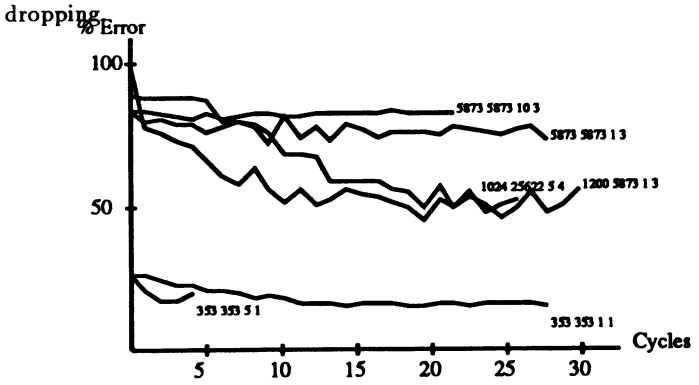
Words	Units	Spurious
150	110	30
353	248	84
968	280	157
5873	489	4774
23500	525	21601
25622	533	19802

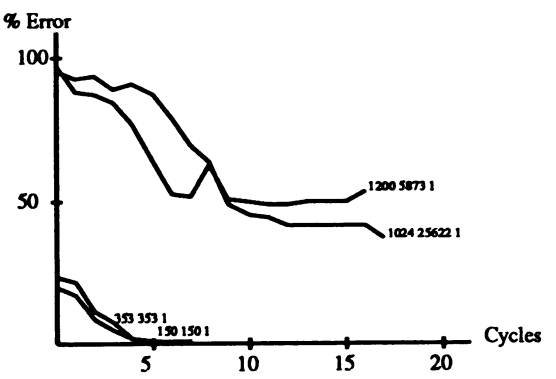
Table 1.1

Experiments

The experiments using LR-I were based on the following algorithm. In practice, it is sufficient to choose $\rho < -w$, where w is the maximum word length. $\rho = -62$, #size units between 10 and 15. $\lambda = 0.1$. The network was trained on all words 10 times to make the minima deep. The network was tested on the trained words for errors (the only errors will be subsets of others (car, carpenter) [3]). The network was re-trained with the erroneous words. This process was repeated until there were no errors. In all experiments, this process converged within 3 steps showing that all words were reachable. The Unlearning Phase was then begun. The following sequence of steps was repeated for P = 1 to 6. All the words were tested with the P_{th} letter removed and the spurious memories noted. The spurious words were unlearned and the correct words were reinforced as follows. Let cor be the spurious word (P = 3). The network and expected results are.. { c1 o2 } -> { c1 o2 r3 } (Actual) { c1 o2 } -> { c1 o2 t3 } (Expected). r3 should be unlearnt and t3 reinforced as follows:- { c1 r3*0 } { o2 r3*0 } { c1 t3 } { o2 t3 }.

The experiments using LR-II were based on the following algorithm. ρ was between -1562 and -3125. m_s was between 10 and 15. There was no reachability phase, word size information (size unit) was included with each test word to make every word reachable without re-training. The network was trained on all the words. The Unlearning Phase was then begun. The following sequence of steps was repeated for P = 1 to 6. All the words were tested with the P_{th} letter removed and the spurious memories were noted. The spurious words were then unlearned as before (same unlearning input as shown in the LR-I example works with LR-II). Unlike for LR-I, the correct words were not reinforced. Unlearning was expected to cause forgetting of real words. Hence the network was then tested with all the trained words and a list of forgotten words was compiled. The network was re-trained with all the forgotten words by presenting each forgotten word again. The above steps were repeated (for same P) until the errors started





Words	Before					After				
	1 Vwl	1 Cns	All Vwls	All Cnss	Multi-Hyp	1 Vwl	1 Cns	All Vwls	All Cnss	Multi-Hyp
150	19	19	23	25	18	7	6	8	27	8
353	55	80	99	175	106	40	33	78	171	95
5873*	908	1104	1157	1097	1108	792	552	1156	1092	-
150	33	26	39	42	25	0	0	6	18	4
353	72	101	-	-	-	46	25	85	126	58
25622**	-	-	-	-	-	482	410	851	746	772

Table 1.3 & 1.4 (* tested with 1200 words, ** 1024 words)

Analysis and Conclusions

The results are only an indication of the errors in retrieveing one of the stored words. It was shown earlier (Section on Advantages and [3]) that if the network retrieved a stored word, it would reach the *appropriate* stored word. This was also verified experimently. The graphs above the tables plot the error % in the Unlearning Phase. The left graph is for LR-I and the right one for LR-II. Each curve shows the error % after repeated unlearning for a fixed value of P (2). In the tables, Table 1.3 (first 3 rows) are for LR-I, Table 1.4 (the next 3) for LR-II. The columns (except Multi-Hyp) indicate items removed. Thus **1 Vwl** means that the first vowel was removed from each word for testing, eg *cat* – > { c1 t3 3 }. **Cns** stands for Consonants. The column labelled **Multi-Hyp** means multiple hypotheses were generated for letters at particular positions (based on lower case visual similarity), eg *that* – > { t1 l1 h2 a3 e3 c3 o3 t4 l4 4 }. The network's ability to reach a single correct word was tested. The tabulated results are based on including word size information in the testing set (see above examples). Tests without such information showed slightly inferior results.

For a small training set (150 to 350 words), both the Learning Rules performed well with LR-II performing slightly better (95 – 100% correct). For a large training set (25622 words), the number of additional spurious memories was extremely high and LR-I seemed unable to unlearn them. But by focusing on only 1024 words of the 25622 trained, LR-I was able to unlearn from 914 spurious memories to about 540. On the same set (25622, 1024), LR-II was able to unlearn from 940 down to 387.

In conclusion, we have described a Hopfield-style network model with large capacity ($> n$) and demonstrated this capacity on a real-world application. We have also shown that this application maps well on to the model. On the negative side, we have shown that spurious memories interfere with the network performance as the number of stored memories is $>> n$.

References

[1] Hopfield, J.J. (1982). Neural networks and physical systems with emergent computational properties. *Proceedings of the National Academy of Sciences, USA, 79, 2554-2558.*

[2] Arun Jagota and Oleg Jakubowicz (1989). Knowledge Representation in a Multi-layered Hopfield network. *Proceedings of the International Joint Conference on Neural Networks, 1989.*

[3] Arun Jagota (1989). Symbolic Memories in a Hopfield-style network. *Submitted for consideration to NIPS-89*

A Neural Network Model for Fault-diagnosis of Digital Circuits

Oleg Jakubowicz and Sridhar Ramanujam
Department of Electrical and Computer Engineering

State University of New York at Buffalo
Buffalo, New York 14260

August 1, 1989

Abstract

This paper describes a neural-network based diagnostic system that directs or assists a technician in diagnosing faults in a piece of electronic equipment. First, a neural-network model is described that finds a fault when there is a one-to-one association between the faults and the symptom state they produce. The overall diagnostic system concept is described. This model has been extended to take care of the condition wherein each symptom state could have been produced by one of several faults.

1 Introduction

The recent advancement in integrated circuit technology has created a demand for precise testing and diagnostic tools. The testing of digital systems requires rapid detection, isolation and rectification of faults and malfunctions as they occur. This is done by analyzing symptoms that are recorded through the application of special test sequences. The conventional approach is to employ an expert technician to test and retest the system from the point of occurrence of fault to the module responsible for the fault, by backtracking, thereby isolating the fault. These conventional methods are known to have exponential worst case performance and are not adequate in the rapidly changing world of VLSI test systems [3, 4]. More sophisticated and powerful tools can be built using the state-of-the-art concepts from computer science, artificial intelligence and neural networks.

This paper proposes a neural-network based diagnostic system for digital circuits that directs or assists a technician in diagnosing faults in a piece of electronic equipment. Its goal is to develop a general diagnostic system that captures the knowledge of test engineers and technicians to perform system fault diagnosis. Section 2 gives a detailed description of the system architecture and section 3 contains an example of applying our method to a 4-bit binary full-adder. Section 4 is the conclusion and briefly discusses the advantages of this approach.

2 System Architecture

The diagnostic system is a three-layer network consisting of an input layer, a hidden layer and an output layer(Figure 1). The number of neurons within each layer can be set at run time depending upon the circuit under test at that instant. All neurons within each layer are fully connected with all

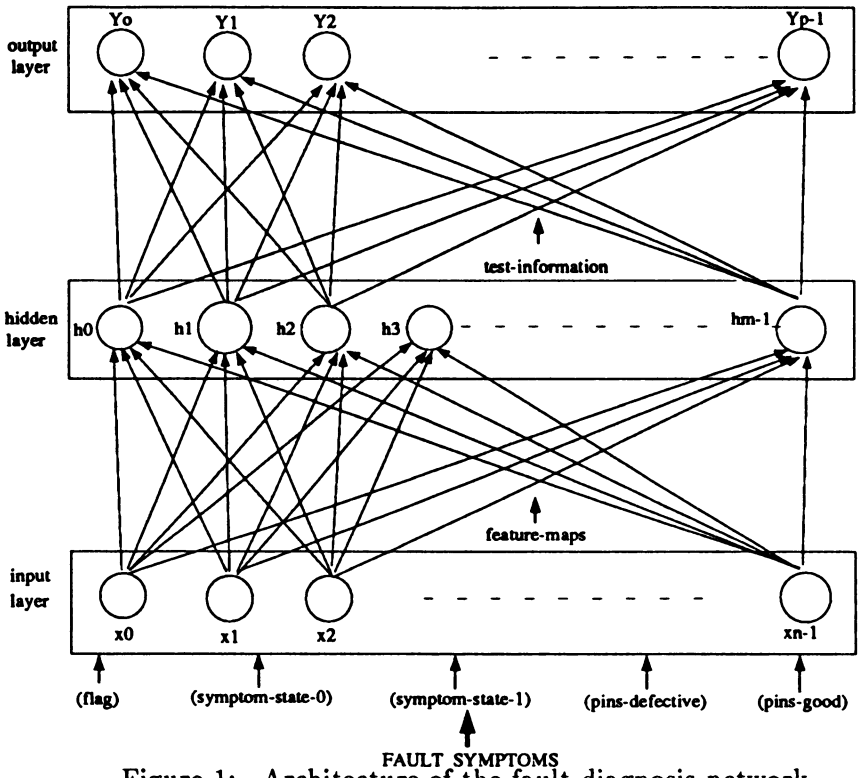


Figure 1: Architecture of the fault diagnosis network.

neurons in the next layer; that is, each neuron in one layer connects to every neuron in the next layer.

For the diagnostic system, the input-hidden layer pair is defined to be a binary linear network in which the input neurons are either on or off, and uses an unsupervised learning paradigm to form self organizing feature maps, formulated by Kohonen [1, 2]. The input representation for this layer consists of features, like symptom-state-0, symptom-state-1, pins which are observed to be bad while testing, pins which are good and a flag used to indicate whether the overall circuit is good or not. The other layer pair composed of the hidden layer and the output layer, uses a supervised learning paradigm based on the delta rule. The input to the highest layer is the feature map containing the knowledge about the fault symptoms, represented in topographical order. Nonlinearity is introduced by normalizing and a thresholding function **F**, as introduced later.

The Kohonen's network starts from the top level of the structural hierarchy of the circuit or device under test, learns the faulty state patterns, and forms feature map corresponding to each input pattern being presented to the network as follows:

The weights between the input and the hidden layer are initialized to small random values. The bit patterns representing the fault symptoms are presented to the network sequentially in time without specifying the output. After enough input vectors are presented, this process will lead to internal clustering of weights that sample the input vector space such that the point density function of the vector centers tend to approximate the probability density function of input vectors. This algorithm requires a neighborhood(or update-window) to be defined at different times as feature maps are being

formed. The neighborhood or the window size defines the set of nodes to be considered when updating the weights during clustering. The neighborhood starts large and slowly decreases over time. The weights are updated for node j and all nodes in the neighborhood defined by $window_j$(t) using the following equation:

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t) * (x_i(t) - w_{ij}(t))$$

For $j \in window_j$(t) $\qquad (0 \leq i \leq no.of inputs)$

The term $\eta(t)$ is a gain term $(0 < \eta(t) < 1)$ that decreases in time. The feature maps are formed by the product of the inputs and the corresponding weights. These feature maps are propagated to the second layer of the network where the system uses the structural descriptions of the circuit to learn the subset of components which might be responsible for the observed symptom-state due to the fault, at the next lower hierarchical level of the circuit. The delta learning rule is used at the second layer to learn the associations between the generalized symptom-states in the Kohonen's topological representation and the possible candidate modules at the lower level which might be responsible for the fault or which might lead to a test that can provide useful information leading to the fault. The training scheme is a little different here: For each pattern, the input units are turned on, and the effect they have on the output unit is observed. Its activation reflects the effects of the current connections in the network. The difference between the obtained output and the teaching input is measured. The weights in this layer are adjusted until the output converges to the teaching input using the following equation:

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t)(teaching\_input_i - output_i) * x_i$$

We have thus far considered information about fault symptoms that look only one measurement ahead in the neural network model. This model is able to take care of the condition where each symptom-state could have been produced by one of several faults, by incorporating feedback into the network. With feedback, the neural network mimics sequential conditional diagnostic searches which are taught to it and proposes tests that lead to identify the correct fault. This includes some basic search procedures such as binary search and searches based on probabilities of component failure. If we keep highly activated output cells and set the others to zero, we can then observe and keep track of movement of activity over time or sequences. This then require a propagation rule of the following form which retains about 50 % of the previous output state activity:

$$activity(t+1) = F(weights * activity(t) + 0.5 * activity(t))$$

where F is a nonlinearization function such as sigmoid. This rule has been tested on sets of sequences and has produced for us excellent generalization and recognition performance.

3 Sample Application of the Method:

Consider the example circuit shown in Figure 2. The block diagram represents the structure and function model of a 4-bit binary-full adder.

The symptom-states observed at the primary outputs(C4, S4..S1) and the carry outputs of the intermediate stages(C3..C1) are fed into the neural network. The symptom-state-0, symptom-state-1 and the flag inputs of the network are used in this example while others are set to zero. There are 17 input nodes to the network; eight for symptom-state-0, eight for symptom-state-1 and one for flag. The input to the hidden layer consists of 100 neurons corresponding to the 10×10 Kohonen's feature

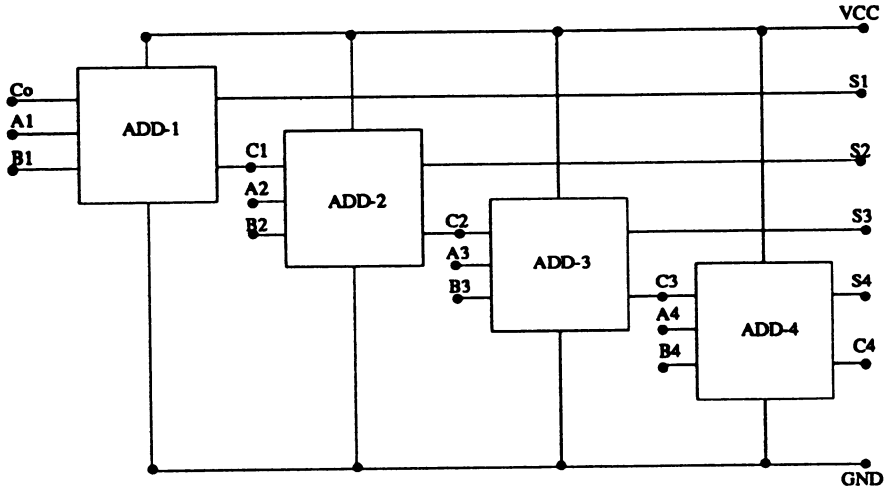


Figure 2: A 4-bit binary full-adder.

map produced by unsupervised learning. The output layer consists of 32 nodes containing information regarding where to test next. In one run we trained 55 input patterns which represent 55 decisions or branches made at 15 nodes of a search tree for diagnostic hypothesizing and the system was able to recognize them correctly. In other runs sequencing for hypothesizing the test nodes were reliably reproduced.

4 Conclusion:

The model and example presented here show that the neural network approach looks promising and efficient for guiding the search and forming fault and test hypotheses in diagnostic trouble-shooting of electronic circuits. Their parallelism, speed and trainability make them fault-tolerant, as well as fast and efficient for handling large amounts of data. A training from example paradigm such as the one presented here could help alleviate the knowledge acquisition bottleneck present in expert system based diagnostic systems.

References

[1] R. P. Lippman. *An Introduction to Computing with Neural Nets.* IEEE ASSP Magazine, vol. 4, pages 4-22, April 1987.

[2] T. Kohonen. *Self-Organized Formation of Topologically Correct Feature Maps.* Biological Cybernetics, 43:59-69.

[3] R. Davis, W. Hamscher. *Model-based Reasoning: Troubleshooting.* AAAI Invited Talks, 1986.

[4] P. Ibarra and S. Sahni. *Polynomially complete fault detection problems.* IEEE Transactions on Computer, C-24(3):242-249, March 1975.

FEASIBILITY OF USE OF A NEURAL NETWORK FOR BAD DATA DETECTION IN POWER SYSTEMS

S.A.Khaparde and Rita Mehta
Department of Electrical Engineering
Indian Institute of Technology, Bombay, India.

ABSTRACT

This paper presents feasibility studies of using a neural network for detecting the presence of bad data in a power system. The neural network model chosen is a three-layer perceptron using the back propagation training algorithm.Details of the implementation are given and the performance assessed.

1. INTRODUCTION

Bad data points are grossly erroneous rather than just slightly inaccurate. These could occur due to several reasons : a momentary failure of a communication link, an intermittent fault in a meter or an error in pseudo-measurements (a priori knowledge of certain variables). These errors need to be supressed before the data is used for state estimation. There are several methods available for the detection of bad data [1,2,3].

2. NEURAL NETWORKS

Neural network models are broadly based on our present understanding of the functioning of the human brain. Neural networks process information in a dynamic, self organising way and exhibit properties such as preferential learning, optimization and fault-tolerance which are usually associated with living systems [4,5].

The massively parallel structure of a neural network makes it inherentlly superior to a von Neumann computer in certain respects. These networks can generalise a common pattern from the presentation of a large number of examples. This is accomplished by virtue of their structure rather than through elaborate programming. Neural networks can abstract the 'ideal' from a non-ideal training set. They are high speed because of the massive parallelism.

There are several models of neural networks, each differing in the algorithm used for training. In this paper, the multi-layer perceptron is used. A multi-layer perceptron can be used as a classifier, classifying the input vector into one of the two classes. The propogation algorithm is outlined below [6].

1. Initialize weights $\omega_{ij}(0)$ and thresholds $\theta_j(0)$ to small random values.
2. Present inputs $x_i(t)$, $0 \leq i < N-1$, and the desired output $d(t)$.
3. Calculate the actual outpur $y(t)$.
4. Adapt weights starting at the output nodes and working back through the hidden layers.

$$\Delta\omega_{ij}(t+1) = \eta\,\delta_j\,X_i + \alpha\,\Delta\omega_{ij}(t)$$

η is a gain factor, α is a momentum factor. δ_j is an error term for for node j.

If node j is an output, then $\delta_j = y_j(1-y_j)(d_j - y_j)$

For an internal hidden node, $\delta_j = X_j(1-X_j) \sum_k \delta_k \omega_{jk}$

5. Repeat by going to step 2.

3. IMPLEMENTATION

Neural netwoks have certain advantages over the conventional computers when it comes to the problem of detecting bad data.

a) No elaborate algorithms are required for the detection of bad data. The neural network learns to do this from a large number of examples.

b) Processing of bad data has to be done in real time. Therefore parallel processing is highly desirable since the time required is independent of the number of inputs. On the other hand, in sequential machines, the processing time increases enormously with the increase in the number of inputs.

c) Here, the inputs are generated bynon-linear processes and are strongly non-Gaussian in the presence of bad data. In such cases, neural networks are more robust than conventional statistical classifiers.

d) Plenty of training data is available and hence training of the neural network is not difficult.

In particular, the perceptron model was chosen since it can handle continuous value data. Also since training data is available, supervised training is preferred. Besides the training procedure is simple and its implemntation is easy.

A three layer perceptron was developed using the backpropagation algorithm. The training data was used to adapt weights until the perceptron could correctly detect the presence of bad data in the training set. Then, a test set was used and the perceptron was expected to detect the presence of bad data based on its training.

4. RESULTS

The neural network program was run several times on the Cyber machine. The parameters studies were the gain factor, the momentum factor and the network architecture i.e. the number of nodes in the hidden layers. The performance criteria were the number of iterations required before the weights were adapted the classify the training sets correctly and the error in classifying test sets not used in training.

20 sets of input data, each set consisting of 20 inputs and the desired output for the set were used for training. The trained perceptron was fed with 20 sets of fresh input data.

Here,

η = gain factor
α = momentum factor
N1 = number of nodes in the first hidden layer
N2 == number of nodes in the second hidden layer

The following decision rule was used :

If $0.09 \leq x_i \leq 1.20$ for all x_i, then the data is good. If one or more of the x_i's are outside this range, then bad data is said to be present.

EFFECT OF GAIN FACTOR $\alpha = 0.9$, N1 = 15, N2 = 10

	no. of iterations for convergence	no. of errors with test data
0.10	363	1
0.12	364	0
0.15	324	0
0.17	315	0
0.20	309	0
0.23	595	0

EFFECT OF MOMENTUM FACTOR $\eta = 0.2$, N1= 5, N2 = 10

α	no. of iterations for convergence	no. of errors with test data
0.88	687	0
0.90	309	0
0.92	799	0

EFFECT OF N2 and N3 $\eta = 0.2$, $\alpha = 0.9$

N1	N2	no. of iterations for convergence	no. of errors with test data
18	10	245	0
15	10	309	0
12	10	145	0
10	10	352	0
8	10	340	0
12	12	145	1
12	8	157	0

For the above observations, we see that there exist optimum values

of η, α, N1 and N2 for which the convergence time is minimum. However, since training is a once and for all task, the convergence time is not so critical. We see that the three-layer perceptron performs extremely well for the test data not used for training.

5. CONCLUSIONS

In this paper, neural networks, and in particular, the perceptron model, have been discussed in brief. The massively parallel structure of neural networks makes them robust, high speed and fault tolerant. Processing is done in parallel and the time required is independent of the number of inputs. The perceptron model is used to detect the presence of bad data in power systems.

While elaborate algorithms are required to detect bad data using conventional methods, the neural networks approach is simple and straightforward. The simulated perceptron was able to classify the good and the bad data in test sets with 95-100% accuracy. From the results obtained, we can see that the convergence time depends on the gain factor, the momentum factor and the system architecture.

The studies carried out for the implementation of neural networks for bad data processing are in the preliminary stage. However, the results are very encouraging and indicate exciting possibilities. The spurt in research in neural networks and the rapid progress made in the hardware for these, lead us to envisage a neural network processor being used to tackle the problem of bad data in power system.

REFERENCES

1. L.Mili, Th. Van Cutsem and M.Ribben-Pavella, 'Bad data identification methods in power system state estimation - A comparative study', IEEE Transactions on PAS, Vol.104, Nov.1985.

2. Doraiswami R., "An algorithm for pre-filtering of gross measurement errors in power system state estimation', 7th PSSC Laussane, July 1981.

3. M.M.F.Sakr, E.Lerch and H.A.Naur Eldin, 'Pre-estimation and bad data detection in power system', IEE Proceedings of International Conference on P.S.Monitoring and Control, U.K.1986.

4. Gary Josin, 'Integrating neural networks with robots', AI Expert, August 1988.

5. Richard Lippmann, 'An introduction to computing with neural nets', IEEE ASSP Magazine, April 1987.

6. James L.McClelland and David E.Rumelhart, 'Explorations in parallel distributed processing', The M.I.T.Press 1988.

DESIGN OF A POLE-BALANCING CONTROLLER USING NEURAL NETWORKS

Yoo Seok Kim and Jang Gyu Lee
Department of Control and Instrumentation Engineering, Seoul National University, Seoul, Korea

Abstract

Historic pole-balancing problem is simulated using a neural network which imitate the control intent of human and its machanisms, and we implement it by a computer interfacing. In order to solve this problem, the neural network gets basic control objective and operation as preknowledgement and uses the reinforcement learning algorithm for better performance. This method can be related to the man-machine operation via control machanism which human performs.

Introduction

Control application of neural network is represented by an automatic motor cotrol using artificial perception. It is an intelligent , pattern recognized control which are made by a process and experience of an animate creature. Thus to accomplish such a control ability, neural network must implement the control intent and the process of human. In this paper, we intend to solve the representative pole-balancing problem of control application by such method. Pole-balancing problem defined in this paper is an action for a pole having one dimensional degree of freedom not to fall down, thus exhibiting a control process of human being. This problem has been a subject of many neural network researchers.

Recently, Widrow[2] studied "trainable expert system" or a new kind of adaptive computer using a method in which the computer learns to associate visual inputs and the correspanding control data of a skillful human. Guez[3] suggests that it is possible to outperform human ability by the filtering of teacher training data dependent on the dynamic model of the teacher and by compensating for a biophysical special feature of a human, using Trainable Adaptive Controllers (TACs). These two studies use the control behaivor of human as important training data. Barto[4] proposed the associative search element (ASE) and adaptive critic element (ACE) using indirect performance evaluation in the state of no pre-knowledgement about the problem and solved this problem by developing a more informative evaluation function. Anderson[5] adds a secondary adaptive layer learning the tranformation of the state variables on the Barto's original layer in which an evaluation function is formed. They focused on the formation of an evaluation function necessary to solve the problem and used reinforcement learning mechanism. Ritter[6] solved the problem by making a mapping of a state space into a total control space using topology conserving mapping appearing in the biological procession of sensual inputs. All the proceeding studies were made via simulation.

In this study, we propose the algorithm in order to mimic the process of solving the pole-balancing problem by human. That is the method which makes a better performance by reinforcement learning rule given the aim of balancing and basic operation as preknowledgement. We demonstrate it by an experiment that this algorithm is suitable to a real time computer interface.

The model

For information coding of a sensor, angle of the pole is divided into ±1, ±3, ±6, ± 9, ±12 degrees from a balancing point, and each region takes charge of the place code[7] as 10 neurons. Also, the position of the cart is divided into ±0.8m, ±1.6m, ±2.4m from zero position, and they are assigned to 6 neurons. In ether cases, an equal number of neurons are assigned to the right and left positions. At an instant of time, a neuron from the pole and another neuron from the cart are participating to produce an output to make an action. Because the control objective is to balance the pole and to position the cart to the zero position, it is a reasonable assumption that the cart must move to the right if the pole is located at the right. We don't take the velocity of the pole as the code information and only use it in the calculation of the motor action. It is the same process for the motor as a human in which the sensual coordination is transformed directly to the space coordination. The initial command of this action is made by the spatial transformation. But, at the result of this action, the position of the cart is biased to one direction, and, in the process, an unbalanced action and movement of the pole is brought. Thus, we have the 6 neurons of the cart taken part in the calculation of the force with weight at the intention of overcoming this illbalancing state. It is the combined sensual stimulus or interpretation of the action intent that makes the motor error decreased on the basis of information between the spacial objective and current position. This step combines two sensual data and forms the pattern code[7]. Neurons of the pole and cart learn by the reinforcement learning rule independently. That is, at the degree of improvement of performance between the current state and earlier state, a credit or a blame is taken. Thus the operation is made by temporal position and angle difference. Due to the right and left division of pole and cart, the weight of each neuron can not have a minus value. The weight begins with a high offset value because it reduces to the direction of inhibition in a learning process[8]. Given the objective of balancing and the control intent of the basic action as pre-knowledge, a fine control is accomplished via learning. In the following, the course of learning algorithm appears :

angle : $w(t+1) = w(t) + 0.9 * rein * abs[theta(t)]$
$rein = hat1 - hat2$
$hat1 = 0.20944 - abs[theta(t+1)]$
$hat2 = 0.20944 - abs[theta(t)]$

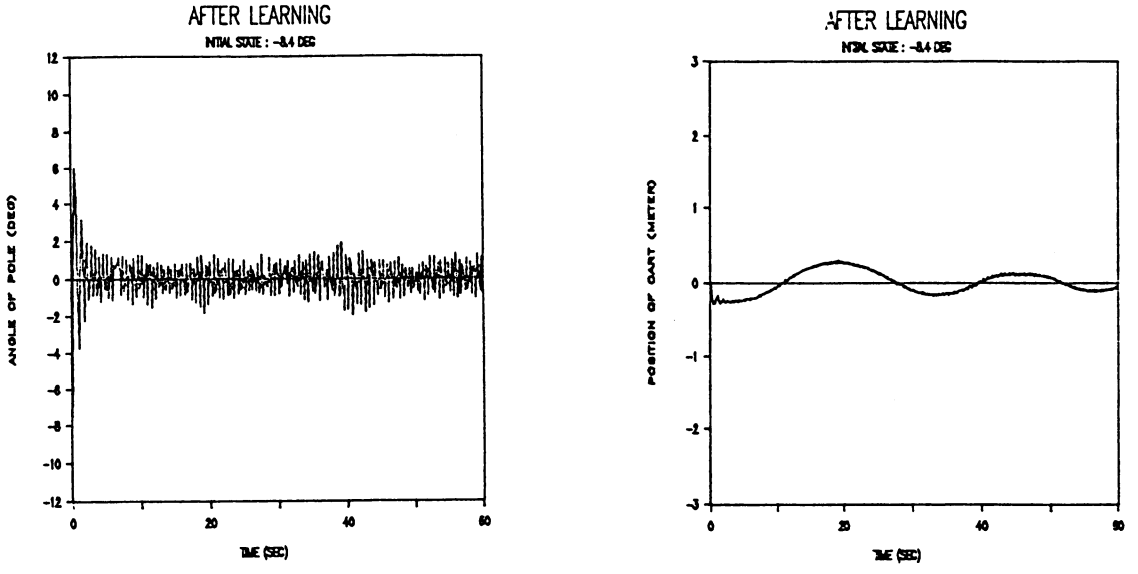
where the angle is mearsured in radian, w is weight and theta is an angle. 0.20944 is the radian value of 12 deg. Abs stands for absolute value.

position : $hw(t+1) = hw(t) + 0.09 * rein1 * abs[dis(t)]$
$rein1 = abs[dis(t)] - abs[dis(t+1)]$

where dis is a position and mearsured in meter, and rate 0.09 is scaled in order to get the same scale as angle.

Experiment and Result

Simulation starts from a random initial pole state and fails if pole exceeds ±12 deg or cart exceeds ±2.4 m. New trials are restarted from a different initial pole state. Learning is accomplished in the total pole state space. Output function is the threshold logic, which is similliar to a human action. Figure 1 depicts the angle of the pole (a) and the position of the cart (b) starting from the -8.4 deg of initial pole state. The figure (a) shows that the angle of the pole is kept within ±1 deg and (b) shows that the cart moves around the zero position. Simulation rate is 50 Hz.



(a) (b)

Figure 1 : Results of simulation

Actual cart-pole system is composed of stepping motor, encoder for mearsuring the angle of pole, the cart mechanics and motor driving circuit, which are connected to an IBM PC computer via an interfacing board. The motor rotates 1.8 deg per pulse. The encoder can measure up to 0.12 deg per pulse. The interfacing board senses the angle of pole from the encoder signal and sends the control signal to the motor. An angular velocity of pole is calculated by an the average velocity over one step and position of cart is calculated by counting the number of clocks and the direction of the stepping motor.

After learning is completed, only output function is enough to control this problem. weight updating is not necessary, and any initial pole state goes rapidly stable region by this control mechanism. The clock of motor is calculated from the cart speed and the simulated force. Real time one cycle take about 30 msec.

Conculusion

It is shown in this paper that, given a basic control objective and action as a control intent, a neural network learning to obtain a skillful control process can produce an appropriate control law for the pole-balancing problem. In this problem, the control intent of a human is less delicate, but this keeps learning velocity fast by coding right information. The process of learning proceeds mainly in the direction of inhibition. It is simillar to the dexterity appearing in the experiment of a human action which is formed via this inhibition. In biophysics, the climbing fiber representing the control error of performance and the parallel fiber inhibit the transmission of the purkinje cell of the cellibelum during the long time[9,10]. It is demonstrated that a suitable control law representing a human undistorted, right control intent can be designed in real situation by neural networks. The present work is aimed at investigating an implementation of a control law of human using various adaptive networks, applying various learning rules for a fine control.

Reference

[1] Richard P. Lippmann, " An Introduction to Computing with Neural Nets," IEEE ASSP Magazine, April 1987.

[2] Bernard Widrow and Viral V. Tolat, " An Adaptive 'Broom Balancer' with Visual Inputs," IEEE Proceedings of ICNN, 1988.

[3] Allon Guez and John Selinsky, " A Neuromorphic Controller with a Human Teacher," IEEE Proceedings of ICNN, 1988.

[4] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson, " Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems," IEEE Tran. on SMC-13, 1983.

[5] Charles W. Anderson, " Learning to Control an Inverted Pendulum with Connections Networks," Proceedings of ACC, Vol.3, 1988.

[6] H. Ritter and K. Schulten, " Topology Conserving Mappings for Learning Motor Tasks," Proceedings of AIP, 1986.

[7] Kermit T. Hoyenga, Psychobiology: The Neuron and Behavior, Brooks/Code Publishing Company, 1988.

[8] A. Harry Klopf, " Derive-Reinforcement Learning: A Real-Time Learning Mechanism For Unsupervised Learning," Proceedings of IEEE ICNN, 1987.

[9] Philip H. Abelson, Neuroscience, The American Association for the Advancement of Science, 1985.

[10] A Scientific American Book, The Brain, W. H. Freeman and Company, 1979.

Application of Neural Network to Information Retrieval

K.L. Kwok, Dept. of Maths. & Computer Science
Western Connecticut State University, Danbury, CT 06810

1. Introduction

Given two items, a document and a query both in natural language, an information retrieval (IR) system tries to decide if they are dealing about the same concepts. If so it concludes that they are relevant, else not. This strict decision is difficult because of the ambiguities of texts. The usual practice is to order documents based on a similarity measure to the query, and a stopping strategy is applied on this ranked list, dividing the collection into a relevant and a non-relevant set. Many models are possible (see [1] for review). We focus on the probabilistic model [2,3], which in practice is based on ranking, relative to a query $q_a=(q_{a1}.. q_{ak}.. q_{am})$, via the log odds $\ln[P(d_j|+R)/(1-P(d_j|+R))]$ that within a relevant (+R) and non-relevant (-R) sample of documents that one would find the feature representation of a document $d_i=(d_{i1}.. d_{ik}.. d_{im})$. Features are content-bearing terms derived automatically from the collection, numbering m. Assumptions are then usually taken, e.g. items are represented by the presence/absence of terms (d_{ik}, q_{ak} = 1/0) and that they are statistically independent. Optimal retrieval means optimal ranking in this sense based on the available samples. Extensions to the theory [4,5] by considering each item to be constituted of conceptual components leads one [6] to overcome some shortcomings such as: allow the theory to self-bootstrap, account for the within-item frequencies d_{ik}, q_{ak}, and include the effects of the analogous situation of probabilistic indexing. The end result is that we can provide optimal ranking of documents based on the following:

$$WQ_i = \sum_k (d_{ik}/L_i)*g_{ak}, \quad \text{where}$$

$$g_{ak} = g^r_{ak} + g^s_{ak} = \ln [r_{ak}/(1-r_{ak})] + \ln [(1-s_{ak})/s_{ak}] \quad (1)$$

$$WD_i = \sum_k (q_{ak}/L_a)*g_{ik}, \quad \text{where}$$

$$g_{ik} = g^r_{ik} + g^s_{ik} = \ln [r_{ik}/(1-r_{ik})] + \ln [(1-s_{ik})/s_{ik}] \quad (2)$$

$$W_i = WQ_i + WD_i \quad (3)$$

WQ_i ranks documents based on the perspective that given a query, what documents are probably relevant [2]. WD_i correponds to probabilistic indexing [7] with the perspective that given a document, what queries are probably relevant. W_i accounts for both, similar to [8]. L_i, L_a are the lengths of each item, and r_{ak}, s_{ak} are the probabilities that given relevance or non-relevance that term k will be present: P(term k present$|\pm$R). They are estimated depending on how much feedback (i.e. the relevant sample $\{d_j\}$ or $\{q_b\}$) is available. The non-relevant sample can be estimated by the rest of the universe. For example, the general formulae for query q_a with n_a-1 feedback documents are:

$$r_{ak} = 1/n_a*[q_{ak}/L_a + \sum_j d_{jk}/L_j], \qquad s_{ak} = \sum_i d_{ik}/N_w \quad (4)$$

N_w counts the total number of index terms used, and $\sum_i d_{ik}$ is the collection term frequency of term k. q_a is self-relevant and the formulae provides continuity starting with no feedback.

2. A Neural Network for Probabilistic Information Retrieval

A 3-layer neural network for the previous theory is shown in Fig.1. Layers Q and D contain neurons to be identified with each query and document. They also serve for external input and output. Hidden layer T neurons are identified with each unique term and are connected to both Q and D bi-directionally with asymmetric strengths. Intra-layer connections are disabled in this report. Operation is feed-forward (QtD) or feed-backward (DtQ) only.

Connection strengths are initially assigned as follows. From an item to a term neuron t_k, it is given by $w_{ki} = d_{ik}/L_i$ ($w_{ka} = q_{ak}/L_a$), and can be obtained from the text. During retrieval neuron k receives signal from an input neuron and leads to an activity $a_k = F(net_k)$. It is known [9] that the power of linear nets are limited. Based on experiments, we also found that a non-linear activation function is crucial for effective retrieval. This function is taken as a family of ramps: $F(d_{ik}/L_i)= (1/HIGH + d_{ik}/L)$, where L=LOW if L_i < LOW, else L= L_i. HIGH and LOW are constants justified from document text length normalization. From a term neuron k to an item the connection strengths w_{ik} (w_{ak}) are taken exactly as the g's of Eqn. 1,2. The s_{ik} (s_{ak}) factor is a constant approximately, while the r_{ik} (r_{ak}) factor can learn based on available feedback. Initially, it is assigned a small constant p. The activation function at the output neurons is also given by a linear ramp with upper and lower bounds. The output signal and the activation of a neuron is taken to be identical.

Ranking for retrieval goes as follows. Corresponding to WQ_i of Eqn.1, we focus attention on q_a, clamp each d_i with activity 1 in turn, spreading it and observe the activity received at q_a (DtQ feed-backward). By focussing attention on each d_i in turn, with q_a clamped to 1, we recover WD_i, Eqn.2 (QtD feed-forward). Adding the activities received at the pair q_a and d_i from the above recovers W_i (sym) of Eqn.3.

3. Learning Algorithms

Feedback (which documents are relevant to which query) forms the clues to be learnt for the r_{ak} (r_{ik}) factor of the connection weights w_{ak} (w_{ik}). A learning phase goes as follows. Fig.1 also shows a set of documents (d_j) given relevant to q_a and has activity clamped to 1. q_a being self-relevant is also clamped to 1, forming a relevant set of size n_a. After one time step, a term neuron k connected to them will receive an activity $a_k = w_{ka} + \sum_j w_{kj}$. a_k/n_a is now the best estimate of the probability P(term k present|R) based on knowledge currently available. A teaching signal of $t_a=1$, and $t_x=0$ elsewhere, is applied on q_a because this is the relevant item of the set. A Hebbian type correlational learning algorithm operates thus:

$$\text{A: } \Delta w_{ak} = h(t_a, a_k, w_{ak}) = t_a * \Delta r_{ak} / [r_{ak} * (1 - r_{ak})], \qquad (5)$$

where $r_{ak} = \exp(w_{ak} - w^s_{ak}) / [1 + \exp(w_{ak} - w^s_{ak})]$ is the probability just before learning. Learning iterates gradually with a rate of eta, and at the (v+1)-th iteration it is:

$$r_{ak}^{v+1} = (1-\text{eta}) * r_{ak}^{v} + \text{eta} * a_k/n_a, \quad 0<\text{eta}<1, \quad r_{ak}^{0} = p. \qquad (5a)$$

As v -> infinity, we see that $r_{ak}^{v} \to a_k/n_a$. For each iteration step the change in r_{ak} is therefore: $\Delta r_{ak} = \text{eta} * (a_k/n_a - r_{ak})$. Simultaneously, the other link w_{ka} also learns according to

$$\text{B: } \Delta w_{ka} = h'(t_a, a_k, w_{ka}) = t_a * \text{eta} * (a_k/n_a - w_{ka}). \qquad (6)$$

Rule B dynamically affects the result of rule A. It adjusts the term k proportion of query q_a towards that of the relevant set during a learning episode (v,eta).The same learning algorithm also applies to w_{ik}, w_{ki}. Currently, this takes place separately from the previous learning process.

4. Experimental Results

Such a network has been built and applied to the 4 collections popular with IR research and which have relevance judgment: MED(30q,1033d) (Medical, 30 queries 1033 documents),CACM(52q,3024d) (Computer), CISI(76q,1460d) (Information science) and CRAN(225q,1400d) (aerodynamics). Three different stages of learning were done. Stage 1 has no learning, with $r^0_{ak} = r^0_{ik} = p = 1/40$; this we call Inverse Collection Term Frequency (ICTF) formula and is analogous to the Inverse Document Frequency (IDF) formula of [10] and popular among IR work. Stage 2 involves item self-learning using a learning episode of (20,0.5). Stage 3 involves full feedback learning and assumes knowing all relevant documents to every query, with a learning episode of (30,0.5). Retrieval is done after each learning stage and evaluated using the average precision at ten standard recall points. Table 1 displays a summary of our experiments. At Stage 1 with no learning, ICTF performed substantially better than IDF across all 4 collections. Thus, within-item term frequencies are important, while IDF counts only presence/absence of terms. After Stage 2 self-learning, we obtained typically a few percent better effectiveness than before. Our results are much more stable and better than Croft's [11], and substantially similar to that of Salton's [12]. After Stage 3 full feedback, an interesting phenomenon was observed. Feed-backward (DtQ) and feed-forward (QtD) learning independently gave results fairly similar to each other, and to Croft's. However when both are accounted in one symmetric formula, it produces cooperatively substantially better effectiveness. So far, IR research has been using DtQ feedback only. When both types of information are simultaneously used, results outpreform Croft's formula from 13% (MED) to over 50% (CISI).

5. Conclusion

Probabilistic indexing and retrieval theory in IR may be implemented using a neural network. More sophisticated techniques for optimization such as the Hopfield net [13], the Boltzmann machcine [14] or the Harmony theory [15] may be used if we switch on the interations in layer T, or that of layer D. By building a bridge from the traditional IR model to this new formalism, we further achieve the advantages that the network can always default to known results and that sophisticated evaluation methods popular in IR work can still be available.

Acknowledgment

Computation was performed on the Cornell Supercomputer Facility, a resource of the Center for Theory and

Simulation in Science and Engineering at Cornell University, which is funded in part by NSF, New York State, and the IBM Corporation and members of the Corporate Research Institute. This work was performed when the author was a recipient of a CSU/AAUP research grant.

References

1. Salton, G, Automated Text Processing: the transformation, analysis, and retrieval of information by computer. Addison-Wesley, 1989.
2. Robertson, SE & Sparck Jones, K, 'Relevance weighting of search terms'. J.ASIS, 27:129-46, 1976.
3. van Rijsbergen, CJ, 'A theoretical basis for the use of co-occurrence data in information retrieval'. J.Doc 33:106-19, 1977.
4. Kwok, KL, 'A probabilistic theory of indexing and similarity measure based on cited and citing documents'. J.ASIS 36:342-51, 1985.
5. Kwok, KL & Kuan, W, 'Experiments with document components for indexing and retrieval'. Info.Proc.Mgmnt. 24:405-17, 1988.
6. Kwok, KL, 'A neural network for probabilistic information retrieval'. In: Belkin, NJ & van Rijsbergen CJ (ed.) Proc. 12th Annual Intl. ACMSIGIR Conf. on R&D in IR. p.21-30, 1989.
7. Maron, ME & Kuhn, JL, 'On relevance, probabilistic indexing and information retrieval'. J.ACM 7:216-44, 1960.
8. Robertson, SE, Maron, ME & Cooper, WS, 'Probability of relevance: a unification of two competing models for document retrieval'. Info.Tech: R&D, 1:1-21, 1982.
9. Rumelhart, DE, McClleland, JE & The PDP Research Group, Parallel Distributed Processing: Explorations in the Microstructure of Cognition. MIT Press, 1986.
10. Sparck Jones, K, 'A statistical interpretation of term specificify and its application in retrieval'. J.Doc 28:11-20, 1972.
11. Croft, WB, 'Experiments with representation in a document retrieval system'. Info.Tech: R&D 2:1-21, 1983.
12. Salton, G & Buckley, C, 'Term weighting approaches in automated text retrieval'. TR 87-881, Comp.Sci. Dept, Cornell Univ., 1987.
13. Hopfield, JJ, 'Neural networks and physical systems with emergent collective computational abilities'. PNAS(USA) 79: 2554-58, 1982.
14. Hinton, GE, Sejnowsky, TJ & Ackley, DH, 'Boltzmann machines: constraint satisfaction networks that learn'. TR CMU-CS-84-119, Comp.Sci. Dept, Carnegie-Mellon Univ., 1984.
15. Smolensky, P & Riley, MS, 'Harmony theory: problem solving, parallel cognitive models, and thermal physics'. TR 8404, Inst. of Cognitive Sci., UCSD, 1984.

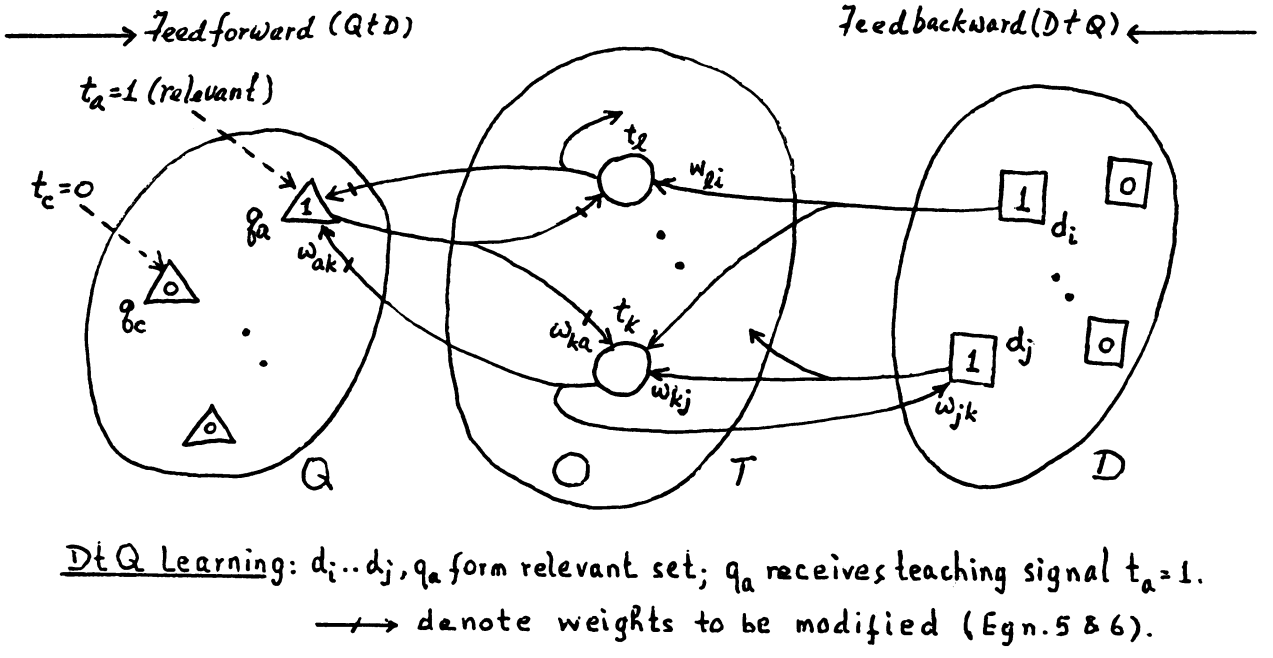


Fig.1: 3-Layer QTD Net, Connection Strengths, and Learning

	MED			CACM			CISI			CRAN		
	DtQ	QtD	sym	DtQ	QtD	sym	DtQ	QtD	sym	DtQ	QtD	sym
Stage 1 -- no learning												
ICTF :	.484	.420	.472	.276	.247	.297	.161	.165	.174	.399	.292	.374
IDF :	.455			.210			.128			.319		
Stage 2 -- self-learning												
NN :	.489	.476	.487	.277	.320	.309	.181	.201	.194	.400	.394	.404
Crofts:	.493			.255			.147			.390		
Salton:			.505			.265			.203			.389
Stage 3 -- full feedback learning												
NN :	.630	.689	.734	.477	.445	.575	.395	.437	.602	.503	.603	.666
Crofts:	.649			.446			.392			.497		

Table 1: Average Precision over 10 Standard Recall Points

Combinatorial Optimization Using Competitive-Hopfield Neural Network

Bang W. Lee and Bing J. Sheu

Department of Electrical Engineering
and Center for Neural Engineering
University of Southern California, Los Angeles, CA 90089-0271

Abstract

Combinatorial optimization required in many engineering problems can be performed using massively connected neural networks. In the Hopfield network approach, the constraint functions and objective function of a combinatorial optimization are combined in the energy function and coded into synapse weightings. Due to the complexity of the network, there is a severe limitation on the scalability of this approach to large-size problems. In this paper, a novel competitive-Hopfield neural network which utilizes a separate competitive network to realize the constraint functions is proposed. The combined competitive-Hopfield network always converges a valid solution in a reduced computational time. The adaptive time-step control technique has also been developed to avoid invalid solutions due to accumulated error of the fixed-time step technique at the network evaluation. Several numerical schemes to solve differential equations using the competitive-Hopfield network have been explored. Experimental results on the Traveling Salesman Problem show that the competitive-Hopfield network always gives valid solutions which are not sensitive to the selection of weighting factors in the energy function.

I. Introduction

Electronic neural networks are quite popular for solving heuristic problems which include pattern recognition, signal processing, and optimization. The immense computational power is derived from the massively parallel architecture and the adapting capability through a learning process. Hopfield network [1], which consists of one layer neuron and fully connected synapses, is one of widely used network due to simple network architecture and well-defined network dynamics. Hopfield network always converges along the direction of decreasing the energy function. The stable outputs are logical values, while the inputs can be analog signals. Given this property, Hopfield network is very suitable to solve combinatorial optimization problems [2]. Many of these problems are often NP-complete, indicating that only an acceptable solution can be obtained within a reasonable computational time.

For a n-city traveling salesman problem (TSP) which finds a minimum round trip distance visiting all cities once, n^2 neurons are required to represent a tour sequence. The energy function for the Hopfield network can be constructed as

$$E = \frac{A}{2}\sum_{X=1}^{n}\sum_{i=1}^{n}\sum_{j\neq i\,j=1}^{n} V_{Xi}V_{Xj} + \frac{B}{2}\sum_{i=1}^{n}\sum_{X=1}^{n}\sum_{Y\neq X,Y=1}^{n} V_{Xi}V_{Yi}$$
$$+ \frac{C}{2}\left[\sum_{X=1}^{n}\sum_{i=1}^{n}V_{Xi} - n\right]^2 + \frac{D}{2}\sum_{X=1}^{n}\sum_{Y\neq X,Y=1}^{n} d_{XY}V_{Xi}(V_{Y,i+1} + V_{Y,i-1}) \,. \qquad (1)$$

Here, X and Y denotes cities, i and j are part of the tour-procedure, and d_{XY} is the distance between cities X and Y. The first three terms in (1) are the constraint functions for a valid tour, while the last term is the total tour distance which is to be minimized. Heuristic weighting factors A, B, C, and D should be carefully chosen to find a valid and optimal tour. It has been reported that the choice of the

This research was partially supported by DARPA under Grant No. F29601-87-C-0069, by AT&T Company, and by USC-Biomedical Research Support Grant from NIH.

Table 1. Average CPU time of TSP on SUN 3/60

Number of city	CPU time
5	1.5 minutes
10	26 minutes
20	490 minutes

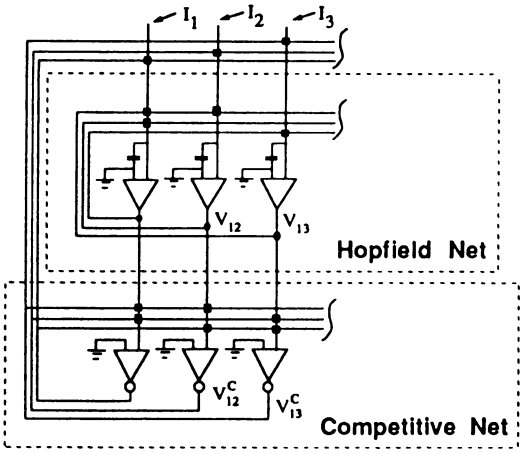


Fig. 1 The competitive-Hopfield neural network.

weighting factors for a valid solution becomes very difficult as the number of city increases [3,4]. This is caused by the mixture of the constraint functions and the objective function in the energy function. During the searching process of decreasing the energy function, there is no distinction between the constraint functions and the objective function so that the final converged solution is usually invalid when the number of cities is large. Some previous results for the TSPs have been reported [5,6]. However, more efforts need to be devoted to this subject.

II. The Competitive-Hopfield Network Approach

A new solution consists of the utilization of a Hopfield network and a competitive network has been developed. The constraint functions are implemented in the competitive network, while the energy function with reduced constraint functions is implemented in the original Hopfield network as shown in Fig. 1. The competitive network monitors the outputs of Hopfield network. Once the outputs are larger than a threshold voltage of neurons in the competitive network, the network starts to search for the highest output through the competing process with the 'winner-take-all' strategy. Here, signal delay in the competitive network is much smaller than that in the Hopfield network. Since the constraint functions of a combinatorial optimization problem are realized by the competitive network, the objective function in the energy function of Hopfield network can be maximized. Hence, solution searching process in the Hopfield network can be mainly determined by the objective function.

For a n-city TSP, the synapses $\{ T^H_{Xi,Yj} \}$ and input currents $\{ I_{Xi} \}$ for the Hopfield network are given as

$$T^H_{Xi,Yj} = -A\,\delta_{XY}(1-\delta_{ij}) - B\,\delta_{ij}(1-\delta_{XY}) - C - Dd_{XY}(\delta_{j,i+1} + \delta_{j,i-1}) \tag{2}$$

and

$$I_{Xi} = Cn + (A + B + C + D)V^C_{Xi}, \tag{3}$$

while the synapses $\{ T^C_{Xi,Yj} \}$ for the competitive network are given as

$$T^C_{Xi,Yj} = \delta_{XY}(1-\delta_{ij}) + \delta_{ij}(1-\delta_{XY}) . \tag{4}$$

Here, V^C_{Xi} is the output of the competitive network and δ_{ij} is 1 if $i=j$ and is 0 otherwise. Notice that the weighting factors A, B, C, and D are the same as in (1).

The dynamics of the Hopfield network can be described in the following differential equation.

$$C_{Xi}\frac{du_{Xi}}{dt} = \sum_{Y\neq X, Y=1}^{n}\sum_{j\neq i\ j=1}^{n} T^H_{Xi,Yj}V_{Yj} - T_{Xi}u_{Xi} + I_{Xi} \ , \tag{5}$$

where T_{Xi}, u_{Xi}, and C_{Xi} are the equivalent conductance, input voltage, and input capacitance at the i-th amplifier input node, respectively. The Hopfield network is always guaranteed to decrease the energy function [1]. A simple Euler integration technique is often used to evaluate the differential equations [3]. But, the accumulated numerical error is the cause of a wrong solution. We have used a adaptive time-step control technique which adjusts the integration time-step by monitoring the local error [7].

III. Experimental Results

In our experiments, the traveling salesman problem were simulated in a SUN 3/60 and the city locations was generated randomly. For simplicity, capacitances { C_{Xi} } in (5) are set to be 1 and amplifier input voltages { u_{Xi} } are initially reset. Two versions of the forward-Euler integration methods were examined: the fixed time-step version and the adaptive time-step version. Figure 2 shows energy function change as the function of time in a 10-city TSP. With the fixed time steps of 1.0e-3 and 1.0e-4, the energy function is no longer continuously decreasing due to the numerical errors. The convergence time using fixed time-step is also adversely longer. Here, the CPU time for the fixed time-step being 1.0e-4 is about 5 hours while that for the adaptive time-step with 1.0e-3 error tolerance is only 0.5 hour. Table 1 shows the averaged CPU time for different size of TSPs. Figure 3 shows the solutions with different weighting values of the constraint terms. As the weighting factors for the constraint functions decrease, the tour distance tends to decrease. This is caused by the fact that the searching process in the Hopfield network is mainly determined by the objective function. Our experimental results show that the competitive-Hopfield network always gives valid solutions when the weighting factors for constraint functions are greater than 1.0

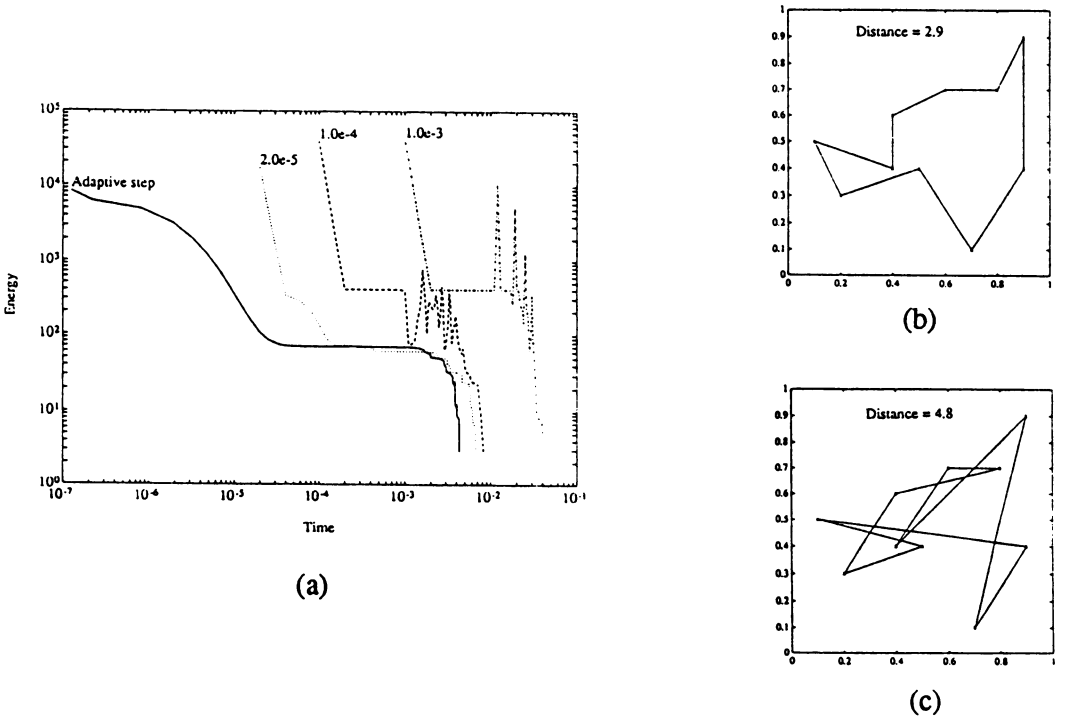
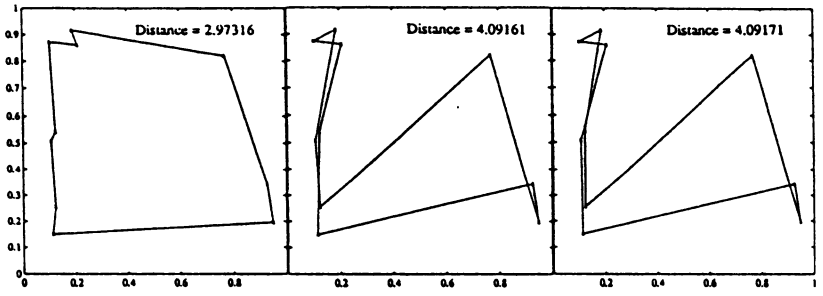
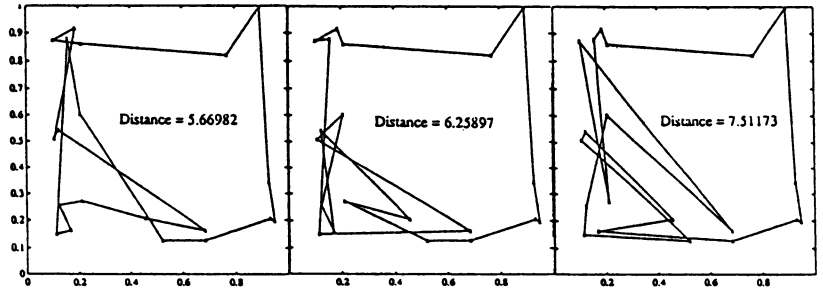


Fig. 2 Network dynamics for a 10-city TSP.
(a) Transient behavior. (b) Solution with adaptive time-step and with fixed time-step of 1.0e-4 and 2.0e-5. (c) Solution with fixed time-step of 1.0e-3.
The fixed time-step methods require a lot of computational time to obtain the solution.



(a)



(b)

Fig. 4 TSP solution with different weighting factors.
(a) 10-city TSP. (b) 20-city TSP.
As the weighting factors of the constraint functions decrease, tour distance decreases.

IV. Conclusion

We have successfully developed and demonstrated a novel competitive-Hopfield neural network for the combinatorial optimization. The network is very effective and efficient in finding consistently valid, near-optimal solutions for a high-city TSP. This competitive-Hopfield network can be extended other combinatorial optimization in signal processing with complicated constraint functions.

References

[1] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceeding of the National Academy of Sciences, U. S. A.*, vol. 79, pp. 2554-2558, 1982.

[2] J. J. Hopfield and D. W. Tank, " 'Neural' computation of decisions in optimization problems," *Biol. Cybernetics*, vol. 52, pp. 141-152, 1985.

[3] G. V. Wilson and G. S. Pawley, "On the stability of the traveling salesman problem algorithm of Hopfield and Tank," *Biol. Cybernetics*, vol. 58, pp. 63-70, 1988.

[4] A. B. Kahng, "Traveling salesman heuristics and embedding dimension in the Hopfield model," *Inter. Joint Conf. on Neural Networks*, vol. I, pp. 513-520, June 1989.

[5] D. E. Van den Bout and T. K. Miller, "A traveling salesman objective function that works," *IEEE Inter. Conf. on Neural Networks*, vol. II, pp. 299-303, July 1988.

[6] Y. Akiyama, A. Yamashita, M. Kajiura, and H. Aiso, "Combinatorial optimization with Gaussian Machines," *Inter. Joint Conf. on Neural Networks*, vol. I, pp. 533-540, June 1989.

[7] G. Dahlquist and A. Bjorck, *Numerical Methods*, pp. 330-350, Prentice-Hall Inc., 1974.

A New Model for Concept Classification Based on Linear Threshold Unit and Decision Tree

Hahn-Ming Lee and Ching-Chi Hsu
Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan

1. Introduction

Classification is an important intelligent behavior. It has been an active research issue in the knowledge-based reasoning, pattern recognition, and connectionist paradigms[1]. One would like to observe positive and negative instances of a concept and then to propose a representation that are both suitable for the observed instances and useful to predict the classification of unobserved instances. This is one important kind of concept learning.

A new neural network model called quasi-LTU-decision model is presented here, which performs very well to the concept classification described above. This model is based on the concept of decision tree and the operation of a simple neural network unit - linear threshold unit (LTU) which is the basic unit of Rosenblatt's perceptron[6]. They are chosen because the decision tree and the LTU are natural choices in terms of handling a large amount of instances. Unlike other models in which the network topology must be specified before training, in this model the network structure is generated during training by non-incremental or incremental learning. The training instances are not restricted to be linearly separable which means there exists a hyperplane to discriminate the positive and negative training instances. The nonlinearly separable instances set can also work well in this model. Moreover, the learning algorithms associated to this model can handle a large volume of training instances efficiently and incrementally, and the resulting concept represented by the network model can also efficiently classify the unobserved instances.

2. Quasi-LTU-Decision Model

This section describes a method in construction a hybrid representation and the associated learning algorithms. This representation not only can be generated by a set of existing training instances, but also can be created by incremental learning, i.e. multistage learning which information learned at one stage is modified to accommodate new facts provided in subsequent stages. Besides, the space of instances to be trained is not restricted to be linearly separable.

Throughout this paper, an instance is described in terms of a set of attributes, which are binary values. Given a set of such instances and their classifications, the task is to distinguish these instances which belong to some class from those which do not. Without loss of generality, we assume the output is a binary value that represents acceptance or rejection of the input instance.

2.1 Quasi-LTU-decision Representation

The quasi-LTU-decision model consists of one multi-branch decision node and many quasi-LTU nodes. The network structure in the model is generated during training. There is a connection from the decision node to each of the quasi-LTU nodes. Attached to each connection is an attribute set belonging to its corresponding quasi-LTU node. An attribute set consists of a set of positive or negative attributes used to test for branching. These attribute sets are mutual-exclusion and the union of these attribute sets covers all the instances space, i.e. during the presentation of an input vector I, one and only one attribute set will pass the attribute test to the input vector I. The complete model structure is illustrated in figure 1.

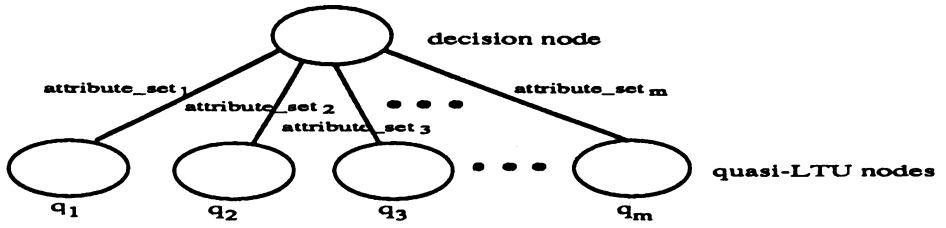
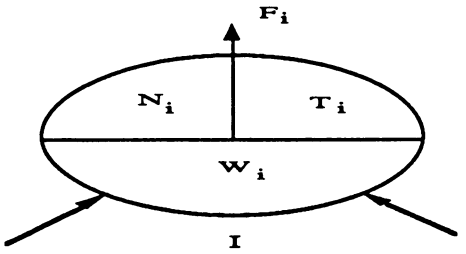


Figure 1. The quasi-LTU-decision model

The quasi-LTU node, described in figure 2, is a modified linear threshold unit. Each node has a set I of n input lines and an output F_i. The weight vector W_i and threshold T_i represent some hyperplane $P=\{I \mid I \in R^n$ and $W_i^t \cdot I = T_i\}$ that linearly separates the node's training instances. The

vector N_i is used to indicate whether the values of the corresponding elements of weight vector W_i are "don't care" or not.



$$F_i = \begin{cases} 1 \text{ if } W_i^t \cdot I > T_i \\ 0 \text{ otherwise} \end{cases}$$

Figure 2. The quasi-LTU node q_i

The operation of the quasi-LTU-decision model is illustrated as follows. When an input vector I is presented to the decision node, all the quasi-LTU nodes simultaneously test their corresponding attribute sets. The node which passes the attribute test is the only one to activate. Since these attributes are mutual-exclusion and the union of these attribute sets covers all the instances space, the node which activates is unique. If the node which passes the attribute test is q_i, then the following operation occurs in q_i.

if $W_i \cdot I > T_i$ ---> the output is 1 (means the input I belongs to Class-1),
if $W_i \cdot I \leq T_i$ ---> the output is 0 (means the input I belongs to Class-0).

2.2 The Training Process

Here, we first describe the non-incremental learning and leave the incremental learning in section 2.2.4. During the training phase, a collection of classified instances indicating a desired class is presented. The set of instances, called training instances, consists of typical examples of in-class instances as well as out-of-class instances. It is not necessary for the training instances to be linearly separable. The task of the training process is to generate the quasi-LTU-decision model which can be used to separate the in-class examples from those out-of-class examples and to predict the classification of the unobserved instances.

The strategy used in the generation of the quasi-LTU-decision model is described as follows. Split the training instances first to many subsets, each of the subsets belongs to one quasi-LTU node. It is guaranteed that each of the subsets can be linearly separated after the splitting. We then use the quasi-LTU learning algorithm described later to compute each component of the quasi-LTU nodes, based on the training instances subsets owned by their corresponding quasi-LTU nodes. Finally, apply some guidances to merge these quasi-LTU nodes as much as possible.

2.2.1 The Split Operation

The initial quasi-LTU-decision model consists of a decision node and a quasi-LTU node with the whole set of training instances. These two nodes are connected by a "true" attribute. If the number of the training instances owned by the quasi-LTU node is greater than the attribute number (the dimensionality) of the training instance vector + 1, then it's time to split the quasi-LTU node, i.e. to split the instances set of the quasi-LTU node into two subsets. While the split occurs, we divide the original quasi-LTU node into two quasi-LTU nodes, each node consists of one subset of the original training instances set. This splitting process then repeatedly applies to the two quasi-LTU nodes independently until the instances number owned by every quasi-LTU node is less than or equal to the attribute number of the instance vector + 1. Here, we use the attribute number of the instance vector + 1 as the guidance for the split of the quasi-LTU nodes. This is because if the number of the training instances is less than or equal to the attribute number of the instance vector + 1, then we can guarantee this training instances set is linearly separable[2].

The reason to split every quasi-LTU node to be linearly separable is as follows. Due to training the linearly separable instances set, it is not necessary to use those complex learning methods , e.g. back propagation learning algorithm, that are time consuming. we can just use the algorithm similar to the LTU learning algorithm to distinguish the linearly separable instances set. Besides, the splitting can be seen as the modularization of the original instances set since the whole instances set is modularized to many instances subsets. Furthermore, the less instances number in the instances set is, the more quickly the LTU learning algorithm will converge.

At this point, it seems important to illustrate how to split the quasi-LTU node. Since we use the instances number as the guidance in the split, it is not necessary to use the complex entropy functions[4,7,8,9] to split the quasi-LTU node. In our approach, what important is to split the

instances set of one quasi-LTU node into two subsets with the same instances number as much as possible. This is because the criterion can reduce the number of the splitting process. Therefore, the attribute to be chosen to split the instances set of the quasi-LTU node q_i is that which minimizes $|NF_{ik} - 1/2 \text{ instance\_no}_i|$. Where instance_no_i is the instances number belonging to the quasi-LTU node q_i and NF_{ik} indicates the instances number in the quasi-LTU node q_i that the kth attribute of these instances equals one.

2.2.2 The Learning Algorithm of the Quasi-LTU Nodes

After splitting the quasi-LTU nodes to be linearly separated, it's time to build a hyperplane for every quasi-LTU node to discriminate different classes. The learning algorithm of the quasi-LTU node q_i is to generate the weight vector W_i and threshold T_i representing some hyperplane $P=\{I \mid I \in R^n \text{ and } W_i^t \cdot I = T_i\}$ that can linearly separate the instances belonging to the node q_i. Besides, this learning algorithm also generates a "don't care" vector N_i for node q_i to indicate whether the corresponding elements of weight vector W_i are "don't care", which means the weights can be any value, or not. Initially, the whole vector Ni is given to be one vector, i.e. every weight can be any value. The value will be kept if there is no change in the weight and the corresponding feature of instance I is zero. Since the spirit of this algorithm is the same as Perceptron Learning Algorithm[3,5], we can guarantee that if the instances trained are linearly separable, then this algorithm will converge and define a hyperplane to the trained instances set.

2.2.3 Merge

The last step in our approach to build the quasi-LTU-decision model is to merge these split quasi-LTU nodes as much as possible. To merge two quasi-LTU nodes means to combine the two instances sets of two quasi-LTU nodes into one instances set that can be discriminated by one weight vector and a threshold. The restrict in merging is we must still keep these quasi-LTU nodes formed after merging quasi-LTU nodes being linearly separable.

To clarify what follows, we first define some variables and terms:

Definition 1: q_i means the quasi-LTU node q_i, N_i is the "don't care" vector of q_i, W_i is the weight vector of q_i, W_{ik} indicates the kth element of W_i, and N_{ik} is the kth element of N_i.

Definition 2: The term W_{ik} "equals" W_{jk} means $W_{ik} = W_{jk}$ or at least one of N_{ik} and N_{jk} is one ("don't care").

Definition 3: W_i "equals" W_j means all elements of W_i "equals" the corresponding elements of W_j.

Definition 4: I_i indicates the instances set in q_i, and I_{ik} is an instance numbered k in I_i.

Definition 5: I_i^+ is defined as these instances in q_i and belonging to Class-1, and I_i^- is defined as these instances in q_i and belonging to Class-0.

Definition 6: The modified weight vector W_{ij}' of W_i and W_j is defined as: $W_{ijk}' = W_{ik}$ if $N_{ik} = 0$; $W_{ijk}' = W_{jk}$ if $N_{ik} = 1$, where W_{ijk}' is the kth element of W_{ij}'.

Definition 7: The modified "don't care" vector N_{ij}' of N_i and N_j is defined as: $N_{ijk}' = 1$ if $N_{ik} = 1$ and $N_{jk} = 1$; $N_{ijk}' = 0$ otherwise, where N_{ijk}' is the kth element of N_{ij}'.

Theorem 1: If W_i "equals" W_j and $T_i = T_j$, then the combination of the instances sets in q_i and q_j is still linearly separable and there exists a modified weight vector W_{ij}' and threshold T_i forming a hyperplane to separate the combined instances set.

Theorem 2: If W_i "equals" W_j, $T_i > T_j$, and $\forall I_{jk} \in I_j^+$, $W_{ij}' \cdot I_{jk} > T_i$, then the W_{ij}' and T_i can be used to separate the combined instances set of I_i and I_j.

Corollary 1: If W_i "equals" W_j, $T_i > T_j$, and $\forall I_{ik} \in I_i^-$, $W_{ji}' \cdot I_{ik} \le T_j$, then the W_{ji}' and T_j can be used to separate the combined instances set of I_i and I_j.

Corollary 2: If W_i "equals" W_j, $T_i < T_j$, and $\forall I_{ik} \in I_i^+$, $W_{ji}' \cdot I_{ik} > T_j$, then the W_{ji}' and T_j can be used to separate the combined instances set of I_i and I_j.

Corollary 3: If W_i "equals" W_j, $T_i < T_j$, and $\forall I_{jk} \in I_j^-$, $W_{ij}' \cdot I_{jk} \le T_i$, then the W_{ij}' and T_i can be used to separate the combined instances set of I_i and I_j.

Theorem 3: If $W_i \cdot [\pm] I_{jk} > [\pm] T_i$, $\forall I_{jk} \in I_j$, then W_i and T_i can be used to separate the combined instances set of I_i and I_j. Where "+" is used in front of I_{jk} and T_i when $I_{jk} \in I_j^+$; otherwise, put "–" in front of I_{jk} and T_i.

Corollary 4: If $W_j \cdot [\pm] I_{ik} > [\pm] T_j$, $\forall I_{ik} \in I_i$, then W_j and T_j can be used to separate the combined instances set of I_i and I_j. Where "+" is used in front of I_{ik} and T_j when $I_{ik} \in I_i^+$; otherwise, put "–" in front of I_{ik} and T_j.

The three theorems and four corollaries can be easily proved and are used as criteria to merge these split quasi-LTU nodes. That is, if the conditions in these theorems and corollaries are

satisfied, then we can merge two quasi-LTU nodes into one that owns all the instances in the two merged quasi-LTU nodes. As the mergence occurs, the "don't care" vector of the new formed quasi-LTU node merged from q_i and q_j is N_{ij}' while the weight vector and threshold are indicated in these theorems and corollaries. The mergence continues as long as the conditions of these theorems and corollaries are satisfied. From these theorems and corollaries, we can make sure that the new formed quasi-LTU nodes from merging are still linearly separable by the given weights and thresholds. It is obvious that the criteria used here can not guarantee to merge any two quasi-LTU nodes that are still linearly separable after merging. It is not harmful, however, since the ability to find a consistent concept representation is not lost. Moreover, the number of the quasi-LTU nodes of the quasi-LTU-decision model is still far less than the nodes number in the decision tree and the training time of the model is much less than the time needed by other neural network learning algorithms.

2.2.4 Incremental Learning

It is not necessary to collect a large set of instances before building the quasi-LTU-decision model. We can also build the model incrementally, i.e. we can accommodate the structure of the model every time when we receive a new instance. The incremental learning algorithm used here is similar to the processing of classifying an unknown instance in the built quasi-LTU-decision model. When a new instance is presented to the built quasi-LTU-decision model, the appropriate quasi-LTU node will activate to classify this input instance. If the classification is correct, then just put this instance to the activating quasi-LTU node and then stop. Otherwise, repeatedly split this activating quasi-LTU node into several linearly separable quasi-LTU nodes and then merge these split nodes themselves as much as possible. After these, merge these new created and modified quasi-LTU nodes into the other quasi-LTU nodes in the quasi-LTU-decision model. The operations of the split and mergence discussed here are the same as described above.

3. Conclusion

There are a number of issues which need to be further investigated. Some of them are:

(1) Handle the training instances with noises. Since it is inevitable that some of the trained examples are not correct, the learning algorithms which can handle the noisy training instances are necessary and useful.
(2) To find fast and efficient ways to merge the quasi-LTU nodes which are still separable after merging.
(3) Build a connectionist expert system based on the representation.
(4) To extend the output to multiple values. That is, to extend the binary classes classification to multiple classes classification.

References

[1] B.Chandrasekaran and A.Goel," From Numbers to Symbols to Knowledge Structures: Artificial Intelligence Perspectives on the Classification Task," *IEEE Trans. on System, Man, and Cybernetics*, Vol. 18, No. 3, pp 415-424, May/June 1988.

[2] R.O.Duda and P.E.Hart, *Pattern Classification and Scene Analysis*, John Wiley & Sons, New York, pp 69-70, 1973.

[3] S.I.Gallant," Optimal Linear Discriminants," *IEEE Proc. of Intl. Conf. on Pattern Recognition*, pp 849-852, 1986.

[4] J.Mingers," An Empirical Comparison of Selection Measures for Decision-Tree Induction," *Machine Learning*, Vol. 3, pp 319-342, 1989.

[5] M.Minsky and S.Papert, *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, Ma., 1969.

[6] F.Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Spartan Press, Washington, D.C., 1961.

[7] R.W.Selby and A.A.Porter," Learning from Examples: Generation and Evaluation of Decision Trees for Software Resource Analysis," *IEEE Trans. on Software Engineering*, Vol. 14, No. 12, pp 1743-1757, Dec. 1988.

[8] G.Z.Sun, H.H.Chen, Y.C.Lee," Parallel Sequential Induction Network - A New Paradigm of Neural Network Architecture," *Proc. of IEEE International Conference on Neural Networks*, pp I-489-496, 1988.

[9] P.E.Utgoff," Perceptron Trees: A Case Study in Hybrid Concept Representations," pp 601-606, *AAAI-88*.

APPLICATION OF COULOMB ENERGY NETWORK TO KOREAN CHARACTER RECOGNITION

*Kyunghee Lee** *and Won Don Lee***

* Electronics and Telecommunications Research Institute
Daejeon, Chungnam, KOREA

** Department of Computer Science College of Natural Sciences
ChungNam National University, Daejeon, Chungnam, KOREA

ABSTRACT

Coulomb energy network originally developed by Scofield is applied to the recognition of the korean characters. In his paper, Scofield demonstrated that his learning algorithm was applied to a system which learns the binary mapping for XOR. The present work describes the result of implementing multi-layer coulomb energy network for the korean character recognition. This neural network system was experimented on VAX 8800 mini computer. korean characters can be classified into 6 types. We tried to make a system to recognize characters in a subset of one type. We trained the system with patterns varying in size, rotation, and with distortion. Multi-layer coulomb network was able to classfy the korean characters and identify them. With korean characters, it is difficult to achieve proper classification using 1 stage training only, because two different korean characters can be very similar in shape. When confusion occured, 2-stage learning is necessary. Learning patterns which would not make a confusion were shown at first, and other learning patterns which might make a confusion were shown next in training sequence. In this manner, the multi-layer coulomb energy network was trained to recognize the korean characters quickly with only quite a few learning patterns(20 patterns/consonant or vowel).

INTRODUCTION

Recently, Bachmann et al. proposed a relaxation model[1] based on an N-dimensional coulomb potential. Their system is similar to the Hopfield model[2][3] in some respects, but has arbitrary large capacity. Scofield presented a learning algorithm[4] for N-dimensional coulomb energy network which is applicable to a single as well as a miltilayer networks. While many learning algorithms to recognize characters already exist, Scofield's learning algorithm is worthwhile to note, because it is applicable independently to each layer of a multi-layer network and does not depend on the propagation of errors through many layers. This property makes the system modular. We have tried to make a neural network for the korean character recognition using this multi-layer coulomb network. The korean characters can be classified into six types. They are 'ka'type, 'kak'type, 'ko'type, 'kok'type, 'kwa'type and 'kwak'type. The problem is that there are many similar chars in shape although they are different charaters. For example, 'ㅏ', 'ㅣ', 'ㅓ' are very similar to each other. We have used 'kak' type koreans to train the networks. We also present an idea to save time and space in training the multi-layer coulomb network(2-stage learning) for Korean recognition. And we show the result of the experiment to classify the korean chars using the system.

MODEL DESCRIPTION

The network model proposed here was composed of three types of layers: the input layer, the internal layers and the output layer. The input layer receives the binary values of an input pattern. The internal and the output layers consist of coulomb potential function units. The output values of the internal layer units are summed by the connection between the internal layer and the output layer to produce the output. The schematic diagram of a multi-layer coulomb energy network for the korean characters is shown in fig.1

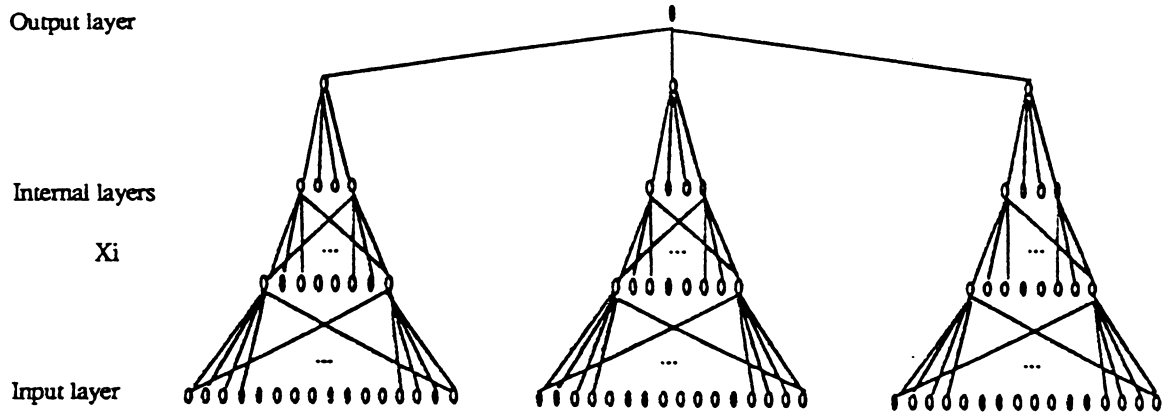


Fig 1 : Multi-layer coulomb energy network model for the korean characters

The korean characters can be classified into six different types, and each of them are composed of consonant(s) and a vowel. In this paper we used the 'kak'type korean characters composed of a first consonant, a vowel and a second consonant. To recognize this type, we constructed 3 neural network modules, one for vowel and two for the first and the second consonants. The korean characters stored in a matrix 8x8 pixel were used as input patterns. Each consonant and vowel is composed of 4x4 matrix. The number of neurons in input layer, internal layers and output layer for each module are 16, 8, 4, and 1 respectively. Each consonant and vowel area where each network module is looking at in the korean character input pattern along with some examples are shown in fig 2.

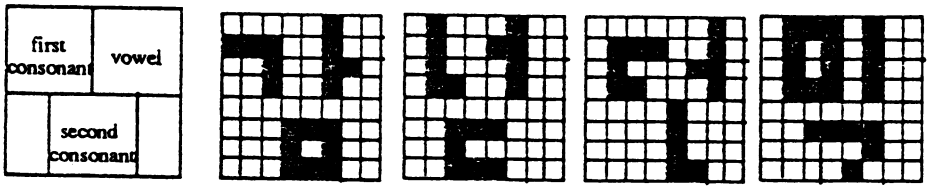


Fig 2 : Consonant and vowel area in the korean character

LEARNING

Learning[4] in a coulomb energy network is to adjust the matrix ω shch that the electrostatic energy, Ψ, of the collection of charges is minimized. Electrostatic potential edergy of the M memory sites $x_1,\ldots, x_M$ in R^N used in coulomb energy network is :

$$\Psi = 1 / (2L) \sum_{i=1}^{M} \sum_{j=1}^{M} Q_i Q_j |X_i - X_j|^{-L} \qquad (1)$$

Where Qi is the charge at memory site i, Xi the representation of the i-th memory site.

Qi(c), the charge at memory site i and of class c is defined as follows:

$$sign(Q_i(c)) \neq sign(Q_i(c)) \text{ for } c = c$$
$$sign(Q_i(c)) = sign(Q_i(c)) \text{ for } c \neq c \qquad (2)$$

And to minimize the electrostatic potential energy, gradient of potential energy was computed with respected to weight ω_{nm}(m-th synapse of the n-th cell).

$$\frac{\partial \Psi}{\partial \omega_{nm}} = -1/2 \sum_{i=1}^{M} \sum_{j=1}^{M} Q_i Q_j |\mathbf{R}_{ij}|^{-(L+2)} \mathbf{R}_{ij} \partial/\partial \omega_{nm} \mathbf{R}_{ij} \qquad (3)$$

Where :

$$\mathbf{R}_{ij} = \mathbf{X}_i - \mathbf{X}_j$$

But, equation (3) may be approximated through the successive computation of these terms over time as follows:

$$\delta \omega_{nm} = (+/-)\ \eta\ |\mathbf{X}(t) - \mathbf{X}(t+1)|^{-(L+2)} \Delta_{nm}(\mathbf{f}(t), \mathbf{f}(t+1))$$

Where eta is the learning rate, negative sign is for subsequent patterns of the same class and positive sign for patterns of different class.

The 'kak'type out of 6 types was used in this experiment. Each consonant and vowel module was trained with 20 patterns(10 patterns were of same class while the rest were of different class). Learning patterns in pairs were selected randomly 38000 times out of 20 patterns and were shown to train each consonant(vowel) module. The learning rate was chosen carefully and in our experiment, it was <0.0005. Here L was 2, and the temperature was set as 0.8 and values between -0.3 and 0.3 were assigned randomly as initial weights.

EXPERIMENTAL RESULTS

We made a network to classify ' ㄱ ', ' ㄴ ', ' ㄷ ', ' ㅁ ', ' ㅏ ', ' ㅣ ', ' ㅓ ', as our target consonants and vowels. At first, since there were only 10 learning patterns for each consonant(vowel), and also there were a few internal neurons, the network was not able to classify ' ㅏ ',' ㅣ ',' ㅓ ' completely. To overcome this, input patterns for learning were made up of 2 classes. Patterns that should be recognized as a same character were put into class 1 and others recognized as different patterns into class 2. There may be some confusion if patterns of class 1 are similar to those of class 2. When confusion occured 2-stage learning was applied. Learning patterns which would not make a confusion were shown as input at first, and other learning patterns which might make a confusion were shown next. By doing that, the module could extract the major features of each consonant(vowel) in the first phase, and minute differences in the second phase. This makes it possible to learn quickly and clearly with a few input patterns in a little learning time. Some example patterns used here is shown in fig 3. As is shown, patterns belonging to the same class may have quite different bit vector representations in the input space as they have different shape, size, and distortion.

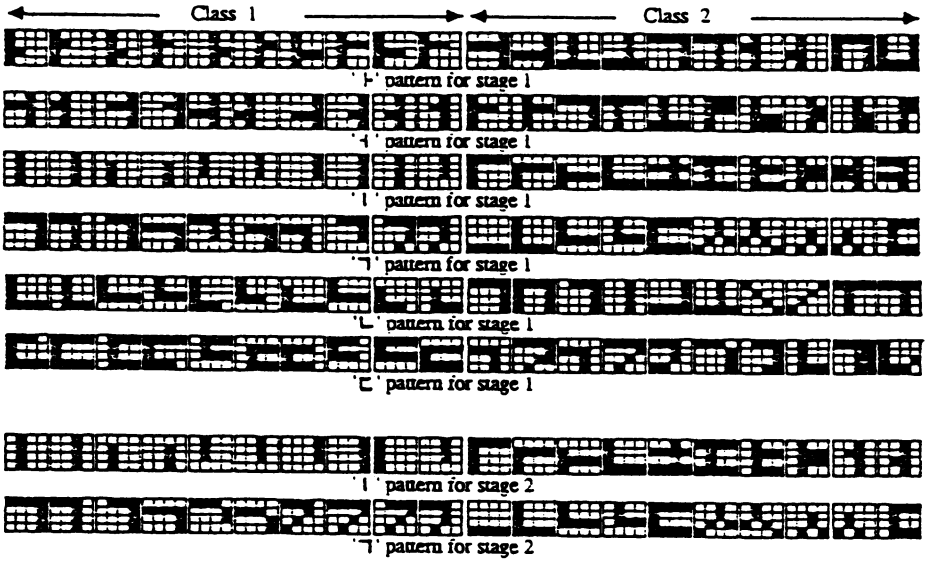


Fig 3 : Example patterns used in the experiment

The summary of experimental result for the recognition of 100 random 'kak' type korean characters are shown in table 1. The test characters were with random noise, varying size, shape, and shift in position. As we can see, after first stage learning, only 65 percents of the test characters are recognized correctly while after the second phase, it increased substancially to 97 percents.

item \ stage no.	stage 1	stage 2
learning count	38,000	19,000
recognition ratio	65%	97%
confusion ratio	32	2
error ratio	3	1

Table 1 : Summary of experimental result

CONCLUSION

Coulomb energy network has been adopted to classify korean characters and performance was evaluated. This study found that 2-stage learning is effective in training the module. Clustering occurs naturally in this model. Without much learning time and computing, we have achieved a good result only with 20 patterns in training each consonant(vowel) and about 90 minutes CPU time for learning per consonant(vowel).

REFERENCES

[1] Bachmann, C.M., Cooper, L.N., Dembo,A.,Zeitouni,O.: A relaxation model for memory with high density storage. Proc. Natl. Acad. Sci. USA 21, 7529 - 7531(November, 1987)

[2] Hopfield, J.J. : Neural networks and physical systems with emergent collective computational abilities. Proc. Natl. Acad. Sci. USA 79, 2554 - 2558 (April, 1982)

[3] Hopfield, J.J. : Neurons with graded response have collective computational properities like those of two-state neurons. Proc. Natl. Acad. Sci. USA 81, 2088 - 3092(May, 1984)

[4] Scofield, C.L. : Learning internal representations in the coulomb energy network. ICNN, Vol I, 271 - 275(July, 1988)

ART 1.5 -- A SIMPLIFIED ADAPTIVE RESONANCE NETWORK FOR CLASSIFYING LOW-DIMENSIONAL ANALOG DATA

Daniel S. Levine
Department of Mathematics
University of Texas at Arlington
Arlington, TX 76019-9408

P. Andrew Penz
Texas Instruments, Inc.
P. O. Box 655936, MS 134
Dallas, TX 75265

The radar clustering problem

A region of the microwave spectrum can contain many radar signals from different emitters. It is assumed that the microwave receivers used can measure azimuth, elevation, frequency, pulse width, and time of arrival of individual radar pulses, with some degree of noise. For defense applications, it is important to be able to tell quickly (1) how many emitters are present; (2) what are the properties of these emitters; and (3) do they match any classes of emitters that have previously been seen and are known to have certain properties in common? We have designed a neural network that can perform all three of these tasks on simulated radar pulse inputs which are perturbations of characteristic pulses from 10 simulated emitters. The main part of our article concerns figuring out how many categories (emitters) the pulses fall into. This is an unsupervised learning task: categorizations are initially unknown. Previously (Anderson et al, 1988), this problem had been studied with the brain-state-in-a-box (BSB) algorithm. We applied to it a different unsupervised learning algorithm, the adaptive resonance theory (ART).

The two main types of ART network are ART 1 for processing binary patterns (Carpenter and Grossberg, 1987a) and ART 2 for processing analog patterns (Carpenter and Grossberg, 1987b). Figure 1 shows generic ART architecture covering both subcases. Since ART 1 is, in general, simpler than ART 2, we first tried to represent the radar data in a binary format. But since each variable has a range of possible values, reasonably accurate binary representations were found to require 40 or more feature nodes. ART is based on a system of differential equations including feature node activities, category node activities (at least 10, since there were 10 emitters), feature-to-category-node (bottom-up) connection weights, and category-to-feature-node (top-down) connection weights. Hence, 40 feature nodes would mean at least 850 equations.

We thus decided to use a simplified version of ART 2 for radar categorization. Five radar variables were used -- azimuth, elevation, signal-to-noise ratio, frequency, and pulse width. (Time of arrival was added later, at the stage when derived categories were compared with known emitters.) The problem thus became one of clustering five-dimensional vectors. Each variable was normalized to be between 0 and 1. For example, the minimum observed pulse

frequency was 8640 while the maximum frequency was 10100. Hence, a frequency of 9339 was converted to a normalized value of (9339 - 8640)/(10100 - 8640)=.476.

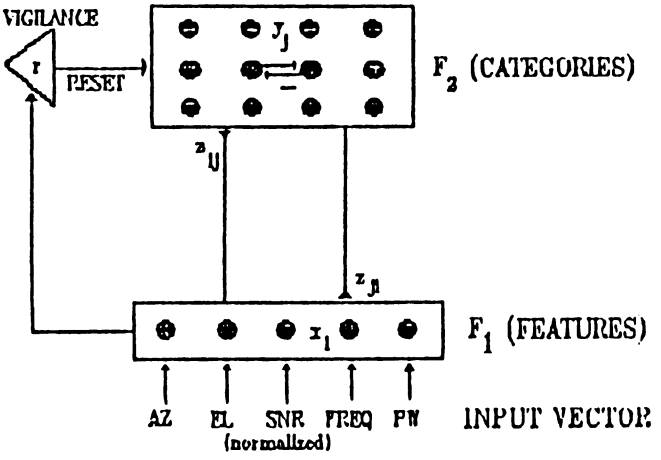


FIGURE 1

For such low-dimensional data, in fact, much of the complexity originally built into ART 2 was unnecessary. The analog data classified by Carpenter and Grossberg (1987b) were curves described by 20 or more points. Hence, classifying these curves required several preprocessing stages at the feature (F_1) level of the network, for suppressing noise in the data. Such preprocessing was not needed for our task. Moreover, ART 2 actually uses simpler learning laws than ART 1. In ART 1, top-down and bottom-up connection weights obey different equations, whereas in ART 2 (and ART 1.5), both sets of weights obey the same equations. (Top-down and bottom-up weights do have different initial values, for reasons to be given below.)

Single versus dual vigilance measures

Our categorization algorithm is as follows. Each normalized radar vector is presented as an input to the five F_1 nodes. The the F_1 node activities are multiplied by the corresponding F_1-to-F_2 bottom-up connection weights, and each F_2 node thereby receives a linearly weighted signal from the F_1 field. The input is first tentatively placed in the category corresponding to the node receiving the largest signal. (This "winner-take-all" mechanism is assumed to arise, in biological neural networks, from recurrent competitive interactions among F_2 nodes. In our simulations, such competition is not explicitly included.)

When an F_2 node is tentatively chosen, the input vector is compared with the vector of top-down connection weights from that node to F_1 nodes (the category prototype). As in both Carpenter-Grossberg articles, a parameter called vigilance measures whether the match is close enough to accept the input into that category. ART 1 and ART 2 use different matching criteria; ours, which is similar to the ART 2 criterion, is an angle-cosine measure. Specifically, if $\vec{x} = (x_1, x_2, x_3, x_4, x_5)$ is the vector of F_1 activities, in response to the normalized inputs, and $\vec{z}_j \equiv$

$(z_{j1}, z_{j2}, z_{j3}, z_{j4}, z_{j5})$ is the vector of top-down weights from the j^{th} (chosen) node, match is said to occur when

$$(\vec{x} \cdot \vec{z}_j)/(\|\vec{x}\| \|\vec{z}_j\|) > r \qquad (1)$$

where r (the vigilance) is some number between 0 and 1.

If (1) fails, the algorithm finds a new F_2 node receiving the next highest F_1 signal and tests (1) again on the corresponding j. In the Carpenter-Grossberg networks, if at least one F_2 node is uncommitted (that is, has all $z_{ij}=0$ because no inputs have yet been placed in that category), an input which fails (1) for all committed nodes is placed in the category determined by some uncommitted node. If (1) is true for some j ("resonance"), the input is placed in that category, and synaptic weights (both top-down and bottom-up) are changed to reflect that input. The top-down weight changes, in our version, obey the equation

$$dz_{ji}/dt = af(j)\,(-(1-a)\,z_{ji} + x_i) \qquad (2)$$

where a is a constant between 0 and 1, and f(j) is 1 if the input resonates with node j and 0 otherwise. The bottom-up weights z_{ij} obey an analogous equation.

To process our noisy data, we varied Carpenter and Grossberg's procedure to include two vigilance values -- r_{max} for "certain match" and r_{min} for "possible match". The reason for dual vigilance is that the clustering of emitter parameter vectors in five-dimensional space is not uniform, so no single vigilance would yield accurate cluster sizes. Too high a vigilance, it was found, led to mismatch of some pulses to their correct categories, whereas too low a vigilance led to other pulses being incorrectly placed together. Moreover, since connection weights were updated with each categorization, these miscategorizations tended to perseverate to later pulses.

To solve this dilemma, we imposed a rule whereby if (1) holds with $r=r_{max}$, a pulse is placed in a category and weights updated, as before. If (1) with $r=r_{min}$ fails for all committed categories, the pulse is placed in an uncommitted category and weights updated, as before. In the intermediate case where the pulse matches no category to within r_{max} but at least one category to within r_{min}, the best match is chosen but weights are not changed. Thus category boundaries will not be shifted by inputs that are close to those boundaries. (Levine, 1989 outlines another use for dual vigilance, the detection of ambiguity).

Initial values of all top-down weights z_{ji} are 0, i. e. no nodes are committed. The bottom-up weights z_{ij} could not be started at 0, else F_2 would receive no F_1 signals. Hence z_{ij} are initialized at low uniform values.

Results and comparison to known emitters

The ART 1.5 network was simulated on a sequence of 294 pulses, called the IOTA buffer, with vigilances $r_{max}=.995$ and $r_{min}=.985$. This network had 5 F_1 nodes and 12 F_2 nodes. The category placement of pulses known to be from given emitters is shown in the matrix of Figure 2. The diagonal entries correspond to correct categorizations. Hence, 290 out of 294, or 98.7% of the pulses, were categorized correctly. The 4 pulses that were miscategorized all had anomalously large pulse widths, far

larger than the expected widths of pulses from any of the 10 known emitters. This occurred because the radar receiver adds widths of pulses from two different emitters if they arrive nearly simultaneously. In hardware, it is easy to add a mechanism for flagging unusually wide pulses.

		CATEGORY #													
		12	11	10	9	8	7	6	5	4	3	2	1	0	none
E	2	140	0	0	0	0	0	0	0	0	0	0	1	0	0
M	7	0	17	0	0	0	0	0	0	0	0	2	0	0	0
I	4	0	0	14	0	0	0	0	0	0	0	0	0	0	0
T	10	0	0	0	15	0	0	0	0	0	0	0	0	0	0
T	5	0	0	0	0	13	0	0	0	0	0	0	0	0	0
E	6	0	0	0	0	0	44	0	0	0	0	0	0	0	1
R	3	0	0	0	0	0	0	14	0	0	0	0	0	0	0
	8	0	0	0	0	0	0	0	16	0	0	0	0	0	0
#	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	9	0	0	0	0	0	0	0	0	0	3	0	0	0	0

FIGURE 2

After training on IOTA, the ART 1.5 network was run with the same vigilances on another pulse buffer known as "2X", based on the same emitter norms as IOTA but with twice as much noise. Out of 286 pulses in 2X, 277, or 96.8%, were categorized correctly. Of the 9 errors, 3 were due to anomalous pulse width and 6 to excessive noise.

In the classification stage of our algorithm, average frequency (FREQ) and average pulse width (PW) were computed for the pulses in each category with three or more members. A pulse repetition index (PRI) was computed for each category by averaging the differences between successive arrival times of the category's pulses. Then these three variables were matched with those of the originally postulated emitters, by using (1) with $\vec{x}$ and $\vec{z}_j$ replaced by the two three-component (FREQ, PW, PRI) vectors and $r=r_{max}$. The category-to-emitter matches obtained were all correct with one exception. The same two categories matched both Emitters 3 and 9, because those two emitters happened to have (FREQ, PW, PRI) vectors that were nearly proportional, though far from equal. In hardware, this anomaly can be solved by replacing the angle-cosine match measure, at this stage, with an absolute vector-distance measure.

References

Anderson, J. A., Penz, P. A., Gately, M. T. & Collins, D. (1988). Neural Networks 1, Suppl. 1, 422.

Carpenter, G. A. & S. Grossberg (1987a). Computer Vision, Graphics, & Image Processing 37: 54-115.

Carpenter, G. A. & S. Grossberg (1987b). Applied Optics 26: 4919-4930.

Levine, D. S. (1989). In W. Webster (Ed.), Simulation and AI -- 1989. San Diego: Society for Computer Simulation, pp. 1-7.

Retro: An Expert System Which Embodies "Chemical Intuition"

by
Hudson H. Luce and Rakesh Govind
Department of Chemical Engineering (ML-171)
University of Cincinnati
Cincinnati, OH 45221

I. Introduction:

One of the tasks of synthetic organic chemistry is to construct complex molecules from relatively simple starting materials and reagents. The principal analytical tool used for planning syntheses is known as retrosynthetic analysis, a process in which the complex target molecule is decomposed into a set of fragments [69Corey]. Each decomposition step represents the "undoing" of a particular chemical reaction. This process of fragmentation is iterated until each member of the set of fragments produced corresponds to a starting material or reagent. A "synthetic tree" is hereby produced, of which the "leaves" correspond to starting materials and reagents, the "branches" give the synthetic pathways, and the "root" is the target molecule.

II. Cognitive Tasks in Retrosynthetic Analysis:

There exists a given set of rules by which these retrosynthetic transformations may be accomplished. A novice chemist will recognize a feature in a molecule (such as a strategically placed double bond, or perhaps a juxtaposition of two different substituent groups), and apply one of the aforementioned rules in a mechanical fashion to obtain the corresponding set of fragments ("synthons") for the respective target molecule. By repeating this process, a synthetic tree will be produced. In contrast, a more mature chemist will employ "chemical intuition" to choose a retrosynthetic transformation; in other words, the chemist will discern a pattern of relevant features which form a molecular "gestalt", and choose the appropriate disconnection to produce a set of fragmentary synthons. The difference in the cognitive processes used by the novice and the expert are analogous to the difference between a novice chess player and a grandmaster level player: the novice inspects the moves each piece can legally make and considers the "pros and cons" for the piece under consideration, and formulates the strategy for the current move accordingly. On the other hand, a grand. master looks for patterns of chess pieces, which in themselves form a series of "gestalts" which are altered as the game progresses. From the current pattern, the master chooses a move which will create a winning pattern, or a pattern

intermediate to a winning pattern; analogously, the mature chemist discerns patterns to create "elegant" syntheses.

III. Models of Retrosynthetic Analysis:

The current computational models [84Wipke] for retrosynthetic analysis use the strategy of a novice chemist, in which individual features are recognized. Rules are then applied to give possible disconnections. By this method, many "synthetic trees" are generated; each must be analyzed with regard to cost and yield. In contrast, our approach uses neural networks [84Hopfield], [86Hopfield], [86Rumelhart] to encode the sets of rules which correspond to the set of retrosynthetic transformations. This approach also examines the entire set of possible disconnection points in a molecule in a parallel manner, making it possible to discern patterns of features in the target molecule. The intuitive leaps which characterize elegant and creative syntheses are dependent on the ability of a chemist to discern these patterns of features; our approach thus embodies "chemical intuition".

IV. Design of an Expert System with "Chemical Intuition":

1. Input and Generation of Internal Representations.

The target molecule is input via a menu-driven graphic input package, which produces a 2-dimensional picture. Choosing a graphic primitive, such as an atom, functional group, or ring, causes its internal representation atom or list to be appended to the target molecule list. When finished, the target molecule list contains all lists of primitives in the molecule. This list is used to construct an adjacency matrix.

2. Parsing Lists for Feature Detection

The target molecule list is scanned so that synthetically important features can be detected. The scanning "window" is seven skeletal atoms long, and the adjacency matrix is used in order to detect features that might not necessarily be attached to the skeletal atom chain currently being examined. This scanning is done for each skeletal atom, in parallel. A set of feature vectors is thus composed, one for each skeletal atom.

3. Selection of Disconnection by Competition

The feature vectors are presented to a set of neural networks, each of which is trained to propose a different type of

disconnection (Figure 1). Each network acts as an "agent" [86Minsky] for its own type of disconnection; competition between agents decides which disconnection is performed, and the loci for the disconnection.

4. Formation of Fragments; Starting Material Detection

Fragment lists are formed according to the disconnection chosen, and compared to lists in a database of starting materials. Iteration through Steps 2 , 3, and 4 continues until all fragments produced are found in the starting material database.

V. Summary:

The current methods of computational retrosynthetic analysis consider the molecule as a collection of discrete features to be analyzed in a sequential manner, thus limiting the process of creating a synthetic tree to a sequence of mechanical steps. Our approach, on the other hand, examines patterns of features, composes feature vectors, and uses neural networks to find "gestalts" that lead to creative disconnection steps, and thus to "elegant" synthetic routes.

VI. References:

[69Corey]: Corey, E.J., Wipke, T., *Science*, 166, 178, (1969).

[84Wipke]: Wipke, W.T., Rogers, D., *J. Chem. Inf. Comput. Sci.,* 24, 255, (1984).

[84Hopfield]: Hopfield, J.J., *Proc. Natl. Acad. Sci. USA,* 81, 3088, (1984).

[86Hopfield]: Hopfield, J.J., Tank, D.W., *Science,* 233, 625, (1986).

[86Rumelhart]: Rumelhart et al., *Parallel Distributed Processing Explorations in the Microstructure of Cognition,* Eds. Rumelhart, D.E., and McClelland, J.L., vol. 1, MIT Press, Cambridge, MA, (1986).

[86Minsky]: Minsky, M., *Society of Mind,* MIT Press, Cambridge, MA, (1986).

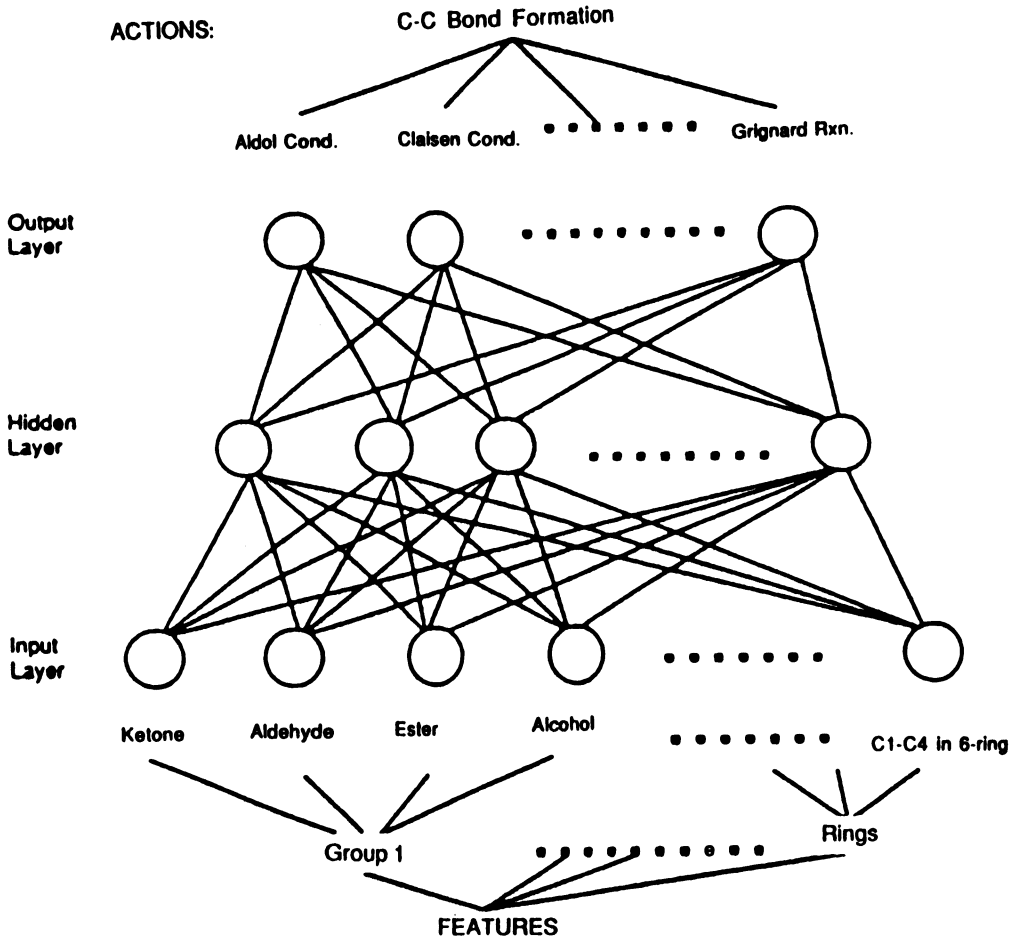


FIGURE 1. Schematic of the proposed Neural Network showing some examples of input features and one possible action.

NEURAL NETWORKS AND GENERAL PURPOSE SIMULATION THEORY

Gregory R. Madey
Jay Weinroth
Computer Science and Information Systems
Administrative Sciences
College of Business Administration
Kent State University
Kent, Ohio 44242
USA

Abstract

The use of a neural network embedded in a larger General Purpose Simulation System (GPSS) simulation used to evaluate alternate human resource policies in a factory setting is described. The neural network is used to model the optimal control of a machine tool by a human operator. We evaluate the feasibility of using GPSS for neural network modeling. The interface between general simulation theory and neural network simulation is examined. Neural networks, when embedded in larger general purpose simulations, are found to offer the potential for improving on the capabilities of those simulations.

Introduction

Most neural network software simulations are implemented in either third generation languages or in commercial software "neural network simulators." The authors' experience with the development of neural networks in both of those environments [Madey-88] and their experience with general purpose simulation packages raised the question as to whether a broader application of neural networks might occur if existing users of commercial general purpose simulation languages could also conveniently simulate neural networks. Thus the research described in this paper was motivated by two questions: 1) can general purpose simulation theory and languages, such as the General Purpose Simulation System (GPSS) [Schriber-74], be applied to neural network modeling, and 2) can neural network theory be used to improve on the state-of-the-art of simulation theory. We answer both of the questions in the affirmative, although not without some qualifications.

Several dozen commercial neural network simulation packages are available today, although the installed base is just starting to grow [PC AI-89]. On the other hand, general purpose simulation languages (GPSS, SLAM, SIMCRIPT, SIMAN, etc.) have been in use for almost two decades and currently have a large installed base with a corresponding large population of trained and experienced users. Thus, it would be of interest to those users if their experience in the general purpose simulation packages would permit their modeling of neural systems. Using GPSS we provide a "proof-in-principle" that this can be done.

Several deficiencies of current simulation theory limit the usefulness of that technique for problem solving. These deficiencies include 1) the inability to optimize, 2) long solution times, 3) unrealistic and simplified modeling of complex real world phenomena (including human behavior), and 4) the requirement for labor intensive development and analysis of simulations. We

demonstrate that neural systems may contribute techniques that address these traditional deficiencies of general simulation theory.

Problem

Ongoing research by one of the authors [Bruning and Weinroth-88] has indicated a need for a more robust method for simulating expert human behavior within a simulation of the impact of human resource management policies in a factory setting. A block diagram of the overall simulation is displayed in Figure 1. The operation of a machine tool — a representative factory work station — is simulated to the level of detail in Figure 2. Machine breakdowns and the variable output of good and defective products per unit of time are modeled as dependent on

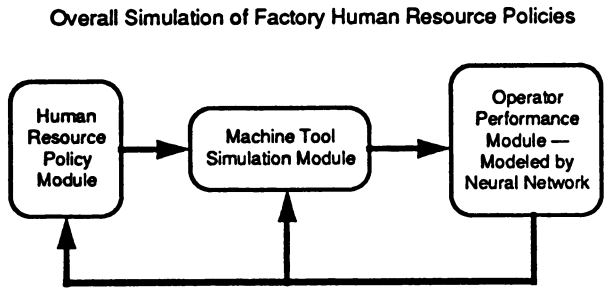


Figure 1: Block Diagram of the Overall GPSS Simulation

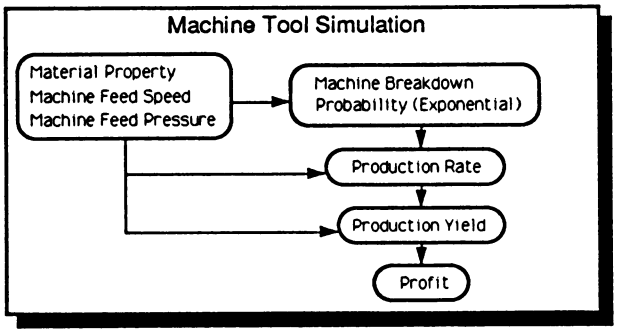


Figure 2: Model of the Machine Tool Workstation for Profit Evaluation

material property, and machine feed pressure and speed. Profit produced at the workstation is optimized by the machine operator (through two control parameters — machine speed and pressure) and is modeled by a 3-layer feedforward backpropagation network that is embedded in the simulation language. See Figure 3. Though the use of the embedded neural network, the relationships between profit and human resource policies are modeled more faithfully by the frequency and timeliness of the operator's corrective intervention to adjust the control parameters on the machine. The optimal control of the machine was taught to the neural net by presenting it examples of the proper response over the range of possible out-of-control conditions. No simple mathematical relationship — as might be derived from regression analysis — was found to perform with equal effectiveness in capturing the dynamics of the optimal control of the machine.

Approach

The overall simulation depicted in Figure 1 was developed using GPSS/H from Wolverine Software running on either an IBM mainframe or PC [Banks-89]. The neural network was trained both offline on the Macintosh using the neural network simulator MacBrain [Jensen-88]. Prototype experimentation was also performed in training the net within the GPSS/H simulation language. A standard backpropagation network with a three layer (4-8-2) architecture was employed. See Figure 4.

The network required approximately 5000 epochs of the entire training set (20 presentations) to reach acceptable performance (approximately +/-5% of the optimal control output).

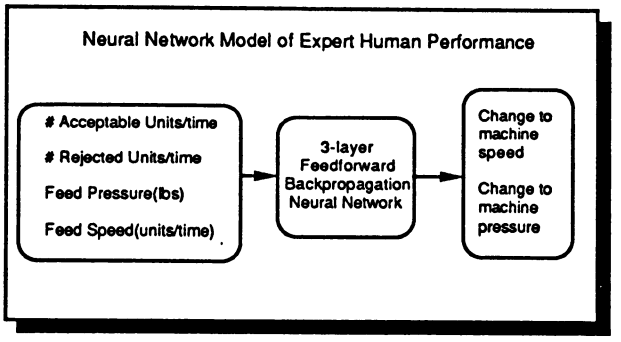


Figure 3: The Embedded Neural Network Modeling the Expert Judgement of the Human Operator at the Machine Tool Workstation

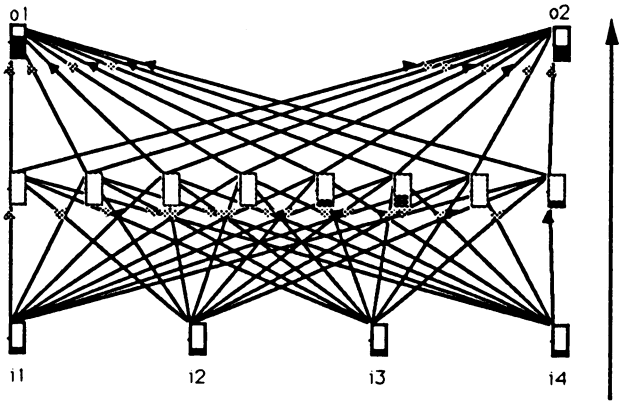


Figure 4: Three layer 4-8-2 Feedforward/Backpropagation Neural Network Used in Simulation

Conclusions

We claim that a "proof-in-principle" of the feasibility of using a commercial general purpose simulation language for neural network simulations is provided. We also demonstrate the potential utility of embedding neural networks within more general simulations (e.g., a discrete event factory simulation). Table 1 displays several examples of the results of the machine tool workstation simulation corrected by the embedded neural network. Parallel branches in the simulation, each using the same random numbers for computing failure times, failed condition and repair rates, were used to determine the effect of the embedded neural net's control output. This was needed to obtain an "apples-to-apples" comparison of the performance of the simulation with and without the input of the embedded neural network.

Other commercial general purpose simulation languages need to be evaluated and compared to GPSS/H for the ease and suitability of simulating neural networks. For example, the version of GPSS used for this research was limited to integer values for several network parameter values. This required extra steps to scale network parameters to and from large integer values. Most other commercial simulation packages do not have this limitation.

Perhaps neural network simulation modules, similar to templates for spreadsheets, can be developed for the various network architectures. Only discrete event simulation has been examined in this research. The continuous simulation languages may also need to be evaluated for their suitability for architectures such as the Hopfield net or Adaptive Resonance Theory.

Trial	Profit: Out-of-Control Condition	Profit: After Intervention by Neural Network
1	$5817.00	$6236.00
2	6730.00	7504.00
3	7246.00	8964.00
4	4621.00	4640.00
5	3834.00	5222.00

Table 1: Comparison of Machine Tool Profitability with and without Embedded Neural Network in Factory Simulation

References

1. Banks, J., Carson II, J.S., and J. N. Sy, ***Getting Started with GPSS/H***, Wolverine Software Corporation, Annandale, VA., 1989.

2. Bruning, N. S. and J. Weinroth, "Simulating the Consequences of Job Redesign," ***Computers and Personnel***, pp. 31-36, Spring 1988.

3. Jensen, M. and D. Chait, ***MacBrain 2.0 - Adaptive Simulation of Complex Systems***, Neuronics, Inc., Cambridge, MA., 1988.

4. Madey, G. and J. Denton, "Credit Evaluation Using Neural Networks," ***Proceedings of the INNS-88***, Boston, August 1988.

5. ***PC AI***, July/August 1989, p. 65.

6. Schriber, T. J., ***Simulation Using GPSS***, J. Wiley & Sons, New York, 1974.

OPTIMIZING THE HOUSEHOLD UTILITY FUNCTION USING NEURAL NETWORKS

Sergio Margarita
Istituto di Economia Politica "G.Prato"
Università di Torino - Facoltà di Economia e Commercio
Piazza Arbarello 8
10100 Torino - Italy

1. INTRODUCTION

Artificial neural networks (ANN) have been applied to problems such as robot control, speech and signal processing, pattern recognition and, in a more general way, to biological simulation and cognitive sciences.

The topology and structure of the ANN may vary from one application to another: Hopfield nets, Kohonen nets, backpropagation nets are only a few examples.

It seems nevertheless that some fields have been forgotten by ANN applications. Economics and especially economic theory is one of these: as we know, only few applications exist, mainly related to loan managing or forecasting [1].

The aim of this paper is to describe one particular aspect of the theory of the household, a problem of static optimization, known as the Neoclassical Problem of the Household.

2. THE PROBLEM

We can define: the *household* as any group of individuals sharing income in order to purchase and consume goods and services; a *commodity* as a particular good or service delivered at a specific time and at a specific location.

The neoclassical problem of the household [2] is to choose a bundle of commodities that is "most preferred" among all the possible bundles, given:

- *the utility function*, which is a continuous, real-valued function defined on commodity space and based on some assumptions about the tastes of the household (i.e. weak preference relation and indifference between two bundles);
- *the budget constraint*, which states that total expenditure on all commodities cannot exceed money income.

In terms of the utility function, the problem is:

$$\max_{x_1,x_2,\ldots,x_n} \; U(x_1,x_2,\ldots,x_n)$$

subject to $\sum_{i=1}^{n} p_i.x_i = p_1.x_1+p_2.x_2+\ldots+p_n.x_n \leq I$

with
- I = household income ($I>0$)
- n = number of commodities
- x_i = quantity of the i^{th} commodity
- $\underline{x}$ = n dimensional vector of quantities
- p_i = price of the i^{th} commodity ($p_i>0$, $i=1,2,\ldots,n$)
- $\underline{p}$ = n dimensional vector of prices ($\underline{p}$ is given).

It is easy to demonstrate that the optimal solution yields to a total expenditure of the income, so the budget constraint may be formulated as $\Sigma\, p_i.x_i = I$.

It is important to remark that the analytical form of the utility function of the household is generally unknown. But some assumptions are made about $U(\underline{x})$:

- $U(\underline{x})$ is differentiable, and the first order partial derivatives (called marginal utilities) are all positive:

$$\frac{\delta\, U(\underline{x})}{\delta\, \underline{x}} > 0$$

- $U(\underline{x})$ is twice differentiable with continuous second order partial derivatives, and the Hessian matrix is negative definite, so the function is strictly concave and we have:

$$\frac{\delta^2\, U(\underline{x})}{\delta\, x_i^2} < 0 \qquad i=1,2, \ldots ,n.$$

The solution of the problem can be found using the Lagrange multipliers method, that is defining the Lagrangian function (y is the Lagrange multiplier)

$$L(\underline{x},y) = U(\underline{x}) + y.(I - \Sigma\, x_i.p_i)$$

and simultaneously solving the system of n+1 equations

$$\frac{\delta\, L(\underline{x},y)}{\delta\, y} = 0, \qquad \frac{\delta\, L(\underline{x},y)}{\delta\, x_i} = 0, \quad i=1,2, \ldots ,n.$$

3. THE NEURAL NETWORK

The aim of the neural network model is to simulate the behaviour of the household in its decision-making process: the choice of the n quantities of commodities to purchase, given the income and the n prices, such as to maximize his utility.

Assuming that the analytical form of the utility function is unknown, we consider that the preferences of the household are based on observed market choices. This set of observations (purchased quantities, prices and income) are used to train the neural network.

For the purpose of this simulation, the training set of 250 observations is generated randomly using the following utility function:

$$U(\underline{x}) = \prod_{i=1}^{n} (x_i+1)^{1-a_i} \qquad 0 < a_i < 1,\ \Sigma\, a_i \leq 1 \quad (i=1,2, \ldots ,n)$$

The neural network we used is the "classic" three-layer (input, hidden and output) feedforward network, with a sigmoidal activation function and with continuous, bounded (from

zero to a maximum) activation values.

The ANN is made of n+1 input units (n for the prices and one for the income) and n output units (for the quantities to purchase). The number of hidden units has been set experimentally to 2.n .

The learning rule used was the generalized delta rule [3] with back-propagation of the error.

The net completes the training after 35000 iterations, with the following parameters: learning rate 0.6, momentum 0.9, error tollerance 0.001, producing a maximum output error of 0.0009 and a total output error for the whole set of patterns of 0.065.

4. SIMULATION RESULTS.

Using the trained network to simulate the behaviour of the household, with values not belonging to the training set, leads to good results. The net achieves a good mapping of the utility function with a maximum error of 8% from generated to expected values in the whole range of activation values.

The good quality of these results is partly explained by the property of the generalized delta rule to minimize the squares of the differences between the actual and expected output values as well as by the uniform ditribution of the training set: if the ANN were trained with true observed values, not uniformly distributed in the allowed range, it could exhibit a less regular behaviour, with worse results.

5. CONCLUSIONS AND FUTURE DIRECTIONS

The capability of this simple neural network to simulate the behaviour of a basic microeconomic agent is quite encouraging.

We wish to develop an extension of this application, still related to economic theory, following two directions:

- the implementation of an ANN acting like another economic agent, the firm, which uses economic inputs (labor, capital, ..) to produce outputs of commodities sold to households and other firms, following a production function subject to some constraints;
- the realization of a tiny economic system made of several ANN (each of them corresponding to an economic agent, household or firm, with different utility functions and production functions) to simulate and analyze the problem of General Equilibrium [4] i.e. the interaction between the households and the firms, in the determination of prices and quantities of goods and input factors.

6. ACKNOWLEDGEMENT

The author would like to thank Pietro Terna, University of Turin, for his support and his comments concerning this paper.

7. BIBLIOGRAPHY

[1] Bailey D.L., Thompson D.M., Feinstein J.L.: Options trading using neural networks, Proceeding of International Workshop on neural networks & their applications, Neuro-Nimes '88, France.

[2] Intriligator M.D., Mathematical optimization and economic theory, Prentice Hall, 1971.

[3] Rumelhart D.E., McClelland J.L., Parallel Distributed Processing: Explorations in the microstructure of cognition, MIT Press, 1986.

[4] Samuelson P.A., Foundations of Economic Analysis, Harvard University Press, 1947.

Detection of Heart Malformation Using Error Back-Propagation Network

Borut Marič[1]
Dragan Beočanin[1]
Branislav Modrič[2]
Jošlko Buljan[3]

[1]CVTŠ KoV JNA, Ilica 256b, YU-41000 ZAGREB, Yugoslavia
[2]Vojna bolnica Zagreb, Aleja izviđača bb, YU-41000 ZAGREB, Yugoslavia
[3]OZIR, Bijenička cesta 97, YU-41000 ZAGREB, Yugoslavia

Abstract

The implemented system for heart shape classification consists of a preprocessor and a classifier. Following the presentation of the image preprocessor, the application of error back-propagation network to the problem of heart shape classification is exposed. The employed network structure is presented, as well as the methodology of training set construction, training and testing. Classification results are discussed.

1. Introduction

The research reported herein concerns the possibility of using neural network architecture for heart shape classification. An automated system for heart shape classification could be used for preliminary heart anomaly detection, based on chest radiographic images being required in mass-screening lung checks. The problem of automating heart shape classification is well known [1], and is usually solved in two phases. Firstly, values of heart shape parameters are extracted from digitalized chest radiographic image. Secondly, some standard classification algorithm is used to classify any given set of parameter values. We tried to use neural network architecture instead. The system consists of a digital image preprocessor and a classifier. The preprocessor finds the heart position on the digital image and extracts a set of heart shape parameter values from it. Based on these values, an error back-propagation network performs classification.
The rest of the paper is divided into three main parts: the preprocessor description, the network description, and the presentation of results.

2. The Preprocessor

The system has been designed having in mind detection of three heart shape categories: normal, aortal and myopathic, as sketched in Figures 1 to 3. The preprocessor implementation is inspired by some well known characteristics of chest radiographic images [1]. Preprocessing consists of several phases: early processing, detection of heart position, generation of binary image, correction of heart edge, and extraction of heart shape parameter values.

Early processing. Standard X-ray chest radiographic film of 36x36cm size is being digitalized to 512x480 pixels of 256 gray levels. To enhance subsequent processing, the image is zoomed to 240x240 pixels of the same intensity range. Taking into account characteristics of the camera optics and the shape of digitalized area, this process converts approx. 2.2x1.5 mm of original radiographic image to one pixel.

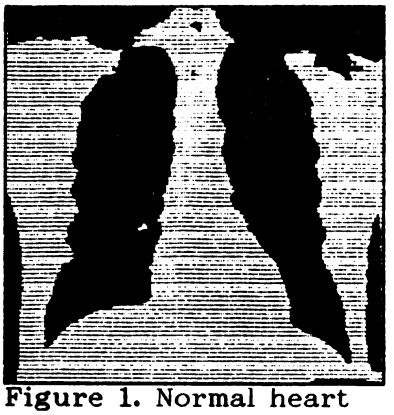


Figure 1. Normal heart

Figure 2. Aortal heart

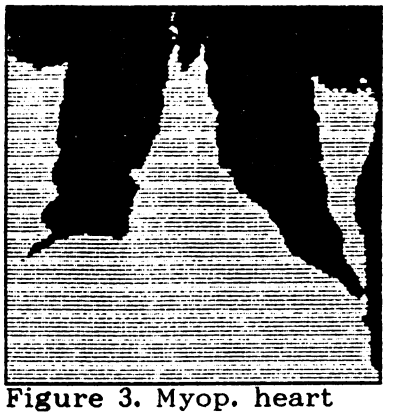


Figure 3. Myop. heart

Detection of heart position. Heart position is determined via horizontal and vertical spatial gray level signatures, as shown on Figure 4. This is being done in two steps. Firstly, signatures are constructed based on the whole image area. Using their extremes, positions of TOH, BOH, REC, LEC, REH and LEH are determined. Secondly, signatures are constructed taking into account only the area enclosed by TOH, BOH, REC and LEC. Using these new signatures, the process of TOH, ..., LEH determination is repeated.

Generation of binary image. To enable later extraction of heart shape parameter values, it is essential to determine the heart edge position. This may be done by converting lungs area to black, and heart cavern area to maximum brightness, i.e. making binary picture. The threshold value for each picture is defined in such a way as to produce fixed percentage of black (white) pixels in the area enclosed by TOH, BOH, REC and LEC. In this way it is possible to reduce the influence of average brightness oscillations among the images.

Correction of heart edge. Our preliminary research [2, 3] showed that high error rate of myopathic heart classification may be caused by preprocessing diaphragm shadow as a left lung edge. To remedy this flaw, left heart edge is being slightly corrected. This is being done using the information on maximum brightness changes in original 240x240 image, obtained by Kirsch directional gradient operator.

Extraction of heart shape parameter values. The set of heart shape parameters consists of distances between heart midline and right (left) heart edge. Parameter values are obtained in the following way: The interval from A to B (from A to BOH), Figure 4, is divided in fifteen subintervals. For each one of them the average distance between midline and right (left) heart edge is determined. Each value is normalized with distance from MDL to REH (from MDL to (LEC+LEH)/2). In this way, a set of thirty values in the range [0, 1] is obtained. These may be used to feed neural network classifier.

3. The Network

By comparison of statistical data obtained from 111 images (37 of each class) it became clear that from thirty heart shape parameters only nine can be used as the basis for classification. The differences between the classes among the rest of the parameters were insignificant, most certainly due to inevitable oscillations in image preprocessing. Hence, only nine of thirty possible parameters were used to feed the network. Classification experiments were performed with standard error

back-propagation three-layer network [4, 5], of 9-9-3 units respectively. Unit thresholds were free to change, learning rate constant was 0.25, and learning momentum constant was 0.9.

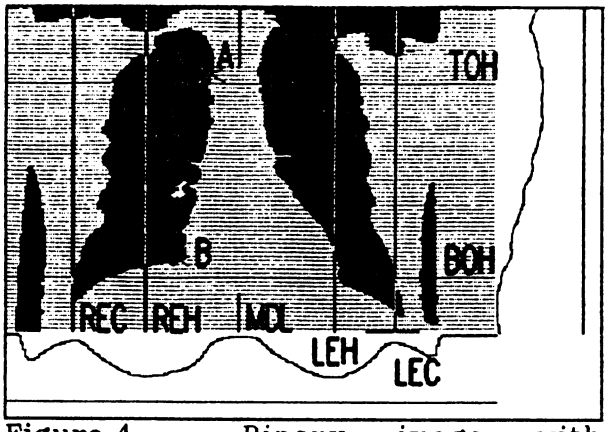


Figure 4. Binary image with signatures

4. Results

Construction of the training set. The training set consisted of four representatives of each class. Choosing them by subjective comparison of digitalized or original images produced no satisfactory results. They were therefore chosen as the best representatives (prototypes) of their respective class. This was obtained by selecting them in such a way as to minimize the difference between their parameter values and average parameter values of their class. From a set of 111 images, twelve were selected in this way. It may be argued that in this way some ideas of statistical classification theory were mixed with connectionism. This is true, and we consider it being a positive approach. One may suppose that an "infinite" testing set will have the same statistical characteristics of parameter values as the finite set used. Hence, there is a good reason to make an informed choice of the training set members, since that greatly affects the learning speed and later classification success.

Training. In each training epoch all members of the training set were presented to the network in the same order. Each input vector was paired with an output vector, forcing one of the three output units to 1, and the rest to 0. After 1200 epochs the total sum of squares error dropped to 0.0105.

Testing. The system's classification ability was tested using 99 images (33 of each class) which excluded the training set. The network output vector was interpreted in the following way: The class of the presented image was assumed to be the one associated with the output unit producing the maximum value. Results obtained using this procedure are summarized in Table 1.

	• Human specialist's classification: normal	aortal	myopathic
• Total number of images	37	37	37
• Number of training images	4	4	4
• Number of testing images	33	33	33
• System's classification:			
normal	15	0	7
aortal	10	23	10
myopathic	8	10	16

Table 1. Results

5. Discussion and future work

It is evident from Table 1 that from 47 - 67 per cent of each image class is correctly classified. Percentage of malformed heart shapes classified as normal is rather low (11% of all malformed cases). On the other hand, false alarms percentage (53% of normals) is intolerably high. There are several possible explanations for these, rather unsatisfactory results: the choice of heart shape parameters, the quality of parameter values extraction, and the small network training set. We have chosen rather straightforward representation of heart shape. Other approaches may be used as well, such as curve fitting. In order to keep the network small, we tried to use only thirty average distance values. Variations in positions of subintervals of heart height led to the loss of some valuable information on the shape of heart aortal area. Finally, the training set was deliberately chosen to be small - it is about nine times smaller than the testing set - to enable realistic testing with images at hand. If we obtain more images, substantially enlarged training set would be used.

We plan to evaluate usefulness of some other heart shape encoding schemes, and enhance preprocessor quality. If that proves successful, we shall try to enlarge the set of heart classes (i.e. malformation types to be classified).

Acknowledgement

The authors acknowledge the helpful advice of Prof. Andrija Hebrang of the Medical Faculty Hospital "Dr. Ozren Novosel" Zagreb, in the early stage of this research. In some phases of the research Parallel Distributed Processing Software Package, enclosed with [5], has been used, which is gratefully acknowledged.

6. References

[1] Hall E L, Kruger R P, Dwyer III S J, Hall D L, McLaren R W, Lodwick G S, A **Survey of Preprocessing and Feature Extraction Techniques for Radiographic Images,** IEEE Transactions on Computers, vol. C-20, no. 9, 1032-1044 (1971)

[2] Maričić B, Beočanin D, Buljan J, **Detection of Heart Position on Radiographic Image,** Proceedings of 33rd Yugoslav Conference on Electronics, Telecommunications, Automation and Nuclear Technique (ETAN 89), Novi Sad, Yugoslavia, June 12-17, 1989, vol. EK (in Serbo-Croatian)

[3] Maričić B, Hebrang A, Beočanin D, Buljan J, **Processing of Heart Shape Parameters Using Neural Network Architecture,** Proceedings of 10th International Symposium on CAD/CAM (PPPR 89), Zagreb, Yugoslavia, October 17-19, 1989

[4] 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

[5] McClelland J L, Rumelhart D E, **Explorations in Parallel Distributed Processing - A Handbook of Models, Programs, and Exercises,** The MIT Press, Cambridge, Massachusetts, 1988

STABILITY ANALYSIS OF POWER SYSTEMS USING MULTI-LAYER PERCEPTRON

D. Rao Marpaka, Seyed M. Aghili, and Michael H. Thursby

Department of Electrical and Computer Engineering
Florida Institute of Technology, Melbourne, FL 32901

ABSTRACT

Transient stability analysis of power systems by Liapunov's second method has been an area of active research for the past twenty five years. This method allows the stability of a system to be determined without directly solving the system state equations. In this paper we present a method for determining the stability of a given system under various conditions. This technique uses a multi-layer perceptron with the back propagation learning algorithm. Three network architectures have been analyzed to determine the most effective one for this type of problem. By using the back-propagation algorithm, the network is trained with known input patterns for the power system and corresponding Liapunov function as the desired output. After the network attained the capability of recognizing the stability region of known system, a set of data representing different operating conditions was presented to the network to determine the state of the system.

INTRODUCTION

In order to determine the most effective architecture for this problem we evaluated three different architectures shown in Fig. 1, for mean square error and convergence. The architecture in Fig. 1b was the most effective for this problem. This architecture has five inputs, six units in each hidden layer and one unit in the output layer. The performance characteristics of these three architectures are analyzed and the results of this analysis are given in Fig. 2. Fig. 2a depicts the mean square error of the given architectures with respect to the number of training sessions and Fig. 2b shows the relation between the different values of momentum and the number of iterations required for convergence.

In the training phase the back-propagation learning algorithm has been used. Initially the network was presented with random set of interconnection weights and thresholds. The network with a learning constant of 0.05, momentum step size of 0.6 has converged with the mean square error bound of 0.04 pu.

In the testing phase the network is presented with a new set of data belonging to different operating conditions. The results of the simulation are presented in Table-1.

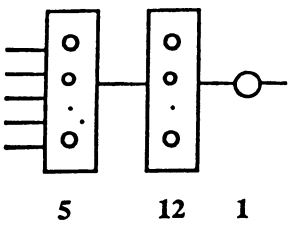


Fig. 1a

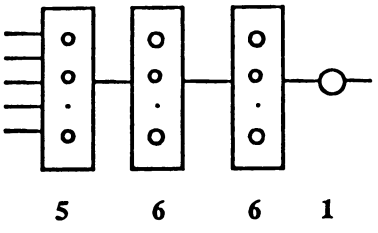


Fig. 1b

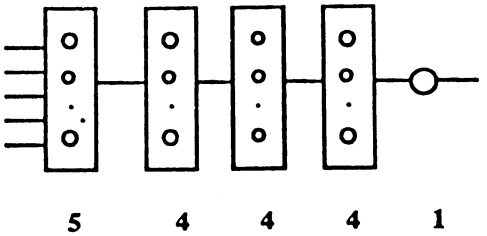


Fig. 1c

BACK PROPAGATION ALGORITHM

The back-propagation learning algorithm was used to train the proposed network. In each training session the input units were presented with two state variables and three parameters of Eq.(13) D,P_i, and δ. The state variables are normalized to the range of (-1.0) to (1.0). The output of each layer was propagated forward through each layer by using Eq.(1) and the sigmoid nonlinearity f(.), Where in Eq.(1), X_k represent the input data, W_{ik} is the weight from kth to ith unit and θ_i is the threshold of the ith unit.

$$Y_i = \sum_k W_{ik} X_k + \theta_i \qquad (1)$$

Each weight was updated after the output had been compared with the desired output using Eq.(2)

$$W_{ik}(t+1) = W_{ik}(t) + \eta \delta_k f'(Y_i) + \alpha[W_{ik}(t) - W_{ik}(t-1)] \qquad (2)$$

η is the step size, α is the momentum term, and δ_k is an error term for node k. If node k is an output node, then

$$\delta_k = (D_k - Y_K) f'(Y_k) \qquad (3)$$

If node k is an internal node then

$$\delta_k = \sum_l \delta_l W_{kl} \cdot f'(Y_k) \qquad (4)$$

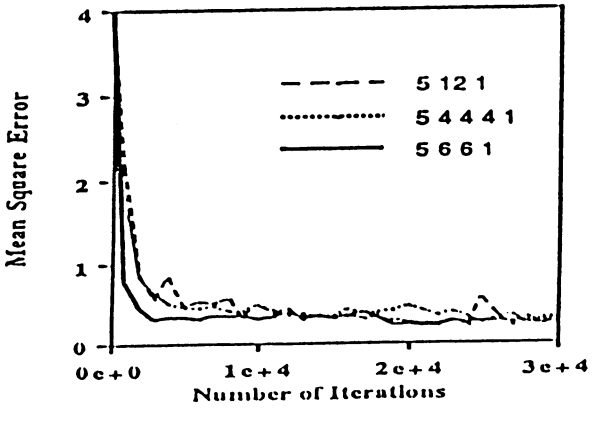


Fig. 2a

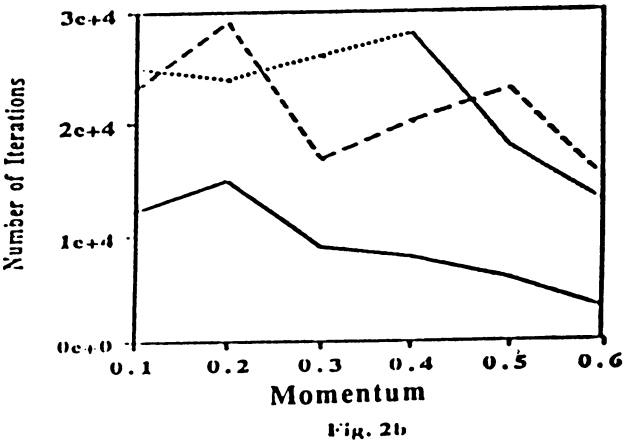


Fig. 2b

SIMULATION OF POWER SYSTEM

The block diagram of a single-machine system with damping is given in Fig. 3. the following assumptions were made for the system under investigation: i) Input to the machine is constant, ii) Field-Flux linkages are constant, iii) Damping power is proportional to slip velocity, and iv) Network is purely reactive.

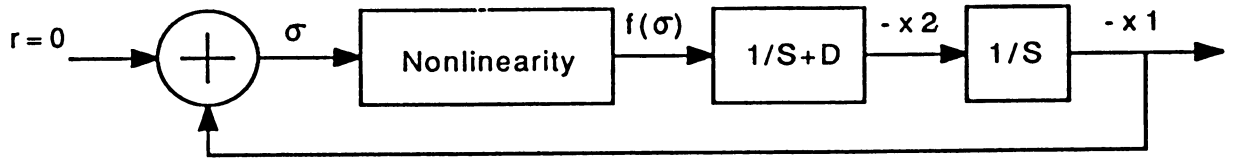


Figure 3

The normalized equations of a synchronous generator-infinite bus system in the post fault state can be written as

$$\ddot{\delta} + D\dot{\delta} = P_i - Sin\,\delta \tag{5}$$

Where δ is the power angle, P_i the mechanical input, and D the damping coefficient. The equilibrium state $\delta_0 = Sin^{-1}P_i$, $\dot{\delta}_0 = 0$ can be transferred to the origin by defining new variable

$$x = \delta - \delta_0 \tag{6}$$

$$\ddot{x} = -D\dot{x} + P_i - Sin(x + \delta_0) \tag{7}$$

$$\ddot{x} = -D\dot{x} + P_i(1 - Cosx) - \sqrt{1 - P_i^2}\cdot Sinx \tag{8}$$

By defining

$$x_1 = x \tag{9}$$

$$x_2 = \dot{x} \tag{10}$$

$$f(x) = -P_i(1 - Cosx_1) + \sqrt{1 - P_i^2}\cdot Sinx_1 \tag{11}$$

The state equation is given by

$$\begin{aligned} \dot{X} &= AX + Bf(\sigma) \\ \sigma &= CX \end{aligned} \tag{12}$$

Where

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -D \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad C = [1 \quad 0]$$

The Liapunov function for the above system is given by

$$V(x_1, x_2, P_i, D, \delta_0) = \frac{Dx_1^2}{2} + \frac{(1+D^3)x_2^2}{2D} + x_1x_2 + \frac{(1+D^3)}{D}(Cos\,\delta_0 - Cos(x_1 + \delta_0) - P_ix_1)\,(M_2$$

$$M_2 = \frac{(\pi - 2\delta_0)^2 D^4}{2(1+D^3)} + \frac{(1+D^3)}{D}(2\cos\delta_0 - P_i(\pi - 2\delta_0)) \qquad (13)$$

The network was presented with the data representing various operating conditions of the power system. This data was not presented during the training session and the results are compared with actual state of the system. The results are given in Table-1.

X_1	X_2	P_i	δ_0	D	Actual State	Computed State
0.2	0.2	0.4	0.4 5	0.01	Stable	Stable
0.6	0.3	0.7	0.3	0.02	Stable	Stable
1.5	0.9	0.9	0.7	0.5	Unstable	Unstable
1.6	1.2	0.5	0.8	0.08	Unstable	Unstable

Table-1. Test Results

CONCLUSIONS

This paper presents a method to determine the stability of power system under various operating conditions using feedforward artificial neural network. Due to parallel processing of information,neural networks perform simple operation to classify the states of the system . This technique is potentially more efficient and less complex analysis. The MSE of three layer network presented earlier can be further reduced by choosing proper momentum and learning rate.The system analyzed involves five parameters causing complexity in computations. This method provides simple and faster solution to the problem. This method can be applied to the stability analysis of Power Systems without damping,with Transfer Conductances, velocity governor and Saliency. This method can be used to determine the Critical Clearing Angle of Power Systems. These are currently under investigation and will be reported later.

REFERENCES

[1] **R. P. Lippmann,**"An Introduction to computing with Neural nets", *IEEE ASSP* Magazine, Vol. 4, No. 2, April 1987.

[2] **D.E.Rumelhart and J.l. McClelland,** Parallel Distributed processing, Vol 1, Cambridge, MA, MIT Press, 1986.

[3] **M.A.Pai, M.Ananda Mohan, and J. Gopala Rao,**"Power System Transient Stability Regions Using Popov's Method", *IEEE Trans. on Power Apparatus and Systems*, Vol. PAS/89,No.5/6, May/June 1970.

INTERFACING DATA BASE TO FIND THE BEST AND ALTERNATIVE SOLUTIONS TO PROBLEMS BY OBTAINING THE KNOWLEDGE FROM THE DATA BASE

By
O. Enrique Martinez and Craig Harston
C.A.S.
858 Oak St. #1
Chattanooga, TN 37403

Abstract

This is a neural network that get its knowledge from a data base. The system is 100 % accurate and can store several thousands of associations. The memory size is defined by the characteristics of the data base. One special property of the system among others is that at recall time, the input can be weighted differently to emphasize the importance of some values over others. The system also provides alternative solutions with a response time near zero for every output neuron.

Introduction

Data bases are the most important source of information. Tying this information to a neural network can create a new system with expertise in any field. The system can provide many answers to a particular input in a well organized manner. As the number of associations increase by hundreds, many of these associations become similar and this makes the need for a very accurate system. The system can find a single matching solution to a particular input if there is only one solution, but it can also respond with alternative solutions to those situations when the input is considerate by us, and no by the system, to be incomplete.

To keep certain consistency with data bases, the system uses the many to one relationship. This relation provides the uniqueness of one element with respect the others. The output generalization depends on the input and the number of matches found by the recall.

Properties of the system

1) 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. This particular system is not one goal oriented. It does not look for one exact match to the given input. The goal is to find all the possible matches by using the input as clue and present the answers to the user in a well organized manner. For many applications, the matching solution to an input could be the worst solution among the other possible responses. We will illustrate this with two examples:

Business application: Suppose that you are looking for businesses that offer benefits in any given criteria. Instead of selecting a business that only meets the requirements, you can have a listing of businesses that meet your needs and provide more benefits. The goal is to find those companies that offers the most by meeting your needs as a minimum condition. Figure B is an example in which the response 3K (see figure 1) is the best match with very little to offer for this hypothetical case, but 3J is the optimal solution because has the most to offer.

Medicine: Here we need information about symptoms and diseases. By given the most notable symptoms of a patient, the system can recognize the disease without presenting all the symptoms. It it is more important to detect what else could develop if the current condition of the patient is not stabled. A single answer is not enough.

The business example shows that the best matching response for a given input can be considered far remote from the best solution to many problems. The other example shows that one input can require more than one answer.

2) Another property of the system is that it can work with inconsistent data. While other system using back-propagation fail to work with inconsistent data [1], the system is well adapted to this common circumstances. We frequently have two or more opinions or facts about a common subject, for instance: "my car is new" and "my car is red". A question to the system like "my car is ____" , the system will simple respond with a combined answer "new " and " red ".

NEURAL NETWORK DEVELOPMENT

System memory: The system is built using the Hebb learning rule. The size of the memory is n by p where the maximum number of association is equal to p. The memory is the outer product of these two vectors.

vectors = Z(n), U(p)
matrix = T(n,p)

$$T_{ij} = T_{ij} + (Z_i * U_j)$$

General Neural Networks Problem

Saturation is a problem that limit the number of associations that can be stored in memory. Saturation also forces systems to increase unnecessarily the size of the memory in order to increase the number of associations even for a small fixed size application.

The BAM formula (Haines and Hecht-Nielsen ,1988) determines the limitation of the other systems [5]. Having two fields n and p, we take the smallest field to find the maximum number of associations m = r/(2log r) where r < min(n,p). Table 1 shows some r values used to reach a certain number of associations.

r	m	matrix size (r2)
8	4	64
15	6	225
274	56	75,000
5,000	675	25,000,000
100,000	10,000	10,000,000,000

TABLE 1

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 might have the same MH frequencies. The MH frequency is stored in the output neuron array. A duplicate of these values are sorted and stored in tables (figure A, B, C, D, and E).

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 the response in a desired order. In all the figures, the highest calculated frequency corresponds to the response that best match the input. The other frequencies correspond to the best alternative solutions.

Testing the system: Running one of our experimental data to test a business information system, we have an input area with 15 input neurons (attributes) and 10,100 output neurons (companies). The memory size was the product of 15 * 10,100= 151,000 and the number of associated patterns was 10,000.

Results: We have 100% accuracy for every recall made. We worked exactly with the number of attributes needed in the input field (fifteen). There was no need to increase the number of input neurons to 100,000 as suggested on table 1. The response time after the recall for additional information was near zero for each output neuron.

Simultaneous recall

It is possible to recall simultaneously two or more associations that are different or to emphasize the important of one attribute over other. Figures B and C show the recall of two different associations and figure D show the simultaneous recall with weighted input. The half rectangle has a weight of two and the large rectangle has a weight of 1. Note how both associations are presented when using the smallest frequency value.

Bypassing the memory

After storing the MH frequency values in 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.

Conclusion

A important characteristic of the system is that its knowledge is growing as the number of association increase without being saturated. The system can become an expert adviser using information from data base. The other characteristics are: memory size is relative to the data base attributes and the number of entries, it can perform in real time, the input can be weighted differently, and it works with inconsistent data.

REFERENCE

[1] Mark Lawrence, "Neural-net learning time shaved," Electronic Engineering Times, pp. 36-38, June 26, 1989.

[2] Sukhan Lee and Rhee M. Kil, " Bidirectional Continuous Associator Base On Gaussian Potential Function Network," International Joint Conference On Neural Networks. June 1989, Vol I, pp. I-45.

[3] J. A. Anderson, "A simple neural network generating an interactive memory," Mathematical Biosciences, vol. 14, pp.197-220, 1972.

[4] T. Kohonen, "Correlation matrix memories," IEEE Transactions on Computers, vol. C-21, pp.353-359, 1972.

[5] Patrick K. Simpson " Heterocorrelation Associative Memories - BAMs ," Higher-Ordered and Intraconnected Bidirectional Associative Memories , General Dynamics Electronics Division, pp. 15, Dec.1988.

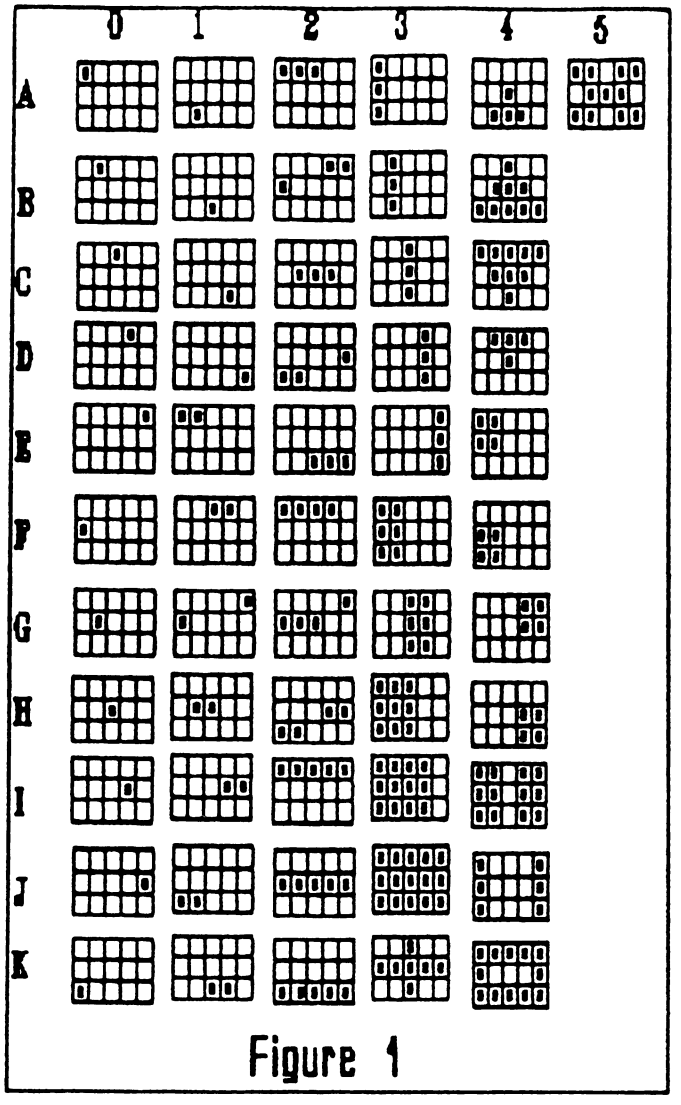


Figure 1

Figure 1 has 56 associations stored in a 15 by 88 matrix.

Example:

this figure is associated to the value 3C

The same association is found by following the coordinates in figure 1.

Example C

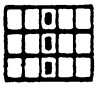


FIGURE A

INPUT PATTERN

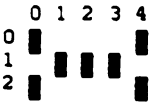


Table	Responses
180.740448	→ 5A
180.444153	→ 3J
176.518219	→ 4I
172.592300	→ 4B, 4C
172.370071	→ 3I
168.666367	→ 4J
168.444153	→ 3H
168.221924	→ 4K

FIGURE B

INPUT PATTERN

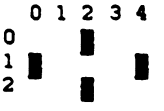


Table	Responses
91.622345	→ 3K
91.244987	→ 4K
91.018570	→ 3J
87.395927	→ 3H
87.169510	→ 3I

FIGURE C

INPUT PATTERN

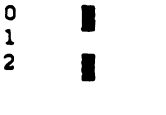
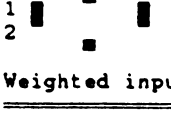


Table	Responses
38.221924	→ 3C
37.999702	→ 3G
37.925629	→ 3K

FIGURE D

INPUT PATTERN



Weighted input

Table	Responses
147.622345	→ 3K
147.244980	→ 4K
147.018570	→ 3J
143.395935	→ 3H
143.169510	→ 3I
139.773285	→ 3C
139.546875	→ 3G

FIGURE E

INPUT PATTERN

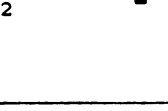


Table	Responses
11.407107	→ OJ
11.333033	→ 1I
11.258959	→ 2D,3E

All the responses for each input set are obtained after one recall. The reaction of each output neuron is measured and stored in the tables. The neurons with the highest reaction (pointed by the arrow) correspond to the best matching solution.

An Interactive Activation and Competition Model for Machine-Part Family Formation in Group Technology

Young B. Moon
Department of Mechanical and Aerospace Engineering
Syracuse University

425 Link Hall
Syracuse, NY 13244

Introduction

Group technology (GT) is a manufacturing principle which is based on the hypothesis that a higher level of efficiency and integration can be achieved by grouping similar parts into part families. The idea is that through a process of generalization the complexity of raw data will be reduced into a manageable size of clusters. For instance, a factory manufacturing more than 10,000 different parts may take advantage of the reduced number of 40 or 50 distinct part families. The group technology principle has been applied in various areas of design and manufacturing such as variant process planning system, group technology cell formation, group technology layout of machining tools, and so on.

A machine-part family formation problem is the way in which the set of parts are partitioned into families, and then the set of machines are grouped into manufacturing cells based on the part families [1]. Typically, a set of machines through which a part must be processed is specified by route sheets or process plans. This information can be represented by a matrix form. That is, each row corresponds to a different part type while each column corresponds to a different machine type. A component of the matrix has a value of one if the corresponding part type needs to be processed in the corresponding machine type. Once the given information is represented by a matrix format, most of the family formation approaches have attempted to rearrange the sequence of rows and columns to find a block diagonal form of the matrix. [2] For example, an initial machine-part matrix and a desirable form of the matrix after family formation procedure are presented in Figure 1.

		parts						
		1	2	3	4	5	6	7
	1		1		1	1	1	
	2	1		1				
machines	3	1		1				1
	4		1		1		1	
	5	1						1

		parts						
		1	3	7	2	4	6	5
	3	1	1	1				
	2	1	1					
machines	5	1		1				
	1				1	1	1	1
	4				1	1	1	

Figure 1

King and Nakornchai classified existing approaches to the family formation problem as similarity coefficient methods, set-theoretic methods, evaluative methods, and other analytical methods [3]. However, none of the approaches had taken the advantage of parallel distributed processing. This paper proposes a new approach to the family formation problem employing the spontaneous generalization capability of an interactive activation and competition model [4].

A neural network for machine-part family formation

Three types of information are assumed to be given initially: a set of part types, a set of machine types, and a machine-part matrix from routing sheets. Each part type and machine type is used as a processing unit of the network. The processing units are grouped into three different pools. Each pool consists of processing units which respectively represent part types, machine types, and part instances.

There are two kinds of connections involved in this network. One is based on the machine-part matrix, and the other based on the similarity matrices derived from the machine-part matrix.

The weights between part instances and part types, and between part instances and machine types are given as a maximum value if there is a connection, if not, the weight is given as 0. The null weight means that there is no significant connection between the two processing units. Normally, the maximum value chosen is 1.

The weights among part types and among machine types are determined by the similarity matrix. The similarity matrix for part types is constructed by counting the number of the same value between two part types, while the similarity matrix for machine types is constructed by using the same method between two machine types. Diagonal values are given as equal to the number of components.

The weights among types are calculated by the following formula: $w_{ij} = (s_{ij} / n)$ - threshold. Here, s_{ij} is the value from the similarity matrix, and n is the number of components of the matrix. The value of threshold is usually chosen as 0.5. Therefore, the value of w_{ij} lies between -1 and 1. The negative weight means that there is inhibitory activity between the two processing units. An example of a machine-part matrix and the constructed network are given in Figure 2.

Each processing unit either receives an external input or output signal from the connected units. A combined input to the processing unit, i, is calculated as follows [4]:

$$net_i = \sum_j w_{ij} \ output_j \ + \ extinput_i$$

where $output_j = [act(j)]^+$. Here, act(j) represents an activation value of the unit j. Also, $[act(j)]^+ = a_j$ if $a_j > 0$, otherwise $[act(j)]^+ = 0$.

The activation values are updated according to the following equation:

If ($net_i > 0$),

delta $a_i = (max - a_i)\ net_i$ - decay $(a_i$ - rest)

otherwise,

delta $a_i = (a_i$ - min) net_i - decay $(a_i$ - rest)

where max = 1

min <= reset <= 0

0 < decay < 1.

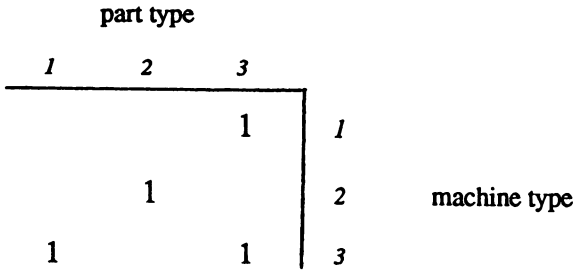
Once a neural network has been established, the family formation is achieved through the following simple procedure.

Step 1: Select a part and give an external signal to the part.

Step 2: Run the neural network simulation and store the result.

Step 3: If all the parts are assigned, stop. Otherwise, go to step 2.

Unlike the traditional approaches, the proposed procedure does not generate a matrix form. The groups are identified from the final activation values of each processing unit. That is, all the units having positive activation values are grouped together.



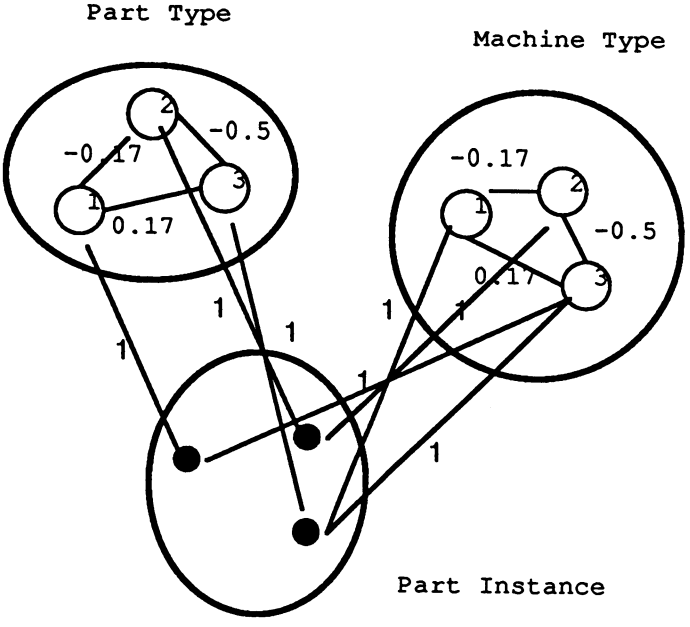


Figure 2

Conclusion

The described approach has been tested using a simulation program [5] on various machine-part family formation problems. The preliminary results demonstrate several advantages of using the proposed method. First, it departed from the most commonly used matrix manipulation in the family formation problem. In the other approaches, the final decision involves another step to identify groups from the rearranged matrix because some overlapping of parts among different manufacturing cells is unavoidable. However, the neural network approach produces a reasonable result without examining the final matrix again. This feature will open up more possibilities of integration with other computer aided manufacturing functions.

Second, the approach takes advantage of the inherent parallel processing of neural networks. Even a huge problem, for example, a problem with 1000 parts and 50 machines, can be solved instantaneously.

Third, the neural network approach can accommodate additional information. For example, when certain parts need to be grouped together the only necessary change to make is to increase the weight between the parts.

Fourth, the procedure can be initiated either from part types or from machine types. Since there is no distinction between part types and machine types when a network is constructed, the described procedure can start from either type. More sophisticated approaches may be devised based on this characteristic.

The most important potential of this approach is that it combines two functions of group technology, that is, classification/coding and family formation. The classification/coding function used to be based on the part design specification, mainly from part drawings, whereas the requirement matrix for family formation is based on routing sheets. Obviously, these two functions have not been integrated well.

The neural network model can also incorporate additional feature pools without discarding the already built connections. This will lead to the successful retrieval of incomplete or ambiguous description of part geometry, and the proper assignment of the part.

References

1. Shtub, A., Modelling group technology cell formation as a generalized assignment problem, *International Journal of Production Research*, vol. 27, no. 5, pp. 775-782, 1989.

2. King, J.R., Machine-component grouping in production flow analysis: an approach using a rank order clustering algorithm, *International Journal of Production Research*, vol. 18, no. 2, pp. 213-232, 1980.

3. King, J.R., and Nakornchai, V., Machine-component group formation in group technology: review and extension, *International Journal of Production Research*, vol. 20, no. 2, pp. 117-133, 1982.

4. Rumelhart, D.E., Hinton, G.E., and McClelland, J.L., A General Framework for Parallel Distributed Processing, *Parallel Distributed Processing Volume 1: Foundation* (by Rumelhart & McClelland), Bradford, 1987.

5. McClelland, J.L., and D.E. Rumelhart, *Explorations in Parallel Distributed Processing*, Bradford, 1988.

A Neural Net Approach to Electronic Circuit Diagnostics

James R. Reeder and L. James Koos
Westinghouse Integrated Logistics Support
111 Schilling Road
Hunt Valley, MD 21031

Westinghouse Science & Technology Center
1310 Beulah Road
Pittsburgh, PA 15235

1. INTRODUCTION

Our project aims to exploit the learning ability of artificial neural networks for identifying or classifying faults in electronic circuits. The obvious utility of detecting and isolating the faulty components after the fact would be sufficient motivation, but there are additional alluring possibilities. Conceivably, an on-board neural net would be able to monitor circuit performance in real time. Warnings of impending failure due to degradation of components may be possible, as might dynamic reconfiguration in order to continue service until maintenance can be done.

The methods that we are developing is required to be useful on a wide variety of circuits, both analog and digital. We have to pragmatically avoid the combinatorial (or worse) explosion of failure possibilities. In analog circuits, it is apparent that even the failure-free nominal case cannot be examined exhaustively. Therefore, it is essential that we concentrate on representative input signals and realistic fault subsets, based on historical component reliability data [1].

Avoiding the preparation of training sets from hardware prototypes, we use computerized fault simulation, taking advantage of available circuit simulation tools. Near-term practical applications will continue to make extensive use of conventional system elements.

2. APPLICATION ORGANIZATION

The system architecture, i.e., its functions and information elements, suitable for a research or a production system, include the following:

- A circuit model description for a computer-aided design (CAD) tool.
- A simulation tool.
- A library of information describing effects on the circuit of faults likely to occur.
- A handler capable of injecting faults in the circuit model and building a training set from the results.
- A means of presenting the training sets to a neural net (model representation [2]).
- A neural net model with the appropriate parameters.

The initial learning paradigm we are using is back-propagation [3], but counter-propagation [4] is also under consideration.

3. APPROACH

Rather than attempt to design the ultimate system immediately, we have graded the types of circuits we want to examine according to difficulty. That way the viability of the architecture can be verified without being impeded by modelling complications — in effect, a proof of concept demonstration. To simplify, we have picked combinational (memory-free) logic circuits as the first target. Their input and output sets are finite; for

relatively small circuits, even exhaustive combinations are feasible. A wealth of information exists on digital test generation [5] to which we may turn for comparison, and reasonably expected faults (e.g., "stuck-at-0", "stuck-at-1") can readily be enumerated. For the present, we do not have to be concerned with the optimality of the test set regarding fault coverage and discrimination. Logic simulation is limited, however, in that it cannot emulate certain kinds of failure, such as power supply fluctuations.

Even combinational logic circuits, when large enough, can be impractical to analyze completely. Digital circuits having internal state (e.g., sequential logic) cause some obvious complications, but still have finite input and output sets. Technically, that is not the case with analog (continuous) and hybrid circuits, whether they are state-free or not. (By state-free we mean that the output signals are instantaneous functions of the input signals, similar to combinational logic.) All the preceding situations involve some simplifying assumptions in their modelling. The hardest case we can model is any circuit that can be represented by linear or restricted non-linear differential equations (e.g., with SPICE).

Most success with neural net classifiers appears to have been achieved when the inputs are quantitatively related. Then, in some sense, the learned relationships have some chance of approximating nearby signals correctly, using the intuitive idea of distance in a multidimensional pattern space. The point of this to permit explicit differences to be formed, between the observations associated with a given fault mode and the corresponding nominal circuit's observations. Two parts of each training case are straightforward: The vector of input values and the vector of visible output values. (We have to allow for the situation where some components' outputs are inaccessible.) The identification of the fault mode is more troublesome. One is inclined to label the fault mode symbolically; an implicitly structural labeling adheres better to the quantitative criterion. To achieve this, we form a list of the components; each item marked "0"

is normal, while non-zero marks correspond to a fault, the value designating the nature of the fault. Although this still has a residual symbolic aspect in that arbitrary designation, the importance of the kind of fault is secondary.

4. SCOPE OF RESEARCH TO-DATE

A system prototype has been evaluated with a familiar small circuit, the single-bit full adder, built from *nand* gates. This has been exhaustively analyzed; it has three inputs and two intentional outputs. Our implementation used 11 *nand* gates, all being visible. Another candidate under review, is a combinational logic circuit (proprietary), containing approximately 100 gates. Here, we do have access to the CAD circuit model description and simulator. The input combinations and fault modes are deliberately incomplete, i.e., not exhaustive, in the training set. Also, the circuit design limits the visible gate outputs. Nevertheless, we still expect to be able to identify some fault modes outside the training set.

5. REFERENCES & NOTES

[1] See databooks published by U.S. Reliability Analysis Center, RADC Griffiss AFB (Rome), NY 13441.
[2] Specifically, in this case, on an ANZA Plus (Hecht-Nielsen Neurocomputers).
[3] D. E. Rumelhart & J. L. McClelland, **Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume I: Foundations**, MIT Press, 1986.
[4] R. Hecht-Nielsen, "Counterpropagation Networks", *Proceedings IEEE First International Conference on Neural Networks, II, 19-32*, 1987.
[5] For example, A. D. Friedman & P. R. Menon, **Fault Detection in Digital Circuits**, Prentice-Hall, 1971.

A NEURAL NETWORK IMPLEMENTATION OF PARALLEL SEARCH FOR MULTIPLE PATHS

LYLE A. REIBLING and MICHAEL D. OLINGER
SMITHS INDUSTRIES
SLI AVIONIC SYSTEMS CORP.
4141 Eastern Avenue, S.E.
Grand Rapids, Michigan 49518

Abstract

The SLI Avionic Systems Corp. (SLIASC) of Smiths Industries has performed Independent Research and Development to investigate advanced parallel processing architectures for path planning applications. We are investigating neural network technology for flight management (FMS) and vehicle management (VMS) system applications of optimal trajectory generation in embedded systems. This paper describes a parallel search model for multiple path generation and its neural network implementation.

INTRODUCTION

This paper describes a neural network architecture for finding multiple paths. The mathematical model which is the basis for this architecture is based on a conjecture that the problem of finding the best path through a region which contains a variable cost function is similar to the problem of finding the maximum current flow through an nonuniform conducting media, such as a plate of inhomogeneous resistive material. The variable cost function is analogous to the nonuniform conductivity of the plate media. Boundary conditions for the source potential and sink potential are analogous to the start and goal nodes, respectively. By numerically solving the finite difference approximation to the appropriate partial differential equation which describes the scalar electric field potential in the media, the current flow can be found as a vector field.

Experimental investigation of this technique has shown promising results. The solutions generated by the simulated architectures have been checked against known admissible algorithm results.

BACKGROUND & PROBLEM DESCRIPTION

The path planning problem consists of a large problem space, typically with several state variables. Due to its large search space, the path planning problem places severe demands on computational resources. These problems often require real-time solutions unavailable from current technology. Path planning is achieved by searching over all possible paths in a multidimensional search space for the path with the optimal performance measure.

A difficult problem in path planning applications is computing an aircraft trajectory[9] in real-time onboard a high performance aircraft. For tactical aircraft the objective of trajectory generation is to increase the aircraft survivability and mission effectiveness by flying low to the terrain to avoid threat radar detection, referred to as threat penetration[10]. The quantitative measure of performance is usually expressed as a probability of survival P_s along the aircraft trajectory. This performance measure attempts to capture a numerical representation of the probability of successfully completing the mission. This is met by finding a trajectory through the threat region for which the performance measure is maximized. Rather than maximizing the probability of survival performance measure, the problem is converted to the minimization of a performance measure which is a function of the negative logarithm of the probability of survival[9]

$$D = \mathcal{F}(-\log P_s) \tag{1}$$

This function computes a *danger metric* based upon the survivability in a dense threat environment taking into account the threat locations, lethality, aircraft exposure due to altitude, and masking effects from the terrain. This metric is used as the cost function for the search by assigning a danger value to each state.

This research[12] has investigated computer architectures and algorithms characterized by massive parallelism which solve combinatorial search problems. Much of the research regarding combinatorial search algorithms has focused on algorithm development and analysis for sequential processors, or distributed multiprocessors. The computational architecture known as the systolic array[3] derives from its similarity to a grid, with each point

of the grid as a processor, and the links of the grid consisting of the links between the processors. This array characteristic of the architecture capitalizes on the regular and modular structures which match the computational requirements of the algorithm. Current research interest[4] addresses developing new computational architectures and algorithms motivated by the massive parallelism of the brain. Neural network architectures for combinatorial search problems[5] are an example of such new approaches.

MATHEMATICAL MODEL

The mathematical model is based on electric field theory[8]. From the definition for current density and its divergence in steady state conditions the electric field potential of a nonuniform conductive media can be derived[11] as

$$\nabla^2\phi + \frac{1}{\sigma}\nabla\sigma \cdot \nabla\phi = 0 \tag{2}$$

where σ is the nonuniform conductivity of the media. For this research, the danger (1) is used to model the resistivity, or (inversely) the conductivity σ of the conducting media. Expanding the gradient and divergence operators in two dimensions results in the following second order partial differential equation

$$\phi_{xx} + \phi_{yy} + \frac{1}{\sigma}\sigma_x\phi_x + \frac{1}{\sigma}\sigma_y\phi_y = 0 \tag{3}$$

Since the electric field and current density results from the gradient of a scalar field, they are conservative fields, and the field lines emanate from source charges and terminate on sink charges[13]. These field lines represent the solution to the search problem as multiple paths. Several properties of the paths are noteworthy. First, every heading from the start node has a defined path due to the continuous vector field emanating from the source. Second, every location which has non-zero conductivity has a path defined through it for the same reason. Third, the model supports zero and infinite danger regions.

NEURAL NETWORK IMPLEMENTATION

The neural network is employed to perform a parallel computation of the scalar potential field ϕ at every spatial point. This requires the network to compute a numerical solution to (3). Thus, the architecture of the neural network must solve an elliptic partial differential equation.

There are many numerical methods for solving partial differential equations[1,6]. In this research, finite differences[2,7] were employed to solve the potential field equation (3). Such an approximation is the five point formula

$$\phi_{i,j} \approx v_{i,j} = \frac{1}{4}[(1 + \frac{\sigma_x}{2\sigma})v_{i+1,j} + (1 + \frac{\sigma_y}{2\sigma})v_{i,j+1} + (1 - \frac{\sigma_x}{2\sigma})v_{i-1,j} + (1 - \frac{\sigma_y}{2\sigma})v_{i,j-1}] \tag{4}$$

For the neural network implementation, the neuron input function $u_{i,j}$ is defined as

$$u_{i,j} = \frac{1}{4}[(1 + \frac{\sigma_x}{2\sigma})v_{i+1,j} + (1 + \frac{\sigma_y}{2\sigma})v_{i,j+1} + (1 - \frac{\sigma_x}{2\sigma})v_{i-1,j} + (1 - \frac{\sigma_y}{2\sigma})v_{i,j-1}] \tag{5}$$

The synaptic weights implement the non-uniformity of the cost functions (the coefficients of (5) which contain σ_x and σ_y). The nonlinear neuron output function $v_{i,j}$ is defined as

$$v_{i,j} = \begin{cases} -V_{ref} & u_{i,j} < -V_{ref} \\ u_{i,j} & -V_{ref} \le u_{i,j} \le +V_{ref} \\ +V_{ref} & u_{i,j} > +V_{ref} \end{cases} \tag{6}$$

A portion of a neural network to solve the Laplacian partial differential equation for $v_{i,j}$ ($\sigma_x = \sigma_y = 0$, uniform cost function) is shown using an operational amplifier implementation in Figure 1. The entire search space is constructed by replicating this portion of the neural network over the entire grid array of the search space. The output of the source and sink neurons which correspond to the start and goal nodes are clamped to $+V_{ref}$ and $-V_{ref}$, respectively. Since the clamping of the source and sink to the maximum and minimum (respectively) potential value occurs on the boundary of the problem, it is appropriate to use these clamped values as the limits of the neuron output function (6), since all potential values inside the boundary of the problem are guaranteed to lie between these limits.

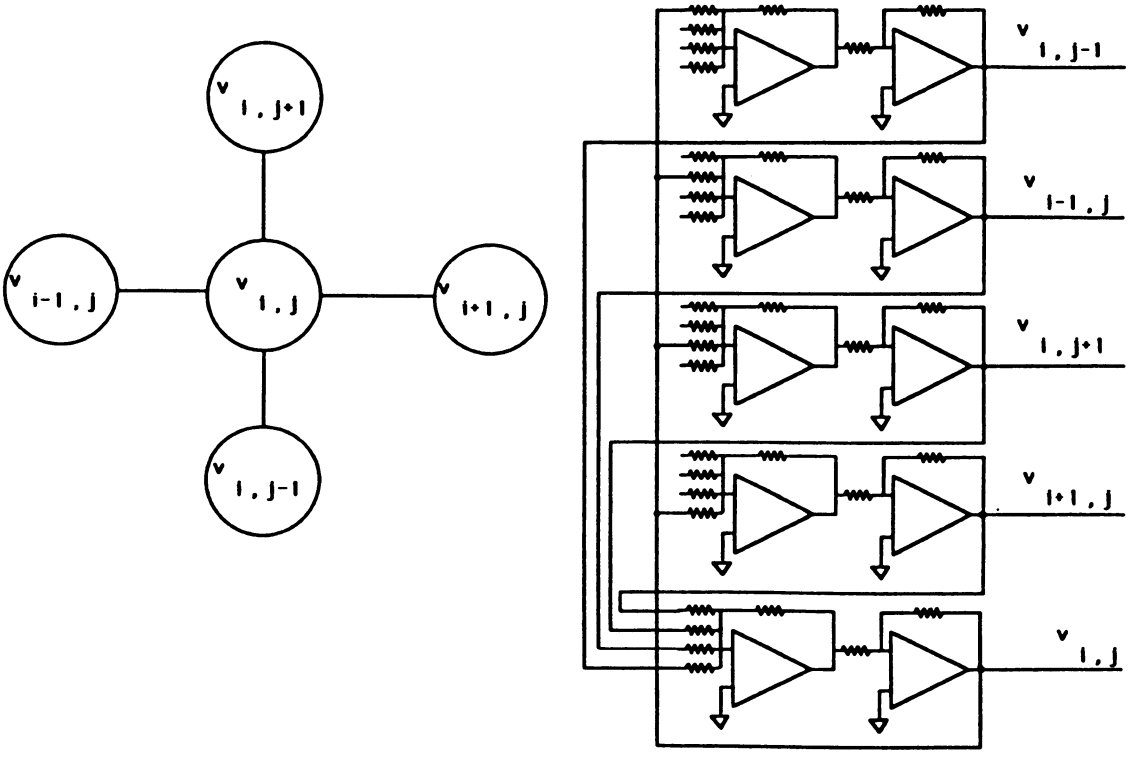


Figure 1: Neural Network Implementation

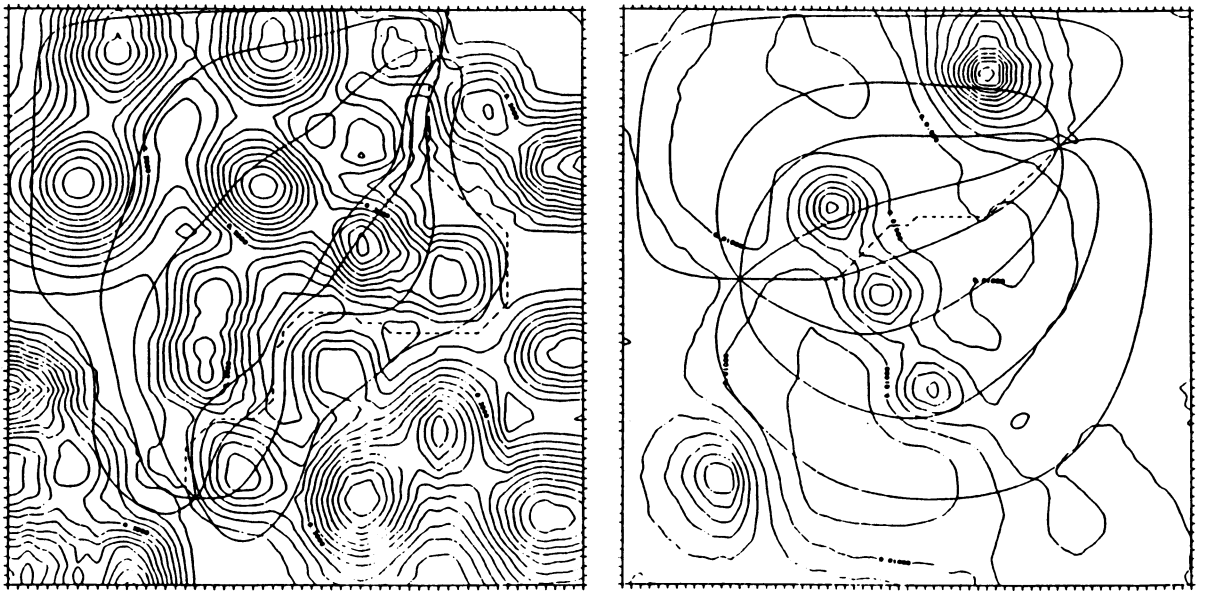


Figure 2: Illustration of Results

EXPERIMENTAL RESULTS & FUTURE DIRECTION

The plot in Figure 2 illustrates the results of experiments. The cost function is displayed using contour lines. The dashed curve is the reference optimal path which was computed using best-first search. The remaining curves connecting the start node to the goal node are the field lines derived from the gradient descent on the potential field computed by the neural network. By inspection, the field lines can be seen to have a reasonable location about the reference path. The reference path is entirely contained within a region which is bounded by a pair of field lines. Several portions of the reference path are closely approximated by the field lines. Both of these examples were computed using a five point approximation, thus each neuron had inputs from its north, south, east, and west neighbors. The network for the plot on the left contains 3600 neurons, while the one on the right contains 4225 neurons. The reference path for the left example had a value of 5.343, while the minimal field line at one degree resolution occurred at a 96 degree heading with a value of 7.678. For the right example, the reference path value was 0.4495, and the 90 degree field line value was 0.4444 which represents a 1.1% error.

Future work on this model is to add additional network layers to compute the gradient of the scalar field. Work will begin on a VLSI implementation of a prototype chip. The paths obtained by this model are reasonable, but model considerations based on the calculus of variations should be studied to determine if another partial differential equation may provide better results. With an analog VLSI implementation, rapid settling times occur and super real time predictions using dynamic costs would be possible.

References

[1] Lothar Collatz. *The Numerical Treatment of Differential Equations.* Springer-Verlag, New York, NY, third edition, 1966.

[2] George E. Forsythe and Wolfgang R. Wasow. *Finite-Difference Methods for Partial Differential Equations.* John Wiley & Sons, Inc., New York, NY, 1960.

[3] J.A.B. Fortes and B.W. Wah. Systolic arrays -- from concept to implementation. *Computer*, 20(7):12-17, 1987.

[4] John J. Hopfield and David W. Tank. Computing with neural circuits: a model. *Science*, 233:625-633, 1985.

[5] John J. Hopfield and David W. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52:141-152, 1985.

[6] Leon Lapidus and George F. Pinder. *Numerical Solution of Partial Differential Equations in Science and Engineering.* John Wiley & Sons, Inc., New York, NY, 1982.

[7] A. R. Mitchell and D. F. Griffiths. *The Finite Difference Method in Partial Differential Equations.* John Wiley & Sons, Inc., New York, NY, 1980.

[8] Allen Nussbaum. *ELECTROMAGNETIC THEORY for Engineers and Scientists.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1965.

[9] M.D. Olinger and M.W. Bird. *Tactical Flight Management Phase 2 Final Report.* Technical Report ID-03R-0484, Lear Siegler, Inc. Instrument Division, 4141 Eastern Avenue, S.E., Grand Rapids, MI, 49518, 1984. LSI/ID is now the SLI Avionic Systems Corp. of Smiths Industries.

[10] M.D. Olinger and M.W. Bird. Tactical flight management threat penetration design. In *Proceedings of the 1984 NAECON*, 1984.

[11] Lyle A. Reibling. The conjectured analogy of the theory of potential fields to parallel optimal path search. March 1989. A brief review of the conjecture and derivation of the nonuniform potential equation.

[12] Lyle A. Reibling. *A Neural Network Architecture for Parallel Search of Multiple Paths.* PhD thesis, Michigan State University, April 1990. In preparation.

[13] Ernst Weber. *Electromagnetic Fields.* Volume I -- Mapping of Fields, John Wiley & Sons, Inc., New York, NY, 1950.

NEURAL NETWORK MODELS AND THEIR APPLICATION TO THE VUV AND OPTICAL SPECTROSCOPY OF MOLECULAR SYSTEMS

Kresimir Rupnik
Department of Chemistry, Louisiana State University
Baton Rouge, LA 70803

ABSTRACT

We are investigating possible applications of Neural Network models to the VUV and optical spectroscopy of molecular (electronic and nuclear) systems. The principal goal of the study is to examine the information content of series observed in the spectroscopy of the excited electronic (Rydberg) transitions as well as of the spectral structures (rotational-vibrational) in the optical (Raman) spectroscopy as a function of the spatio-temporal characteristics of the excitation fields. A model for the calculation of the continuous-time sustained momentum gain or momentum transfer to molecules from external fields is being developed. The study of the nonlinear systems in which the coupling of nuclear and electronic moments takes place in the presence of the external fields is also important for better understanding of the mechanisms of biological information processing and possible applications in the engineering of neural and optical computers and Very Large Systems of Neural Networks.

INTRODUCTION

The physical characterization of biological systems, as provided by atomic and molecular properties, must describe the biological structure and the biological functions relative to the natural environment of the biological systems. Biological molecules are particularly sensitive to perturbations due to the interactions with electric and magnetic fields. Absorption and Raman spectroscopy of atomic and molecular systems in the optical and Vacuum Ultraviolet (VUV) spectral regions, provide important information about the molecular structures and possible biophysical or photophysical processes (e.g. photosynthesis, photomovement, photosensitization). In order to characterize these processes and structures in a proper manner it is necessary to apply a "structured" perturbation to the system and analyze the induced changes which appear in the molecular spectrum. These measurements, in which time-dependent changes of the polarization of the electric and magnetic fields occur, are referred to as polarized light spectroscopy. A major advantage of polarized light spectroscopy is the precision with which the perturbation can be applied. Experimental investigations of polarized light spectroscopy of small molecular systems and of some larger biological materials are requisite to a proper interpretation of parallel distributive processes involved in the momentum (and energy) transfer under the influence of external fields. In Circular Dichroism spectroscopy (CD), for example, one measures the difference in absorption between left and right circularly polarized light caused by the different torques exerted on the molecular constituents by the electromagnetic field. Consequently, the measurement is dependent on the molecular structure as well as on the changes that occur during interactions with the field, which is a continuous-time process sustained by a characteristic input pattern (of traveling waves, for example) and the response of the molecular system. In Magnetic Circular Dichroism (MCD) measurements the two forces, circulary polarized light and the magnetic field, are coupled. Since there is a coupling of a number of constituent parts in all time-dependent interactions, the measurements are expected to show explicit nonlinear behavior. From that viewpoint, Rydberg progressions are clearly one of the most important subjects for the study of nonlinear effects since they span broad and relatively well-resolved patterns in the frequency space. The information content of these spectra is related both to the spatial and temporal

characteristics of the processes. Recent spectroscopic investigations of the effects of the strong magnetic fields and circulary polarized light on atomic[1] and molecular[2] Rydberg transitions and of the effects of perturbers on molecular Rydberg transitions at relatively high number density[3] have given a new impetus to the study of the nonlinear mechanisms of electronic excitations. We have begun investigations of the VUV absorption and MCD spectrum of the HI molecule[4]. The measured spectra of that molecule is of particular interest since it represents a study of excitation of a simple AB type molecule that exhibits resolved rotational and vibrational electronic (Rydberg) absorption structure. This is the first attempt of interpretation of the MCD measurements of that molecule and a challenge that will test the proposed theoretical models.

NEURAL NETWORK MODELS AND THEIR APPLICATION TO THE SPECTROSCOPY

The studies of the response of the molecular systems to the external fields, as measured in optical and VUV spectroscopy, provide a new framework for application of Neural Network models. A possible phase-space approach to the calculation of the Rydberg constant has been discussed recently in the framework of a study of nonlinear dynamics of atoms and molecules in highly excited states[5]. One of the principal goals of this study is to examine the information content of optical and VUV atomic and molecular spectra as a function of the spatio-temporal characteristics (structures) of the excitation fields. Here, the spectral features are calculated using the algorithms that evaluate the momentum (energy) transfer through continuous-time momentum gain (transfer) mechanisms, as the result of the interactions between the molecular and field moments. These processes can be described by a non-linear function of a neutral network system[6]. A program "CORRECT" for the calculation of the momentum gain (transfer) is written for the tabletop computer (in ObjectLogo for MacPlus or MacII). The program contains a simple algorithm (primitive) for a simple momentum gain and loss (XG(t)) and XL(t)) in time t, from/to an external field. It can be represented by the equations that are generally written in the form:

$$XL(t+1) = (1-a/N)\ XL(t) \quad [1]$$

$$XG(t+1) = (1+a/N)\ XG(t) \quad [2]$$

where the parameter a = .45158 and N is the number of steps.. In the case of Rydberg transitions it is shown[5] that the parameter R, from the expression:

$$a = -(1-R/2) \quad [3]$$

gives the value R in agreement with the measured Rydberg constant[7]. The agreement is explained in the framework of a continuous-time phase-space model, where the parameter R enters the formula:

$$XG(t+1) = (1 + (R/N)/N)\ XG(t) \quad [4]$$

which leads to the accurate representation of the Rydberg transitions[8]:

$$d(\nu)/(cm^{-1}*10^{-5}) = R/(1/M^2-1/N^2) \quad [5]$$

for different numbers N,M (1,2,3...). The expressions (1), (2) and (5) indicate that a consistent transcription of the various different approaches to the study of Rydberg transitions (such as the Quantum Defect Theory (QDT)[9], for example), is possible in the Neural Network models. It should be mentioned that the algorithm for the Rydberg series (5) represents the general pattern of the measured series. The formula (5) relates to the so called n-quantum numbers. The development of the relations for the other quantum numbers (1,m,...) will be given elsewhere.

In addition to the study of Rydberg series we have also investigated the application of the Neural Network models to the rotational-vibrational spectral structures. If the momentum gain (transfer) relations (1) (2) and (3) for the combinations of the Stokes and Antistokes transitions are written[5] in the form:

$$XG(N+1) = (1+(R/2)/N)\ XG(N) \quad [6]$$

they lead to the relation that can give the rate between some of the successive[10] frequencies X,Y of the Raman vibrational structures. Here, we treat the rotational envelopes as the "perturbation" of the vibrational structures. Our calculation gives the value k = 0.4326 for the ratio between the measured frequencies X/Y. The correlation between the measured experimental values of the successive vibrational frequencies in optical (Raman) spectroscopy for a number of small molecules gives the value k(experimental) = 0.435. This result is in excellent agreement with the value calculated by the momentum transfer model. We have also measured some vibrational spectra (for molecules like CH3I and CD3I) on our Raman system (Coherent Inova Ar-ion (20W) laser, ISA-1000 spectrometer and IBM XT computer with data acquisition system). The rate of the observed vibrational frequencies for these molecules gives the parameter k (experimental)in the range between 0.42 and 0.46. This work is in progress.

In conclusion, a brief description of the work done on the application of the neural network models into the VUV and optical spectroscopy is given. We do believe that the interpretation of the spectral structures in the optical and VUV regions could be given in terms of the Neural Network models for the calculation of the momentum gain (transfer) from the external fields, even for more complex molecular systems.

References

1. J.C. Gay in "Atoms in Unusual Situations", p. 107, J.P. Briand, Ed., NATO ASI, Plenum, New York (1986).

2. S. P. McGlynn, J.D. Scott, and W.S. Felps, J de Phys. (Paris) 43, C2, 305 (1982).

3. U. Asaf, W.S. Felps, K. Rupnik, S.P. McGlynn and G. Ascarelli, to be published in J. Chem. Phys.

4. K. Rupnik, W.S. Felps and S. P. McGlynn, unpublished results.

5. K. Rupnik, "Phase Space Approach to the Calculation of Rydberg Constant", Optical Society of America 1989", Washington D.C. 1989

6. For definition of the Neural Network Systems see: IEEE International Conference On Neural Networks" Vol 1. IEEE, New York, 1987.

7. The value of the Rydberg constant calculated from the hydrogen series is Ry = 1.0968, see also P. Zhao, W. Lichten, H. P. Layer and J.C. Bergquist, in "Laser Spectroscopy VIII", p. 12 W. Persson and S. Svanberg Ed., Springer-Verlag, New York, 1987.

8. The experimentally observed regularity of the hydrogen series was first mathematically formulated by Balmer. The equation for frequency is given in the form of eq (5).

9. U. Fano, Phys. Rev. 124, 1866 (1961).

10. As an example, in the case of CH_3I and CD_3I we have compared the rates

between the frequencies of the vibrational modes 1 2 and 3. A precise description of the assignment of the vibrational frequencies for all molecules taken into account will be given elsewhere.

11. A. Weber, "Raman Spectroscopy of Gases and Liquids" Springer Verlag, New York 1979.

Principles of Sequential Feature Maps in Multi-level Problems

Jagath K. Samarabandu and Oleg G. Jakubowicz

Dept. of Electrical and Computer Engineering
State University of New York at Buffalo, Amherst, NY 14260
bandu@cs.buffalo.edu, jakubowi@cs.buffalo.edu

Abstract

A neural network architecture based on self-organizing feature maps which is suitable for interpreting sequential information is presented together with how it can be applied as a building block for multi-level systems. The network is applied to the case role assignment problem in sentence understanding and the results shows good generalizing and asociation capabilities coupled with the ability to regenerate missing or empty features in unfamiliar input words.

1 Introduction

Interpreting sequential information is a part of most cognitive tasks such as speech recognition, natural language processing etc. In this paper, we will present a general purpose neural network architecture for processing sequential information using self-organizing feature maps [1] that can be applied to multi-layer systems. We will also show the application of these neural networks to the problem of speech perception.

At the lowest level, speech perception consists of phoneme classification from the acoustic signal. The next higher level can be the recognition of words from phonemes. Subsequent levels can be formed as assigning case roles to the input words (local context), story understanding (modeling global context) etc.

In the past, researchers have applied various neural network models to each of these levels and it is our intention to present some general principles that can be applied to this particular problem.

At the phoneme classification level, McClelland & Elman [2] uses the interactive activation model in their *Trace I* model and Kohonen [3] uses self-organizing feature maps in his *phonetic typewriter* to recognize phonemes.

At the word level, McClelland and Kawamoto [4] uses a distributed representation (**semantic microfeature encoding**) where each word is classified according to a set of features (dimensions). Each feature (dimension) is assigned a node and the representation is in the form of a pattern of activation over the set of nodes.

Another method proposed by Miikkulainen and Dyer [5] exploits the formation of internal representation in the hidden layers of a back propagation network by extending the error signal to the input layer and developing representations of input data at the input layer.

The notion of microfeatures are applied at a higher level in word sense disambiguation by Bookman [6] in which each sentence modifies a set of microfeatures representing global context and these microfeatures in turn affects the interpretation of subsequent sentences.

2 System Description

2.1 Basic Architecture

The basic neural network architecture can be considered as having three layers (figure 1). i.e. an input layer, a topographically ordered intermediate layer and an output layer.

2.1.1 Propagation

Propagation equations are as follows.

$$a_i(t) = \sum_{j=0}^{l} x_i(t).m_{ij} \quad + \theta.O_i(t-1) \tag{1}$$

$$O_i(t) = f(a_i(t)) \tag{2}$$

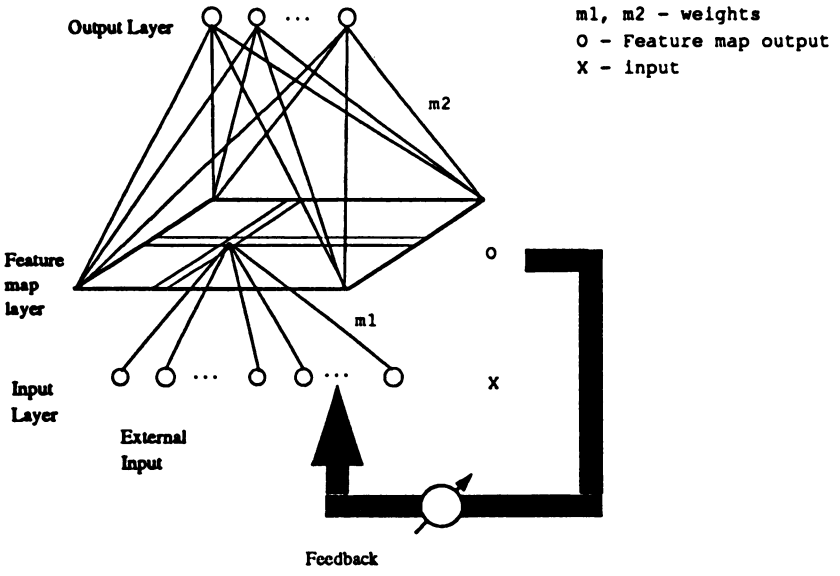


Figure 1 Basic Neural Network Architecture

Where

O_i — feature map output
$f(x)$ — sigmoidal output function
θ — hysterisis

Propagation from the feature map layer to the output is similar to the above equation except that θ is zero. Input is propagated to the feature map layer for every input where as the feature map layer is propagated to the output only at the end of an input sequence. We have successfully implemented this model in robust word identification problems

2.1.2 Learning

Learning for the feature map occures in two steps. During initial learning, feedback from the feature map is disabled to allow context free learning. Subsequent enabling of feedback allows learning input with the context. Standard Kohonen feature map learning rules are used to learn m1 and m2 except that the output layer utilizes supervised learning and hence does not need to find the node with the minimum distance.

The main difference between the above network and Kohonen self-organizing feature maps is that the feature map retains output during a sequence of input vectors and that there is time delayed feedback from the feature map output to the input. This provides the ability to extract contextual nformation as well as learning time varying sequences..

2.2 Application

The above model can be applied to the task of speech understanding as shown in figure 2

Each level of the system can be implemented using the proposed neural network architecture. In particular, we have applied the network for case role assignment in which each word is represented by 47 microfeatures and outputs the microfeatures for each of the five case roles.

First, the feature map layer was trained with the given vocabulary. Once the topological ordering of the input vocabulary is learned, the output layer was trained with the input sentences. This was accomplished by first calculating the feature map output using equations (1) and (2) for the entire sequence (assuming $O_i(0) = 0$) and then training the five micro feature vectors for agent, action, patient, instrument and modifier with the corresponding input micro feature vectors as teaching patterns. (figure 3 (b))

A secondary supervised learning mechanism is also incorporated such that it learns both the microfeature representation of words in the lexicon and the weights between the output layer and the feature map of the neural network if the output microfeatures substantially differ from those of corresponding input word (as in the case of

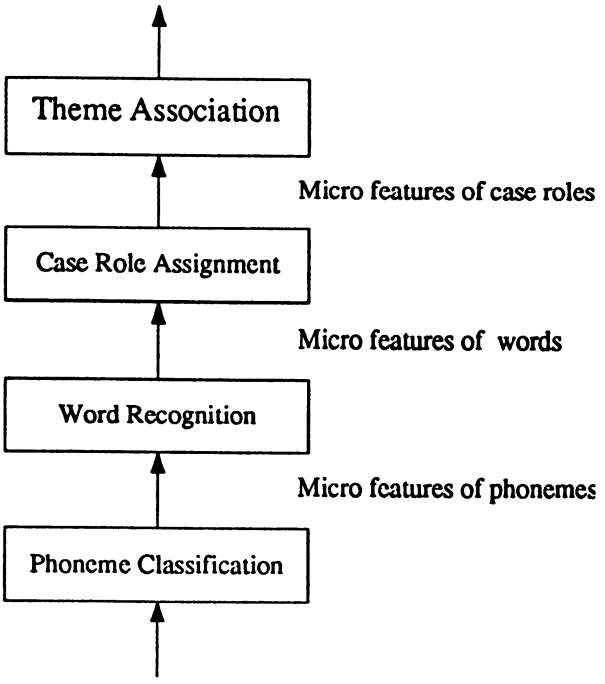


Figure 2 System Block Diagram

unknown words). The new microfeatures are learned at a rate that is proportional to the number of times that word has been encountered. Thus, the course features are learned quickly for a new word and fine distinctions occur slowly when that word occurs in different contexts.

2.2.1 Performance

The system was initially trained with a vocabulary of 28 words (feature map size was 10 x 10) and the output layer was trained with 120 sentences. The vocabulary, sentences and the case roles were adapted from [4] in order to make the comparison easier. Sentences from the training set gave 100% recall (despite the fact that there were three overlapping words on the feature map). When tested with unfamiliar sentences (known words) 81% were identified correctly.

For unknown words (blank micro feature templates) the system selected the most frequent training sentence form, i.e. the training set contained 5 sentences of the form *boy ate <food>* out of 8 *boy ate <food,instrument, modifier>*. Thus the sentencs *boy ate <new_word>* interpreted the *<new_word>* as food.

Capacity of the feature map was tested using three different feature map sizes and the results show the degradation of the learning ability with increasing vocabulary size and decreasing feature map size(figure 3 (a)).

3 Future work

The main thrust for futher work is in applying the network to other levels of speech perception. At the phoneme level, the network will accept acoustic signal as the input and outputs either a distributed or local representation of phonemes and at the word level, outputs the microfeatures of the words. At a more higher level, information about case role assignment can be used to produce a global context in understanding a script.

4 Conclusion

We have proposed a neural network architecture based on self-organizing feature maps which can be used as a building block for complex cognitive tasks involving mutiple levels. With the modified Kohonen feature map, the sytem is capable of handling sequential information and shows good performance in handling variable time length sequences. Addition of an output layer makes it easy to transform the feature map output to the desired form, making it easy to use in a network with several levels. Application of the system in case role assignment shows

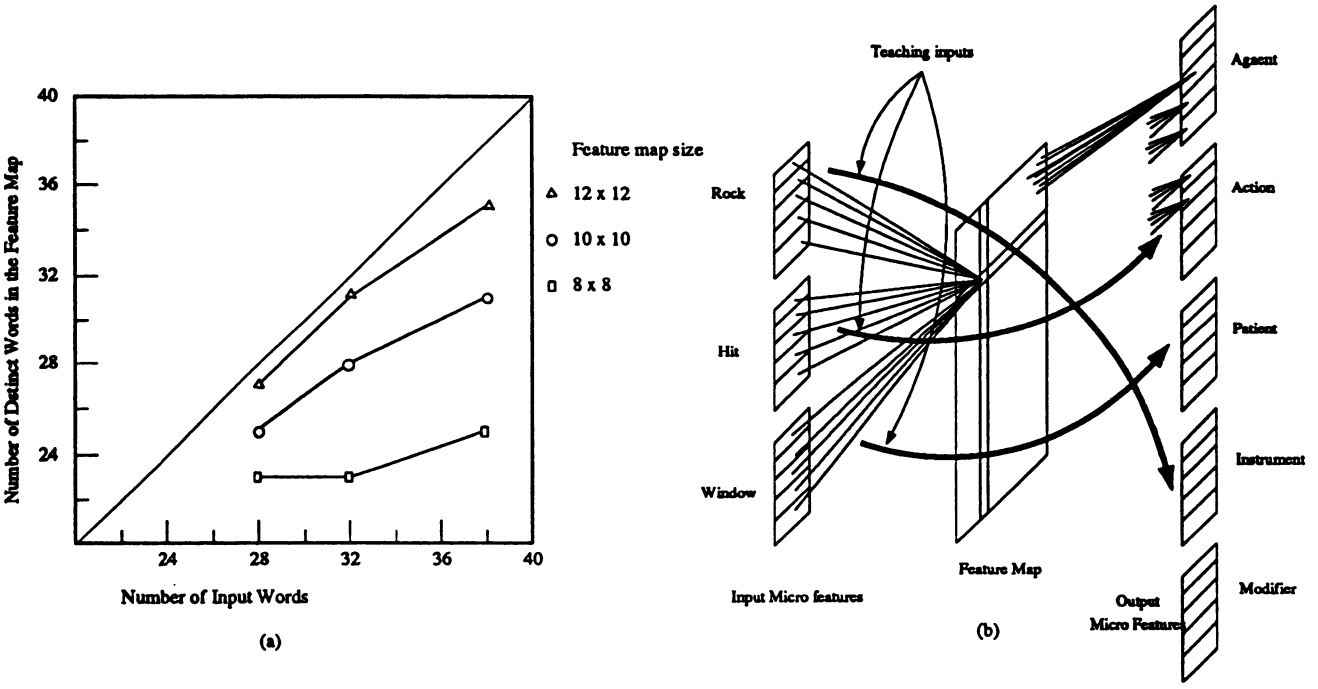


Figure 3
(a) Number of distinct words learned as a function ot the size of the feature map
(b) Case Role Assignment system

its ability of forming good associations and generalizations. Further, the system is able to learn the coarse features of unknown inputs quickly and to learn the fine features slowly over time. This ability was adopted to enable the system to learn while testing, thus gathering "experience" from each test session.

References

[1] Teuvo Kohonen, Self Organization and Associative Memories, chapter 5, Springer Verlag, Berlin, New York, 1985

[2] J. L. McClelland and J. L. Elman, Interactive processes in speech perception: The Trace Model in J. L. McClelland and D. E. Rumelhart. editors, PDP Vol. II, MIT Press

[3] J. L. McClelland and A. H. Kawamoto, Mechanisms of sentence processing: assigning roles to constituents, in J. L. McClelland and D. E. Rumelhart. editors, PDP Vol. II, MIT Press

[4] Teuvo Kohonen, Phonetic Typewriter, IEEE ASSP Magazine 1988

[5] Risto Miikkulainen and Michel G. Dyer, Encoding Input/output representation in connectionist cognitive systems, Proc. Rochester Connectionist summer school 1988

[6] Lawrence A. Bookerman, A connectionist scheme for modelling context., Proc. Rochester Connectionist summer school 1988

NEURAL NETS vs. ANALOG COMPUTERS
AN OBSERVATION

Gursel Serpen and David L. Livingston
Dept. of Electrical and Computer Engineering
Old Dominion University
Norfolk, VA 23508

ABSTRACT

The computation performed by an analog computer to solve partial differential equations numerically is studied from the viewpoint of a Hopfield neural network that optimizes a quadratic performance index. Energy equations for a Hopfield network that will solve Laplace's and Poisson's equations are defined by using the constraints that must be met for a numerical solution of both partial differential equations. The connection weights and the external bias terms are generated by comparing the standard Hopfield net energy equation and the proposed energy equations for solving Laplace's and Poisson's equations. It is also shown that the minima of the proposed energy equations correspond to solutions of Laplace's and Poisson's equations.

INTRODUCTION

The goal of this paper is to show that spatial solutions of certain classes of partial differential equations (pde's) can be computed by Hopfield networks with linear activation functions. An analog computer configured to solve pde's may be viewed as a special case of a Hopfield network with linear activation functions.

Analog computers with identical computing elements have long been used to solve certain classes of pde's [6]. For example, to solve Laplace's equation spatially, each computation element might consist of an adder and an integrator configured as shown in Figure 1. The governing dynamics of the computation elements are given by the following equations:

$$du_i/dt = -u_i + (1/\Psi)\sum_{j\epsilon N} \sigma_{ij} V_j, \text{ and}$$

$$V_i = f(u_i) = u_i, \text{ for } i = 1, 2, \ldots, N; \qquad (1)$$

where σ_{ij} is the topological neighborhood (TN) function defined by

$\sigma_{ij} = 1$, if V_i and V_j are topological neighbors,

$\sigma_{ij} = 0$, otherwise, and

Ψ and N are the number of topological neighbors and the set of computation elements respectively. A network topology to solve Laplace's equation spatially in two dimensions is a two-dimensional grid where each grid point is assigned a computation element [7]. In order to incorporate the boundary conditions in the problem solution, the activations of computation elements along the boundaries are clamped to appropriate boundary values. Initially, all activations are set to zero except the ones representing the boundary values. At $t > 0$ all computation units start to integrate the error which is

the difference between its current output and input. The computation is complete when all activations are equal to the average of the topological neighbor activations.

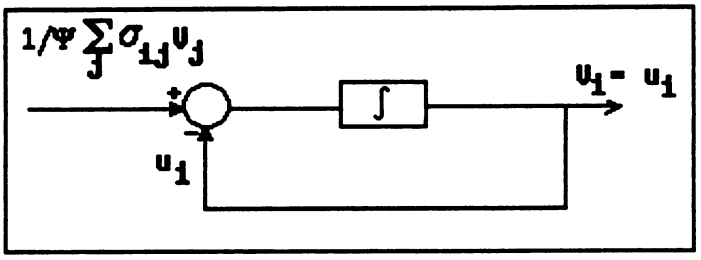


Figure 1. A Computation Element

A HOPFIELD NET AS A PDE SOLVER

System Definition

Let $V_i \epsilon [0,1]$, $\forall i \epsilon N$, where N is the set of activation functions. Hopfield [1-4] shows that

$$E_H = -(1/2) \sum_{i \epsilon N} \sum_{j \epsilon N} T_{ij} V_i V_j + (1/2) R^{-1} \sum_{i \epsilon N} (V_i)^2 + \sum_{i \epsilon N} I_i V_i \qquad (2)$$

is a Liapunov function for the system of equations defined by

$$du_i/dt = -u_i/\tau + \sum_{j \epsilon N} T_{ij} V_j + I_i, \text{ and}$$
$$V_i = f(u_i) = u_i, \qquad \forall i \epsilon N. \qquad (3)$$

Laplace's Equation

The numerical solution of Laplace's equation in a two-dimensional coordinate system satisfies the following equation for each spatial coordinate [5]:

$$V_i = (1/\Psi) \sum_{j \epsilon N} \sigma_{ij} V_j, \qquad \forall i \epsilon N, \qquad (4)$$

where σ_{ij} is the TN function, Ψ is the number of topological neighbors for coordinate element "i" and N is the set of coordinate elements.

Consider the following quantity as an energy equation for a network that will solve Laplace's equation spatially in two dimensions ($\Psi = 4$ for two dimensions).

$$E_L = \sum_{i \epsilon N} \left[V_i - (1/\Psi) \sum_{j \epsilon N} \sigma_{ij} V_j \right]^2. \qquad (5)$$

For a given vector, $\underline{V}$, that is a solution to Laplace's equation, the energy function is equal to zero. This result can easily be observed if Eq. 4 is inserted into Eq. 5. For a vector, $\underline{V}$, that is not a solution, the energy function is greater than zero due to the fact that at least one term in the summation is nonzero and quadratic. Thus the energy function minimum corresponds to the solution of Laplace's equation.

Comparing Eq. 2 with Eq. 5, we can determine that $T_{ij} = (4/\Psi)\sigma_{ij}$ and $I_i = 0$ subject to the constraints that R = 0.25 and

$$V_i = (1/\Psi)\sum_{j\epsilon N} \sigma_{ij}V_j, \qquad \forall i\epsilon N. \tag{6}$$

Note that last constraint simply imposes the condition that the net input sum to i-th node needs to be weighed by $1/\Psi$. We can therefore transform Eq. 5 into the form of Eq. 2 using the above conditions and conclude that Eq. 5 represents a Liapunov function for the network.

Poisson's Equation

The numerical solution of Poisson's equation satisfies the following equation for each spatial coordinate in two-dimensional space [5],

$$2\left[(\Delta x/\Delta y)^2+1\right]V_i - \sum_{j\epsilon N}\Phi_{ij}\sigma_{ij}V_j = -(\Delta x)^2 f(x_i,y_i), \quad \forall i\epsilon N, \tag{7}$$

where x_i, y_i are spatial coordinates of V_i,

Δx, Δy are spatial difference between nodes along x and y axes,

σ_{ij} is the TN function as defined earlier,

$\Phi_{ij} = (\Delta x/\Delta y)^2$, if V_j is a topological neighbor of V_i along y-axis,

$\Phi_{ij} = 1$, if otherwise,

$f(x_i,y_i)$ is the value of source term at coordinates (x_i,y_i), and

N is the set of spatial coordinates in a two-dimensional system.

Let $\alpha = 2\left[(\Delta x/\Delta y)^2+1\right]$, $\beta = (\Delta x)^2$ and $\mu_{ij} = \Phi_{ij}\sigma_{ij}$ then consider the following as an energy function for a network that will solve Poisson's equation:

$$E = \sum_{i\epsilon N}\left[\alpha V_i + \beta f(x_i,y_i) - \sum_{j\epsilon N}\mu_{ij}V_j\right]^2. \tag{8}$$

Consider the case when $\underline{V}$ is the solution to Poisson's equation, then the energy function is equal to zero since every term in the summation is zero. This can be verified by substituting Eq. 7 in Eq. 8. If $\underline{V}$ is not the solution vector, then the energy for this case is greater than zero since the summation term is always positive due to some of the quadratic terms violating Eq. 7. Therefore the energy function has its minimum which is equal to zero for a solution vector.

In order to put Eq. 8 in the form of the standard energy equation, let $E_p = E/\alpha^2$. Defining $T_{ij} = (4/\alpha)\mu_{ij}$ and $I_i = 2(\beta/\alpha) f(x_i, y_i)$ subject to the constraints R = 0.25 and

$$V_i = -(1/\alpha)\left[\beta f(x_i, y_i) - \sum_{j \in N} \mu_{ij} V_j\right], \qquad \forall i \in N, \tag{9}$$

demonstrates that E_p (and hence E) is a Liapunov function for the network. Again, Eq. 9 clearly states that the activation function of the computation node must be linear.

CONCLUSIONS

We have shown that the analog computation performed by an analog computer to numerically solve pde's is stable and convergent to unique solutions of pde's by interpreting the computation of the analog computer from the viewpoint of a constraint satisfaction neural net. We first noted the similarity of analog computer solutions to that of a Hopfield net and then applied the same mathematical tools used for Hopfield net analysis towards establishing computational stability and convergence properties of pde-solver analog computers.

It is worth mentioning some of the problems associated with the representation scheme. The representation of the solution to a pde does not have any inherent redundancy to compensate for the failures of computational elements and hence is not fault tolerant in that sense. Another aspect is that the solution of the pde is a numerical approximation; hence some error is introduced. Also, the representation is very sensitive to noise inherent to the computational elements. Additionally, the activation function of a computational element must be linear, which may be difficult to realize with physical elements.

REFERENCES

[1] Hopfield, J. J., "*Neural networks and physical systems with emergent collective computational abilities.*", Proc. Natl. Acad. Sci., USA, Vol. 79, pp. 2554-2558, April 1982.

[2] Hopfield, J. J., "*Neurons with graded response have collective computational properties like those of two-state neurons.*", Proc. Natl. Acad. Sci., USA, Vol. 81, pp. 3088-3092, May 1984.

[3] Hopfield, J. J. and Tank, D. W., "*Neural Computation of Decisions in Optimization Problems.*", Biological Cybernetics, Vol. 52, pp. 141-152, 1985.

[4] Hopfield, J. J. and Tank, D. W., "*Computing with Neural Circuits: A Model.*", Science, Vol. 233, pp. 625-633, August 1986.

[5] Burden, R. L. and Faires, J. D., Numerical Analysis, Boston: PWS Publishers, 1985.

[6] Johnson, C. L., Analog Computer Techniques, New York: McGraw-Hill Inc, 1956.

[7] Tou, J. T., Advances in Information Systems Sciences, New York: Plenum press, 1969.

NEURAL NETWORK ENHANCEMENT TO TRADITIONAL COMPUTER ENVIRONMENT

Yuri Shestov

Department of Computer Science
MET College, Boston University
Boston, MA 02115

TU Inc.
8460 Tyco rd.
Vienna, VA 22180

Abstract - Neural type connections are introduced in *hypermedia* like memory organization, utilizing parts of traditional file system structure without corrupting the latter. Logical *chunks of memory* (e.g. files, their parts or combinations thereof) are uniquely represented by small and convenient-for-manipulation forms or *MEmory Descriptors*. These descriptors are interconnected (with weighted links) into a large, adaptive *MEmory MAnifold*. This neural hypermedia or *Neural Interactive Paradigm* is an adaptive environment with *computer situation* sensitive functionality where sensitivity is tuned via an *attention span* filter mechanism. This schema allows smooth and effective control of a computational process with more streamlined user-system interaction. It implies a very effective caching procedure.

Memory organization - Typical file structure organizes files into hierarchical directories with little consideration of logical, situation or procedure dependent closeness of files or adaptability to different users or repetitive computational patterns. Symbolic means of referencing the files are the corresponding file names. Here we introduce dynamically adaptable, computer ***system situation*** dependent memory organization. System situation describes the system in terms of the activity of its memory.

Memory formation, typically a file, is pointed to from a small template like object called a MEmory Descriptor or MED. This descriptor provides information on the pointed memory. As a symbolic means of referencing corresponding memory it affords a much richer functionality compare to a mere index information of the traditional file system. MED is a template or a collection of fields from an expandable set of standard information fields of certain types (e.g. memory pointer, date of modification, author, rights, functionality, help pointer etc). It is an object with multiple inheritance from the classes of constituting fields [MTW]. To point to the memory at a finer grade then a file, the file's MED contains additional pertinent fields with corresponding information utilized by appropriate methods.

MEDs are further dynamically interconnected into a more logical or computational process dependent memory structure. Information about links is stored in MED fields. All together, information on the described memory may be efficiently manipulated and utilized by the system or user working with large sets of MEDs and invoking actual memory only when it is indeed required. So far it is just a different type of hypermedia memory organization [BV, CEJ]. However, MEDs memory structure consistently incorporates a native file system plus the descriptor information organized more like Minsky's frame [MM].

This memory organization does not need to violate or change the way of a conventional file system. Rather, it offers a more convenient alternative practice without interference with the former way of operation. This compatibility opens a door to a great number of new applications to be introduced consistently into traditional and dominating computer system environments. Some application areas to consider are knowledge representation (especially the correlation or unification of an accumulated

set of perhaps diverse information), expert systems enhancements.

Now we marry one great idea to another.

Neural structure of memory - In addition, MED incorporates fields describing the neural nature of the environment: threshold, excitation, list of *normalized weighted connections* to other MEDs etc. They are instrumental in shaping a geometry of memory. Geometry here is a dynamic structure depending on changes in actual connections, their weights, and excitations [MWS+, RF]. Topology of the memory is derived from the architecture of actual connections. Long term memory metric is defined by weighting of those connections, MEDs with heavier links are stronger related and thus closer in our memory organization. The short term memory metric is defined via actual levels of excitation of our nodes. At a given time the latter metric depends on the absolute value of the excitations of memory nodes, depicted in the corresponding MED fields. It is presented to the user or system via an attention span filter, showing locally all sufficiently excited memory.

Total collection of all interconnected MEDs is called MEmory MAnifold (MEMA). It may be represented as an arbitrary directed graph with weighted links. MEMA may serve as a more generic memory model to embrace different excitation flows and their interactions, complementing (e.g. allowing smooth incorporation in traditional computing environments) a multitude of particular experiments in neural architectures of formal neurons (e.g. ART, back_prop, Kohonen, Hopfield etc).

MED, as a node, in MEMA is different from a traditional formal neuron. It is not only a part of an architecture, where an individual neuron may not mean much on its own. The MED node carries not only external information through its weighted connections and firing regulations but it also has an internal structure providing information on the functionality and status of memory it represents. Therefore, active and rich functionality, which may be attached to the node, makes a much greater functional complexity then that of a formal neuron. The neural node in the proposed Neural Interactive Paradigm or NIP model is not like an individual neuron but perhaps some neuron assembly responsible for a particular functionality which is in turn in neural ensemble with other functional groups. This schema is not an atomistic one in that smaller constituting blocks need not to be considered of elementary functionality but of open, dynamically evolving one [GS].

One may point rather quickly to discrepancies with organic models, which may be mellowed in refinements of the schema, but we are making no such claims to the correspondence and rather making a technical proposition.

Excitation propagation - Neural excitation in a network of MEDs propagates from active nodes through local connections to the neighbors. In addition to the local excitation propagation via weighted connections there is also a global, related to "long" haul message communication mechanism, where one node sends a message to a group of other nodes (e.g. one to one, broadcast), causing appropriate actions and events e.g. excitation of some MEDs with corresponding propagation of the excitation and perhaps modification of weights (e.g. arriving of mail to user's mailbox).

Attention span plays a significant role in the modification of excitation propagation by selecting resolution levels of a global picture and thus the direction of an excitation propagation.

Attention span - The number of excited or neighboring nodes may be considerable (e.g. a word processor, called by user to action, may excite a large number of files that could be edited with this word processor or a problem of overloading with large fanout hypermedia). Thus, to focus the attention of a system or user on a related subset it is natural to introduce a filter (e.g. a prioritized subset of recently or frequently accessed word processing files by this user or more typical nodes for hypermedia transition). An adjustable attention span is a view comfortable to a user (e.g. a screen or window full of appropriate and affordable information). The only nodes with a certain level of excitation, default or satisfying some other pertinent property get into immediate view and access an area of attention span. Attention span masks sufficiently excited nodes and may compute propagation from this set.

Thus attention span is a mechanism to map a global picture into a local one. This mechanism allows experiments where global conditions affect local modification to excitation propagation. It could be a convenient instrument in determining a mode change or a global shift of behavior.

Navigation and flow of attention in memory manifold - Navigation or transition through MEMA of attention span or locale of activity can be done by a user or system in control or via an *auto transition* both using presented information on a system situation. The last may be done in several ways. For example, one may transit on the heaviest link from the current situation to the next one (we may refer to it as a local or hyper transit). Another way, is to transit to the most excited node in the attention span, perhaps with a construction of a corresponding path plan. In that case, there is a convenient opportunity to identify special cases where a new link may be created. Any auto transition should provide some default time window to override it.

In effect, the system offers and may enact reasonable action based on the combination of experience stored in LTM, current information on the situation from STM and conventional information provided by MED fields. This mechanism enables a potentially very intricate response of the system to accumulated information relating to the specific situation. Therefore it provides architecture for sophisticated interaction or "control" in complex multimode environments, where global information can be taken into consideration.

Learning - Learning in the system occurs through interaction with it. Modification of weights perhaps to be done when a desired or actually made transition differs from the one suggested by the system. That situation may be easily detected and the learning subsystem is alerted. That is when a user or system forces transition to a particular situation and then, if it is satisfactory (e.g. when some process or underlying memory engaged), issues a command to learn. At that time the weights for the appropriately computed short path will be modified and weights of pertinent links renormalized or even connection architecture may be changed.

Implementation - Software, reflecting this type of architecture, is implemented as Hyper Information Manager (HIM) in the Neural Interactive Paradigm[1]. It provides adaptive, situation sensitive (e.g. user sensitive) interaction with the computer system. Standard primitive types of files that the application allows to connect are executables, text and graphics.

We adopted a Markov chain as an restricted but easy to implement model for excitation propagation. Excitations and weights are real numbers from the [0,1]. A particular computer *system situation* may be represented as a distribution pattern of activity of the system memory or excitation of MEDs in MEMA. Thus it is a vector (w_i) from $[0,1]^n$, where w_i is a probability of MED node i to be active and n is a total number of nodes in the system. Weighted connections are represented by the transition matrix (w_{ij}) of n*n size, where w_{ij} is interpreted as a probability of transition of activity from node i to j. One step of propagation in our model is equivalent to multiplication of a state vector (w_i) by the transition matrix (w_{ij}). Dynamic behavior of that system is well known [FW]. We fuzzy the system up by computing propagation from filtered and normalized attention span mask.

The system starts up in a special junction node ROOT (as parameter, we may specify user bias node) being active. From here system starts to propagate to the neighbors and fills in the attention span window (input or perception), presenting more likely choices to transit. Evaluation of the most excited node in the attention span determines the next most likely activation site or/and transition path (processing or cognition). The auto navigation system then moves to this new active node and/or does other appropriate actions (output or action). A user may always preempt and correct the transition of the system, leading it to any desired computer situation (fact of correction is again considered as an input or perception).

The lack of success in auto navigation is marked by the user's change in the learned flow with consequent engagement of the memory for base functionality. At that point

[1] This implementation is done under a contract with TU Inc.

recomputation of weights message is issued, shortest path computed and corresponding weights are modified and renormalized (learning or adaptation). The cycle of perception, cognition and action is repeated.

Comments in conclusion - The neural dynamics already implemented in HIM are rather trivial but sufficient to illustrate the potency of the idea with a simple but effectively useful application, and thus invite further exploration. We are currently working on Unix version of HIM, and some variations on neural dynamics in MEMA. Mathematical foundation for this kind of memory organization is being developed [SY]. The whole schema is wide open for different implementations and experimentations.

Proposed memory organization enables the following theoretical and practical implications:

streamlining interaction with complex systems, utilizing the experience of experts: through learning or compressing information of the "typical" computational flow and memory organization of the experts and suggesting those choices to a less experienced user

emergence of naturally intelligent behavior of NIP system: reasonable anticipation and response through a "look ahead" for some most likely computational behavior recorded in the learned weights from previous experience

efficient cache memory procedure: by bringing in the memory logical pieces which are more likely to be used next, via monitoring excitation values and perhaps their rate of change through appropriate filters

natural paradigm for parallel and distributed computation: by automatic invocation of processes based on the appropriate level of excitations in the environment of these processes, and information on the most likely future for distributed computational flow.

productive way to neurocomputer architecture: by a mutually advantageous relation with a rich and fertile world of classic computing on von Neumann machines through the incorporation of the existing computing structures instead of building them from scratch.

This paradigm offers an effective and natural mechanism to interact with (e.g. manage) great volumes of information by compressing it, or making it handy (localized and filtered) on situation dependent demand, through "on the job" training. That fact is very timely and important in lieu of the advent of very powerful machines and the rising information management bottleneck: gigabytes of interrelated information on line we would like to handle. Here we offer an example of some appropriate tools for the task.

References

[BV] Vannevar Bush, *As we may think.* Atlantic Monthly 176, 1, 101-108, 1945

[CEJ] Conklin, E.J., *Hypertext: An introduction and a survey.* IEEE Computer 2, 9 (Sept. 1987), 17-41.

[MTW+] Thomas W. Malone at al., *Intelligent Information Sharing Systems.* Communications of ACM, 390-402, May 1987

[MM] Minsky, Marvin, *A Framework For Representing Knowledge.* In Mind Design, 95-128, edited by J. Haugeland, The MIT Press, 1981

[MWS+] Warren S. McCulloch and Walter Pitts, *A logical calculus of the ideas immanent in nervous activity.* Bulletin of Mathematical Biophysics 5:115-133, 1943

[RF] Rosenblatt, F. *The perceptron, a probabilistic model for information storage and organization in the brain.* Psychological Review 62: 559ff, 1958

[GS] Stephen Grossberg, *Neural Networks research: from personal perspective.* Electronic Engineering Times, March 7: A12-A40, 1988

[SY] Yuri Shestov, *Considerations on Local Invariant of Functionality as a generic memory schema.* draft, 1988

[FW] William Feller, *An Introduction to Probability Theory and Its Applications.* vol.1, ch. XVff, John Wiley & Sons, 1957

A Method for Neural Network Based Melody Harmonizing

Naoki Shibata, Hideo Shimazu and Yosuke Takashima
C&C Information Technology Research Laboratories
NEC Corporation

Abstract

Several automatic music arrangement systems have been developed with the pervasiveness of MIDI[1] controlled music instruments. Harmonization modules usually used in such systems are based on rules. Both AI and music knowledge are necessary for describing harmonization rules. Moreover, each music *genre* requires different rule sets. This paper describes an alternative idea using associative memory, a neural network based melody harmonizing method whose network can be trained by examples. It also presents the results of a subjective evaluation for the accompaniments obtained by the personal computer based music arrangement system equipped with the module based on this method. As the result of the evaluation, the module based on this method without making any rules generates as natural chord progressions as a transition table driven harmonization module does.

1. Introduction

In accordance with the development of digital controlled musical instruments, especially music synthesizers, several automatic music arrangement systems has been developed. Such an automatic arrangement systems are constructed on an assumption of existence of tonality, and includes chordal harmonization module [1] [3].

In some popular music, like modern jazz, harmonizations are symbolized by chord progressions, usually indicated by chord symbols such as Am7-5, Cmaj7,C+, etc. However, there is no established harmonization theory which provides *one-and-only* chord progression for given melody, and this lack causes rule based machine harmonization approach difficult to generate natural chord progressions. For example, if a user or a knowledge engeneer is to improve those rules, s/he may avoid inconsistencies which are productions of ambiguous rules by eliminating some rules and may suffer unnatural progressions caused by those eliminations. Some systems avoid this problem by showing possible progressions and leaving selection to user, others use transition rules [1][3] which are sometimes mapped into the tables.

In this paper, a rule description free associative memory based harmonization method is described. The results of the subjective evaluation for the musical performance outputs obtained by the music arrangement system [1] equipped with this method based harmonization module is also reported.

2. An Overview of Automatic Music Arrangement System

Fujii et al.[1] developed a music arrangement system based on hummed melodies. The system uses a 16-bits personal computer (NEC PC-9801 series), a digital signal processor, an A/D converter, a microphone and music synthesizers . This system consists of four modules; a transcription module, a chord generation module, a melody analysis module and an arrangement module(see

[1]Musical Instrument Digital Interface

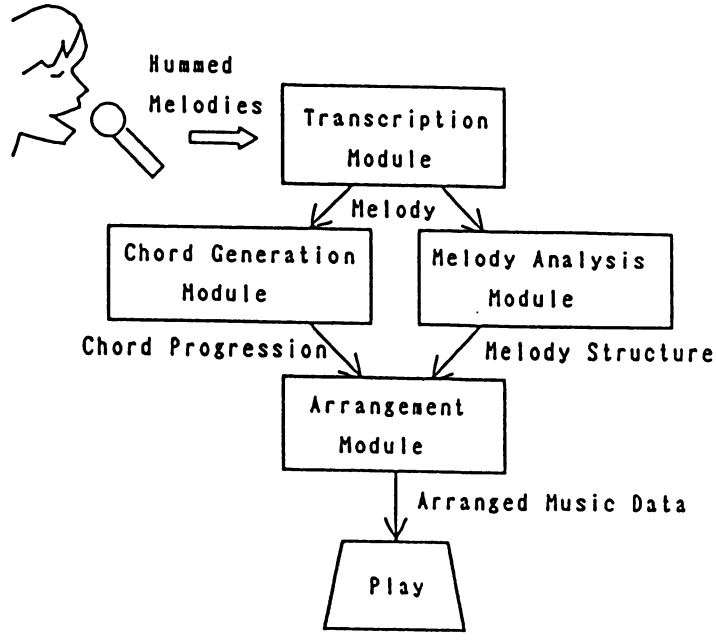


Figure 1: Four Modules of the Music Arrangement System

Fig.1). Hummed melody is transcribed into musical notation in the transcription module. Then results of the transcription are passed to two modules which provide chord progression and structure of the melody line. The melody analysis module divides the entire melody into relatively small melody blocks, and determines relations among blocks as the structure of the entire melody. The arrangement module generates an accompaniment, an introduction, an intermission and an ending using transcribed melody and generated chord progression.

The chord generation module which provides chord progression selects candidate chords every half measure by examining the melody line. Then fittest chord among the candidates is selected for each half measure according to a chord transition table. This method is referred as the transition table method in later chapters.

In this table-driven system, what most affects chord progression is the chord transition table in the chord generation module. However, chord progressions generated by using such table are sometimes unnatural, because the transition table are made by considering not about the really existing melodies and chord progressions but only about the possibilities of adjacent chord transitions.

The maintenance of such transition table is difficult because of ambiguity of harmonization. So the easy maintenance for the transition table emerged as a first motivation. Music harmonization is a kind of pattern association. If that association is to be self-organized by only showing examples, maintenance of rule set is replaced with selecting typical examples of melody and chord progression.

3. Chord Association from Melody

In most cases of tonal popular music, especially in chordal compositions, basic harmonization is done in the following sequence [4]:

- Step 1 To search where the cadences are.
- Step 2 To search most important note in each bar.
- Step 3 To see if such notes forming or suggesting a particular chord.

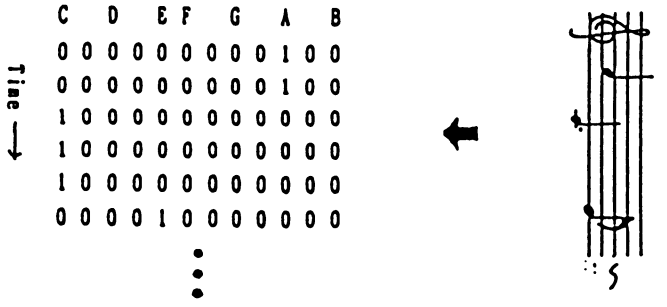


Figure 2: An Example of Input Representation

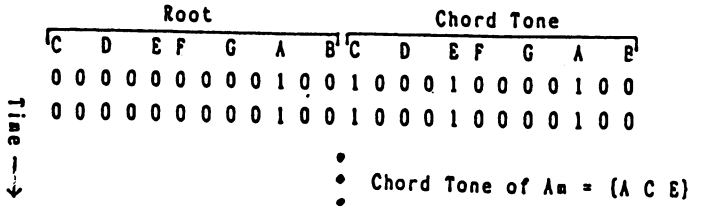


Figure 3: An Example of Output Representation

Step 2 and step 3 suggest the possibility of an associative memory approach for musical harmonization — associating chord progressions from melody/phrases.

And step 1 is an interesting but difficult problem. In our experiments, we restrict music *genre* to Japanese traditional popular songs, *Enka*, whose melody may easily be divided into 4 bars long phrases like Gershwin's standard songs, so that we can easily extract typical chord progressions from *Enka* songs and avoid cadence searching problem. The next chapter describes a neural network based harmonizing method.

4. Network Architecture and Representations

A three layers feed forward network is used for associative memory model. This network was *disciplined* to recall chord progressions from input melody using error back propagation learning method [2].

Input melodies, transcribed to MIDI code in the transcription module discussed in chapter 2, must be translated into the representation seemed fit for such networks. According to the solution for step 1 and network architecture, such a representation must have indices for note positions and pitch indices, say timing-index and pitch-index respectively (see Fig.2). In this simulation, timing resolution was set to an eighth note, and pitch resolution to twelve tone, so that network requires 384 ($= 8notes \times 4bars \times 12tones$) units for input layer.

Chord progression representation was decided to use chord tone. Figure 3 shows an example of chord progression representation. In this example, one chord is represented by 24 units, 12 for chord tone and 12 for root. As for pitches considered in both input and output representations, octave-equivalent pitches category is represented by one unit. We gave two chords per a bar in the implementation, so the output layer requires 192($= 2chords \times 4bars \times 12tones \times 2$) units . Note that representations like one unit per one chord symbol for every harmonic rhythm lead impractical number of output unit if we consider combinations of root notes and triads and tension notes used in chord symbols. Representations like bursting chord symbol into root, triad type and tension

notes literally do not seem appropriate because network using such representations will be tend to give conflicting triads against desirable output.

In this implementation, we put 150 units in the hidden layer. We have not examined other number of units in the hidden layer, and it may be reduced to less number.

5. Simulation and Subjective Evaluation for System Output

Melodies and chord progressions for training data are chosen from Japanese traditional *Enka* songs of minor key. Each song is transposed to one key, to say, A-minor. 198 patterns are derived from 33 songs, and the network is trained with those data for 1000 epochs, ϵ in Rumelhurt [2] set to 0.01. The total sum of squared error became 63.98 .

The efficiency of this method was compared with the table-driven chord generating module by subjective evaluation. The results showed that chord progressions generated by this method are no worse than the ones produced by the original chord generation module at least. Twelve accompaniments for chosen six *Enka* song melodies, not included in the training data set, were evaluated. Those Melodies were hummed into the system and every transcribed data was harmonized by two modules, then twelve accompaniments were obtained. 19 subjects evaluated these accompaniments in terms of naturalness by five categories — 1 for *unnatural* to 5 for *natural*. The scale values for this method and the transition table method were 3.13 and 2.91, respectively, but it followed from F test that the difference between the two values was not significant.

6. Conclusions

In this paper a neural network based machine harmonization method which requires no rules as for chord progressions is described. In the implementation on the personal computer based music arrangement system, the network was trained with Japanese *Enka* songs to associate chord progressions from melodies. Through the training, only the training pairs of the existing melodies come in through the transcription module and chord progressions burst into chord tones were presented to the network. Nevertheless, this method showed the equal performance with transition table method, that the results of the subjective evaluation for accompaniments obtained by the arrangement system equipped with this neural network based module yields.

References

[1] Fujii, H. et. al., "A Music Arrangement System based on Hummed Melodies", Proceedings of The 1st International Conference on Music Perception and Cognition, 1989.

[2] Rumelhart, McClelland and the PDP Research Group, "Parallel Distributed Processing vol.1", the MIT Press, 1986.

[3] Mikami, T. and Kazuo, I., "The Problem and the Knowledge for Solving on the Stage of Chord Choice of MUSAS:Musical Arrangement System"(In Japanese), The Trans. of the Japanese Inst. of Electronics, Information and Communication Engineers, vol.J72-D-II No.6, pp896-905, 1989.

[4] Ricigliano, D.A., "POPULAR & JAZZ HARMONY", Donato Music Publishing Company,1967.

Fault Tolerance in Neural Networks

Gnanasekaran Swaminathan, Sanjay Srinivasan, Shanka Mitra,
Jay Minnix, Barry Johnson, and R. M. Iñigo
Dept. of Electrical Engineering
University of Virginia
Charlottesville, VA 22903
804-924-6111

Abstract—In this paper, we have investigated the faults that may occur in hardware implementations of neural networks. The effect of these faults on network performance is simulated and analyzed by modeling the faults and the neural network in software. We shall consider four different types of faults in our analysis—1. Noisy inputs, 2. Connection weight modifications, 3. Stuck-at-faults, and 4. Delay faults. A Hopfield auto-associative memory is used as the example network in our experiments.

1. Introduction

A fault-tolerant system is one that can continue to correctly perform its specified tasks in the presence of faults. The failure of a hardware component in a fault-tolerant system does not inhibit that system's ability to correctly execute its functions. Fault Tolerance is the attribute that enables a system to achieve fault-tolerant operation [1].

Systems achieve fault tolerance through the use of redundancy. Since a neural network (NN) employs a large number of functionally simple and similar elements, more than necessary to realize the function the NN is supposed to realize, it is inherently fault tolerant. Belfore et al. [2] and Kidwell [3] have also looked into the fault tolerant behavior of NNs. The objective of this paper is to model faults that occur in hardware implementations of NNs in software and analyze the behavior of the networks in the presence of faults. A Hopfield auto-associative memory is used as the example network in our experiments.

The faults that occur in NNs include,
1. Noisy inputs
2. Connection weight modifications
3. Stuck-at-faults
4. Delay faults.

NNs are expected to operate in noisy environments, where the received inputs do not exactly match the inputs on which they have been trained. A large class of NNs is used to solve pattern matching problems where almost always partial information is provided as the input. This partial information can be viewed as *noisy input*. In the binary domain (inputs are binary vectors), a noisy input would be a vector with some bits complemented.

Modification of connection weights can occur in hardware implementations of NNs. Carver Mead, in his VLSI implementation of a Hopfield net, used an operational amplifier (Op Amp) as a neuron [4]. Each Op Amp provides positive and negative outputs to realize positive and negative connection weights. A bridging fault between the Op Amp's outputs could reverse the polarity of the weights used. In Howard et al. 's implementation, each weight is stored as a binary word [5]. Complementing bits in a word results in weight modification.

The classical stuck-at-fault model from switching theory is used to model faults within a neuron (with binary outputs) as stuck-at-0 and stuck-at-1 faults at the neuron's output. Hardware neurons in a net have varying processing delays because of variations in device parameters (due to fabrication, aging). This effect is modeled as a delay fault. Section 2 describes the simulation settings. The simulation results of the above four faults are presented in Section 3 before concluding in Section 4.

Table 1: Patterns stored in the Hopfield net

Patterns	Groups			
	G_1	G_2	G_1	G_3
P_1	11111111	00000000	1	00000000
P_2	00000000	11111111	0	00000000
P_3	00000000	00000000	0	11111111

2. The Hopfield Net and the faults

We illustrate, as an example, the fault tolerant behavior of a Hopfield net. A 25 neuron binary net is used as an auto-associative memory [6]. The net stores the 3 patterns shown in Table 1.

Each pattern is divided into 3 groups (G_1, G_2, and G_3), a group of all 1s and two groups of all 0s. These patterns are mutually orthogonal to each other. This is done to enhance the fault tolerant behavior. The neurons are also considered to belong to one of the three groups. The Hamming distances (HDs) between the patterns are,

$$HD(P_1,P_2) = 17 \quad (1)$$

$$HD(P_1,P_3) = 17 \quad (2)$$

$$HD(P_2,P_3) = 16 \quad (3)$$

The HNC-Anza neurocomputer was used for the simulations. The simulation results are presented in the following section.

3. Simulation Results

(A) NOISY INPUTS

In this experiment, the stored patterns with some bits complemented are presented to the net. A specific method is followed in the introduction of noise. At any time, the noise introduced causes the resultant pattern to be related to two of the stored patterns. For example, consider P_1 and P_2. In case 1, we start with P_1 and introduce noise in G_1 until the G_1 field of P_1 becomes equal to the G_1 field of P_2. Next, we start with P_1 and introduce noise in G_2 until the G_2 fields of P_1 and P_2 are equal. Finally, we start with P_1 and introduce noise in both G_1 and G_2 until the result is P_2.

The simulation results of the above example are presented in Table 2. In case 1, the G_1 bits of P_1 are complemented sequentially and the pattern is presented to the net as P_i. The $HD(P_1,P_i)$ increases while the $HD(P_2,P_i)$ decreases. We see that the net produces the correct pattern, P_1, as long

Table 2: Simulation results of a noisy input experiment

Input Pattern (P_i)	$HD(P_1,P_i)$	$HD(P_2,P_i)$	Pattern Found
0111111100000000100000000	1	16	P_1
0011111100000000100000000	2	15	P_1
0001111100000000100000000	3	14	P_1
0000111100000000100000000	4	13	P_1
0000011100000000100000000	5	12	P_1
0000001100000000100000000	6	11	P_1
0000000100000000100000000	7	10	P_1
0000000000000000100000000	8	9	P_1
1111111110000000100000000	1	16	P_1
1111111111000000100000000	2	15	P_1
1111111111100000100000000	3	14	P_1
1111111111110000100000000	4	13	NONE
1111111111111000100000000	5	12	NONE
1111111111111100100000000	6	11	NONE
1111111111111110100000000	7	10	NONE
1111111111111111100000000	8	9	NONE
0111111110000000100000000	2	15	P_1
0011111111000000100000000	4	13	P_1
0001111111100000100000000	6	11	NONE
0000111111110000100000000	8	9	NONE
0000011111111000100000000	10	7	NONE
0000001111111100100000000	12	5	NONE
0000000111111110100000000	14	3	P_2
0000000011111111100000000	16	1	P_2

as at least 1 bit in G_1 remains uncomplemented. This is because of the zero inhibitory signals from groups G_1 and G_2.

In case 2, the G_1 bits of P_1 are correct and the G_2 bits are sequentially complemented. The expected output is P_1 whose G_2 neurons should have output of 0. The negative weights from the G_1 neurons give a total inhibitory input of -9 to all the G_2 neurons. Thus, as long as the excitatory input to the G_2 from their own connections within the G_2 neurons remains less than +10, the net produces the correct output. Thus, the net can withstand 3 bits of G_2 being complemented as a fourth bit complementation will give the G_2 neurons an excitatory input of +12 and then cause the net to produce a wrong output.

The behavior of the network when noise is introduced in both G_1 and G_2 is similar except that the inhibitory input to a G_2 neuron from the G_1 neurons is reduced and hence a fewer number of faults are tolerated. The results obtained for the other two pattern combinations can be explained in a similar way.

(B) WEIGHT MODIFICATION

In this experiment, connection weights are modified and correct inputs are presented to the network. The net is considered to have failed when it fails to reproduce any one of the inputs. The weights to be modified can be selected in two different ways, randomly and modified singly, or randomly and modified symmetrically, *i.e.* both W_{ij} and W_{ji} modified simultaneously. Faults in VLSI chips may be restricted to very small areas, spot faults, or may affect very large areas. Accordingly we consider both, the modification of weights located within a small area of the weight matrix and weights distributed uniformly across the entire matrix. Further, selected weights are either set to 0 to model physical shorts across resistors or their signs are reversed to model bridging faults between the Op Amp outputs.

A random number generator is used to select the weights. The seed was used the control the range of numbers generated and hence the area of the weight matrix that was affected. For example, a seed of 52 resulted in a uniform selection across the matrix, while a seed of 364 caused weights in the central area of the matrix to be selected.

When a large number of the weights of a neuron are modified, there is a change in the sign of the sum of its weights, resulting in faulty outputs. Thus, the net fails when a particular neuron has, in the process of weight selection, a large number of its weights modified. Setting of weights to 0 requires a larger number of modifications to cause a sign change than when the signs of the weights are reversed. Hence, the net is able to tolerate a larger number of short circuit faults than bridging faults. When weights are modified uniformly across the matrix, the probability of a large number of a particular neuron's weights being modified successively is low, and hence the net is able to withstand a larger number of faults than when the area affected is small. The effect of modifying symmetric weights again depends on the area of the matrix being affected. This may or may not result in quicker failure than when non-symmetric weights are modified. The results are condensed in Table 3.

The values of the modified weights also affects the performance. Modification of large valued weights causes a faster decay in the performance.

(C) STUCK-AT-FAULTS

In this experiment stuck-at-faults are introduced at the outputs of neurons and correct inputs are presented to the net. A net is said to produce the correct output if the fault free neurons output the correct values. Also, s-a-1 (s-a-0) faults are introduced in neurons which normally output a 0 (1).

The effect of s-a-0 faults is to eliminate some normally necessary weights in the output evaluation, while the s-a-1 faults introduce a normally ineffective weight into the output evaluation. Hence, the relative effects of these two faults is the same and depends on the values of the weights they affect. In one

Table 3: Weight modification experiment results

Seed	Type of weight modification	No. of weights modified before failure	Pattern at which the net failed
52	Non symmetric	180	P_3
52	Symmetric	144	P_3
364	Non symmetric	21	P_2
364	Symmetric	38	P_2

experiment, pattern P_1 was presented to the net with neurons in G_2 and G_3 s-a-1. The results are shown in the Table 4. The interesting point is that when G_2 and G_3 neurons are simultaneously s-a-1 the net withstands a larger number of faults than when the faults lie only in G_2 or G_3. This is because with s-a-1 faults, the G_2 and G_3 neurons mutually inhibit each other causing them to continue to output 0s for a larger number of faults.

(D) DELAY FAULTS

In this experiment, neurons with delay faults are modeled as having D latches at their outputs. If the processing delay of a neuron has to be increased over its normal value Δt by a factor of n, we place n D latches in series with the output. If each evaluation iteration through the net is considered as the basic propagation delay of the neuron, then after (n+1) iterations the correct output will be available. The net will then take n more iterations to arrive at the correct output.

4. Conclusion

In this experiment we have investigated the faults that may occur in hardware implementations of NNs and have observed the effects these faults have on network performance through simulations. The faults considered include: (1) Noisy input, (2) Connection weight modification, (3) Stuck-at faults, and (4) Delay faults. The results presented here are for the cases where all faults were considered independently. Simulations with combinations of faults were run and as expected, the network performance degraded faster. With some combinations however, one type of fault counteracts the effect of another type and increases the fault tolerance. The large numbers of faults tolerated by the net is due to the high orthogonality of the stored patterns.

5. References

[1] B. W. Johnson, "*Design and Analysis of Fault tolerant Digital Systems*," Addison-Wesley, Reading, MA, 1989.

[2] L. A. Belfore, B. W. Johnson, and J. H. Aylor, "The design of inherently fault tolerant systems," *Concurrent Computations*, Ed. S. K. Tewksbury, B. W. Dickinson, and S. C. Schwartz, Plenum Press, NY, pp 565-583, 1988.

[3] E. Kidwell, "Brainchild," *IJCNN*, 1989.

[4] D. W. Tank and J. J. Hopfield, "Collective Computation in Neuronlike circuits", *Scientific American*, Dec. 1987.

[5] R. E.Howard, L. D. Jackel, and H. P. Graf, "Electronic Neural Networks," *AT&T technical journal*, Vol. 67, Issue 1, Jan./Feb. 1988.

[6] J. J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Abilities," *Proc. Natl. Acad. Sci., USA* , Vol. 79, 1982.

Table 4: Results of s-a-1 fault simulations
Input pattern = P_1

No. of s-a-1 neurons in G_2	No. of s-a-1 neurons in G_3	Pattern found
3	0	FOUND
4	0	NOT FOUND
5	0	NOT FOUND
6	0	NOT FOUND
0	3	FOUND
0	4	NOT FOUND
0	5	NOT FOUND
0	6	NOT FOUND
2	0	FOUND
2	2	FOUND
4	2	FOUND
4	4	FOUND
5	4	NOT FOUND
5	5	NOT FOUND
6	5	NOT FOUND
6	6	NOT FOUND

Space-Scanning Curves for Spatiotemporal Representations... useful for large scale neural network computing

Harold Szu* and Simon Foo
FAMU/FSU Engineering, Florida State University,Tallahassee, FL 32316
*Naval Research Laboratory, Code 5756,Washington, DC 20375

Abstract:

Neural Networks (NN) for computing should be able to deal with a large scale visual processing, since animal species have survived through the prey-and-predicator relationship by quick scanning and computing. In order to avoid the unnecessary detail in the wide open and cluttered field, an efficient spatiotemporal analysis must be based on motion detections and the cause and effect judgement. Thus, a space-filling curve covers a large scale spatiotemporal domain to any desire degree of accuracy, suggested by the french mathematician Peano and supported by human eye scanning experiments, is adopted for dividing a large scale NN in a controllable and successive coarse graining sense. This kind of Peano curves is general and applicable to many classes of spatiotemporal control problems in a natural environment: TSP, Rocket Engine Plume, Autonomous Robots and Vehicles, to name a few; all require real time optimization, ahead planning for anticipation, as well as the associative recall for the past performance experience.

Keywords: Hierarchical Neural Network, Spatiotemporal Representation, Space-filling, Fractal, Trajectory, Optimization, Planning, Associative Recall, Divide-Conquer, Cut-Splice, Travelling Salesman Problem, NP-Complete

1. Background:

Our case study involves a classical generic problem - the large scale traveling salesman problem (TSP), which belongs to the class of NP-complete (nondeterministic polynomial-time complete) problems, as noted by Papadimitriou (1977) and Garey and Johnson (1980). The task of the salesman is to make a round trip of all the N cities once only, and determine the tour of the minimum possible distance. In the simplest case where the cities are points in the plane and travel cost is a function of the Euclidean distance, the problem is to find a polygon of minimum perimeter passing through all cities. Although easy to state, the problem is hard to solve; the number of possible tours increases exponentially with the number of cities. In fact, for a N -city problem, there are $N!$ possible tours. However, since each tour may be traversed either clockwise or anti-clockwise and each tour may be started from a different city, the number of unique tours is $N!/2N$. Since exact solutions are unattainable for problems of any appreciable size in reasonable time, there has been interests in heuristic and parallel distributed processing (PDP) approaches. Lin and Kernighan [1] introduced a heuristic method for solving TSP based on an interchange strategy and backtracking. In 1985, Hopfield and Tank [2] pioneered a heuristic method based on neural network energy landscape approach to solve the TSP. Since then, Wilson and Pawley [3] have pointed out a convergence difficulty of the Hopfield and Tank algorithm in solving a larger scale TSP. Szu [4] proposed a Fast TSP algorithm based on an efficient hybrid neuron model (analog input and binary output) that is appropriate for an internal logic problem without noise(as opposed to the sigmoidal neuron suited for an external sensor problems with noise). Moreover, the binary output ignores other neurons' inhibitions and, by jumping to conclusion to visit a particular city for any weak positive encouragement, can energize the monotoneously decaying Lyaponov system to keep the optimization ball game going. Fast TSP allows no self-talk (zero-diagonal interconnect matrix elements) in order to insure the absolute convergence [$dE/dt < 0$ for any real $E(V_i)$, e.g. [11] for a general proof]. Lastly, Fast TSP adopts the necessary and sufficient constraint of a permutation matrix (row-sum and column-sum equal to one), in order to eliminate the ground state ambiguity in converging to a valid tour only. Never mind about the dynamics. Imagine the statistics what happens when a board of chinese go game gets bigger, as one requires to throw one and only one piece of stone in each row and column. For much larger size TSP, even the modified H-T algorithm, Fast TSP [4], becomes inefficient using single layer neural networks, because the phase space for the valid solutions becomes exponentially smaller as the problem size increases. Thus, Foo and Szu [5] introduced an ad hoc cut-and-splice algorithm using a first layer of four partitioned neural networks operating in parallel and a second layer single network for global interconnect. Other recent approaches includes an "elastic net" method by Durbin and Willshaw [6], and Angeniol, Vaubois, Texier [7] based Kohonen adaptive net, divide-and-conquer algorithms by Prof. Karp (Turing Award, 1985), and space-filling curves by Bartholdi and Platzman[8].There are several interesting developments that have led to the present work. Our work is motivated by the question of how to solve large scale optimization problems using divide-and-conquer neural networks and fractal space-filling curves (c.f. **Appendix A What are Fractals ?**).

2. Reduced Neural Nets over domains controlled by Space-Scanning Curves:

A lexicographical scan is not suitable for vector representation of a two dimensional *image* since two pixels next to each other on two rows become far apart on the row-by-row scans. A spiral scanning is only better in maintaining the relationship near the center of the spiral but has a similar drawback as in row-by-row scan. Our challenge is to systematically break down a large scale neural network over a large spatiotemporal domain. We wish to map arbitrary spatiotemporal domain onto a linear sequence that can be easily partitioned and yet maintain the causal neighborhood relationship.

The basic question of how to partition and preserve a neighborhood relationship has been posed by a French mathematician Peano [9], who has proved that the mapping of the two-dimensional neighborhood relationship can be preserved onto a real line. In other words, any partition on the real line defines connected regions on the plane that can fill the plane. A deterministic fractal curve has been developed in a recursive algorithm that can fill the plane in a successively increased resolution, as shown in Figure 2. In other words, the line density of the space filling curves increases linearly as the number of iteration increases, from a coarse graining to a fine resolution, e.g. Dragon curves, etc.

3. Applications:

Numerical simulations will be presented. A stochastic application has been documented in the 1-D fast simulated annealing feature extraction technique based on the 1-D Peano curve from the 2-D occluded and cluttered moving objects [12].

3.1 Fractals for large-scale TSP? We think so. In small-scale TSP, the "fractals" may not be applicable. But for large-scale TSP (with thousands of cities), fractals can be used to identify the flavor, the attrator source, the chaos, etc. The proof is made by calculating the fractal dimension D of both city sets and showing that they both have approximately the same D. This phenomena is analogous to that of the convergence of a fern leaf (c.f. Barnsley [10]). Imaging how do we exam a maple leaf, whose circumference may be approximately inscribed by a convex polygon of minimum perimeter connecting edge cities solving TSP of several thousand cities. Consider initially clumps in fewer tens cities, and later hundreds, and thousands cities, by beginning with a blurred/lowpass maple leaf that has been successively bring into sharp focus.

We present two algorithms to do precisely that based on divide-and-conquer with space-filling fractal curves. The first ad hoc procedure is as follows: (1) Partition an N-city map recursively until each square (or cluster) has an approximately even distribution of cities. (2) For each neural network representing each cluster of cities: (2a) Prefix a pair of "starting" and "ending" terminal cities; (2b) With local tour partially fixed, let NN converge a open-tour. (3) The fractal curve (such as the one in Figure 2) is then applied to globally connect the open tours in a successively increased resolution until all cities are covered. Figure 1 illustrates the first algorithm with the city map being recursively partitioned and the corresponding representation by a hierarchy of neural networks. The fractal curve is applied at the "merging" layer. The depth of partitioning depends on the distribution and concentration of cities. The second algorithm is a variation of the first with "even" partitioning of the city map, as shown in Figure 2. Each square (representing a neural network) is processed based on the flow of the fractal curve. With increased resolution (or more levels of partitioning), it is possible to preserve the neighborhood of the cities. However, this means more boundaries, i.e., more global interconnects. Therefore, we will investigate the relationship of the minimum tour distance and the number (or depth) of partitioning. A parameter as a function of tour distance, number of cities, and number of cycles required for convergence, is identified.

3.2 Spatiotemporal Jet Engine Plume Diagnosis: Consider experiments performed by Strahle (1987) to study the chaotic dynamics present in jet exhaust flame. The input data are time series for the temperature and velocity at two different points in the jet flame. The flame is probed by: (a) a **laser beam**, where the photons bounces off the fast moving (turbulent) molecules in the exhaust, and a sensitive photodetector measures the characteristics of the deflected light. The output of the receiver is a voltage representing the temperature of the jet flame, a function of time. (b) a **very thin wire**, where a constant temperature is to be maintained through a wire in the flame. The voltage necessary to hold the temperature constant is recorded. This voltage (a time series), after scaling, gives the velocity of the jet flame as a function of time. A close-up look of a portion of the waveform reveals the self-similarity property of the time series. The fractal dimensions of the graphs of the two time series are calculated using the Box Counting Theorem. Both sets of data yields a fractal dimension $D \approx 1.5$, suggesting that, despite the different appearances in the graphs, there is a common source for the data. This common source is the chaotic dynamics of a certain flavor present in the exhaust. Therefore, fractal dimension provides a means or a measurable parameter to classify chaos.

3.3 Autonomous Robot/Vehicle Control, Planning, Optimization: For example, the two-dimensional(2-D) x-y space and one-dimensional (1-D) time t are mapped onto a one-dimensional (1-D) arc length s, e.g. Peano curve. The arc may be a parameter of time ordering on a linear interval (say at a piecewise constant speed).

Then, all those possible world-line arc partitions are mapped back onto the space-filling 2-D regions, where every regions are governed with reduced neural networks. This 1-1 mapping can be used for searching for better trajectory on such a large-scale spatiotemporal representation.

4. Conclusion:

Hierarchical neural networks are useful for such a successively coarse graining approximation controlled by deterministic space-scanning (or space-filling, namely fractal) curves. Thus, a dynamically reconfigurable neural network architecture [11] can provide a piecewise quasilinear approximation, where increasing the resolution gives better approximation to a large scale nonlinear problem.

Appendix A: What are fractals?

Fractals (or space-filling curves) are iterative functions operating on a geometry with statistical *self-similarity* and *self-affinity*. In particular, fractals consists of transformation, translation, and duplication. Fractal geometry provides both a description and a mathematical model for many of the complex shapes found in nature. Traditional Euclidean geometry suits man-made objects but could not easily described natural shapes such as coastlines, mountains and clouds. The property of objects whereby magnified subsets look like (or identical to) the whole and to each other is known as *self-similarity*. It is the major characteristic of fractals and it differentiates them from traditional Euclidean shapes. Therefore, fractal shapes are said to be self-similar and invariant to scaling. *Self-affinity* is a non-uniform scaling where shapes are (statistically) invariant under transformations that scale different coordinates by different amounts. Fractals belong to the class of "mathematical monsters" (such as Cantor sets, Weierstrass functions, Peano curves, curves without derivatives and lines that could fill space), created at the turn of this century by "naturalist" scientists. However, they could not find any real applications until computer graphics came along. Since then, fractal geometry plays an important role in realistic modelling of natural phenomena in computer graphics.

References

[1] S. Lin, B. W. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem," *Operations Research*, Vol. 21, pp. 498-516, 1973.

[2] J. J. Hopfield and D. W. Tank, "Neural Computation of Decisions in Optimization Problems," *Biological Cybernetics*, Vol. 55, pp.141-152, 1985.

[3] G. V. Wilson and G. S. Pawley, "On the Stability of the Travelling Salesman Problem Algorithm of Hopfield and Tank," *Biological Cybernetics*, Vol. 58, pp. 63-70, 1988.

[4] H. Szu, "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," *Proc. of IEEE ICNN*, San Diego, CA, 1988.

[5] S. Foo and H. Szu, "Solving Large-Scale Optimization Problems by Divide-and-Conquer Algorithms," *Proc. of Int. Joint Conf. on Neural Networks, Washington, DC, June 1989.*

[6] R. Durbin, D. Willshaw, "An Analogue Approach to the Travelling Salesman Problem using an Elastic Net Method,"*Nature*, Vol. 326, 16 Apr. 1987.

[7]B. Angeniol, G. d.L.C. Vaubois, J.-Y. L. Texier, "Self-Organizing Feature Maps and Travelling Salesman Problem," *Neural Networks, Vol.1, pp.289-294, 1988*

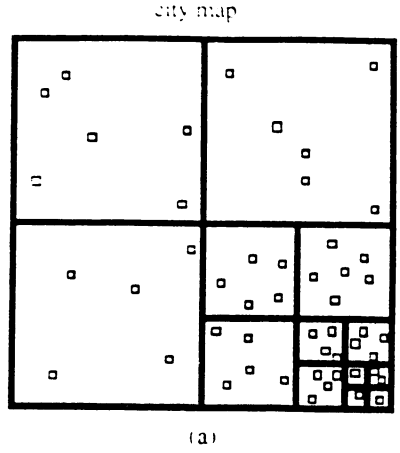
[8] J. J. Bartholdi and L. K. Platzman, "Heuristics based on Space-filling Curves for Combinatorial Problems in the Plane," *PDRC Technical Report 84-08, Georgia Institute of Technology, 1984.*

[9] G. Peano, "Sur une Corbe qui Remplit Toute en Aire Plaine," *Math. Ann., Vol. 36, 1890.*

[10] M. Barnsley, "Fractals Everywhere," Academic Press, 1988.

[11] H. Szu, "Dynamic Reconfigurable Neural Networks based on extended McColluch-Pitts Neurons," *Proc. of Int. Joint Conf. on Neural Networks, Washington, DC, June 1989*

[12] H. Szu, K. Scheff,"Simulated Annealing Feature Extraction from Occluded and Cluttered Objects,"*Proc. of Int. Joint Conf. on Neural Networks, Washington, DC, JJan. 15-18, 1990*



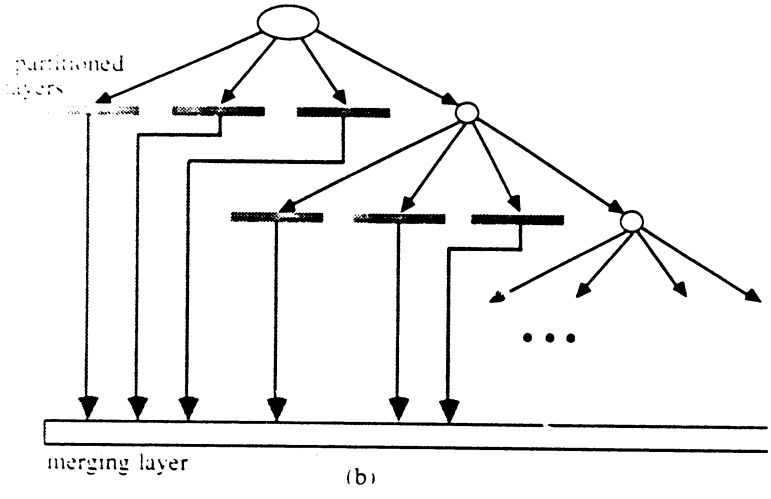
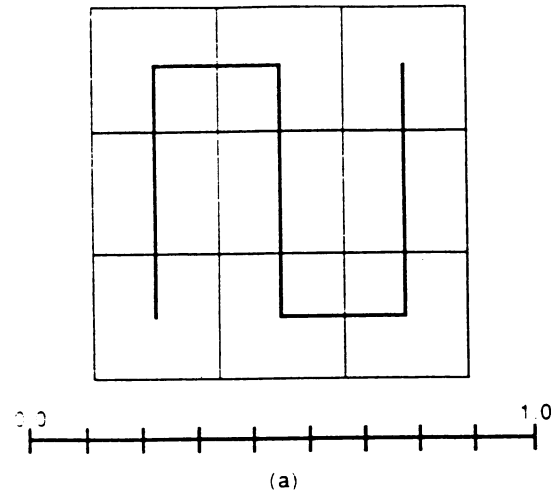


Figure 1. City map partitioning and corresponding representation based on hierarchical neural networks.



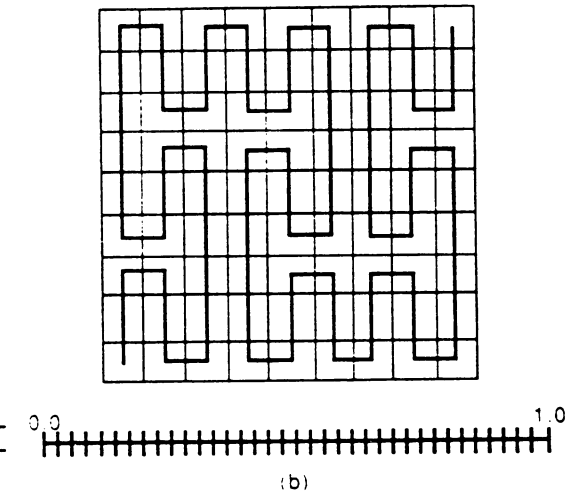


Figure 2. An example fractal curve on a unit square city map and its mapping onto a normalized unit interval, with (b) increased resolution.

Comparison of the performances of three popular neural network architectures

A C Tsoi
Department of Electrical Engineering,
University College,
University of New South Wales,
Campbell ACT 2600,
Australia.

Phone: +61 62 688236
Fax: +61 62 688443
Telex: ADFADM AA 62030

Indexing terms: Neural networks, Hopfield net, Hamming net, Multi-layer perceptrons.

Abstract.

In this paper, the performances of three popular neural network architectures, viz. Hopfield net with asymmetrical connection matrix, Hamming net, and a multi-layer perceptron trained using radial basis functions are compared on a reasonably realistic problem of database retrieval. The neural net architectures are compared on the following criteria: (a) noise immunities, (b) generalisation capabilities, (c) random failure of neurons.

1.Introduction.

Recently, there is much interest in artificial neural network methodology [4,9]. This may be attributed to the popularisation of a training algorithm for a multi-layer perceptron [4], and to the enormous interest generated within the physicist community on the possibility of applying spin glass analysis to a class of auto-associative neural networks, viz Hopfield net [see eg ref 9].

There are indeed many possible neural network architectures. For example, there is an auto-associative neural network commonly known as the Hopfield net [1,2]. This network has a very simple architecture. The output of each neuron is fed back to the inputs of the neurons. Because of the possibility of extending current understandings in spin glass analysis to this network architecture, as a result, its performance has been analysed to a large extent by physicists who are trained in statistical thermodynamics. There is a class of neural network architectures which is based on the simple clustering concept [6]. The following neural network architectures can be loosely classified in this category: Hamming net, Carpenter/Grossberg net, Kohonen Feature map [6]. There is yet another class of neural network methodology which is based on the perceptron algorithm [4,5]. This is a feed forward only network, ie the signals are flowing in one direction only.

Despite the popularity of some of these neural networks, there is very little comparisons on their performances. Very often, one researcher would use one particular neural network architecture for a particular application, without giving the underlying reasons why that particular neural network architecture is chosen in the first instance. To a newcomer to this field, this may give an incoherent view of the different types of neural networks. In fact, one of the questions that a newcomer to the neural network field would ask is: which neural network architecture is most appropriate to the application in mind.

In this paper, we will consider three typical neural network architectures, viz. Hopfield net Hamming net and a multi-layer perceptron. The training algorithms for these networks are atypical in that for the Hopfield net, we will use a training algorithm which will give rise to an asymmetrical connection matrix [3]; in the multi-layer perceptron we will use a training algorithm based on radial basis function method [7,8] rather than the more traditional gradient based algorithm known as the back propagation method [4]. The main reason why we chose to use slightly different training methods in preference to the more established training methods is that, we find, in actually applying the various network methodologies to a practical problem, it is necessary to modify the training algorithms somewhat in order that the methods can be applied.

In order to focus our attention, we will apply the three neural network architectures to the same problem, viz a database retrieval problem. By applying all three methods to the same example, we hope to convey the idea that these three neural network architectures, which are typical examples of most of the neural network architectures that have been proposed by the researchers, even though they might appear quite different, are nevertheless capable of being applied to the same practical problem. Furthermore, we hope, by applying these three methods on the same problem, their performances can be compared, thereby indicating the strength and the weakness underlie each method.

We compare the performances of the networks with respect to the following criteria: (a) noise immunity, (b) generalisation capabilities, (c) fault tolerance. By exposing each neural network architecture to these performance measures, we hope to give some indications as to which network architecture to choose for a particular application.

Due to the space limitations, we will only present the results rather than to present the theory involved in each network. They can be obtained from the literature [see eg ref 6] or from the author.

2. Results of the comparisons.

The database under study is the same one as given by Anderson et al.[10] It is a pharmacological database involving drugs, bateria, and the route of administration. It consists of 31 records, each being 25 alphanumeric characters long. We use the same encoding algorithm as Anderson et al [10], each record is converted to a 200 x 1 vector using the alphabets {-1,1}

In the Hopfield net and the Hamming net, there is no problem in identifying the input and output variables as they are self evident. For the multi-layer perceptron case, we assume that the output classes to be 31

The results of different runs through the programs can be summarised as follows:

(1). Retrieval of the original basis pattern vectors.
In this case, it is found that all three neural network architecture can correctly retrieve the basis pattern vectors when the probing vector is one of the original basis pattern vectors.

(2) Retrieval of the basis pattern vectors when they are contaminated by noise.
We simulate the contamination of noise by the following method: We assume that the bits of the original basis pattern vector are flipped according to whether a random number obtained from a random number generator is greater than a predefined probability function.

It is found that as long as the percentage of bits flipped is less than 35% of the length of the basis vector for the Hopfield net; and it is less than 45% of the length for the Hamming net, the neural networks can retrieve correctly the original basis pattern vectors. The percentage of the number of bits flipped in the Hopfield net could possibly be increased by using a large constant associated with the Gardner's method [3] to yield a deeper basis of attraction.

For the multi-layer perceptron the noise immunity of the network appears quite significantly lower than the Hopfield net or the Hamming net. We find that if the noise is more than 10% the network ceases to classify the input vecotrs.

Furthermore, for the Hamming net and the multi-layer perceptron, the output would attain to the correct classification as indicated by the output of the network. On the other hand for the Hopfield net, it is possible to observe the way in which the network converges to the minimum. For example,if the initial vector is:

Enteroba-bacUrTrInCephalo

and that we have 38 bits of the basis vector are flipped

Then we find that the network converge in the following manner:

iteration	output from the network
0	Eotecnjm%faDM25zI.A%xza-G
1	Dktesobi-`acUpTrInCephado
2	Djtesoba-bacUpTrInCephado
3	Dntesoba-bacUrTrInCaphalo
4	Dnteroba-bacUrTrInCephalo
5	Enteroba-bacUrTrInCephalo
6	Enteroba-bacUrTrInCephalo

It is worthwhile to note that the network seems to have settled quite quickly from its initial rather noisy input in the first iteration as something already similar to the original pattern vector. It is observed also that the changes in each iteration after that are small.

It is tempting to speculate that this behaviour mimicks the way how human recalls a pattern. Initially we "home" into a pattern vector which is already quite close to the final steady state vector. Thereafter each iteration, the pattern recalled is gradually refined until we recognise that this is the correct vector. However one should not attempt to place too much emphasis on this speculation, as the Hopfield net architecture, admittedly, is a very crude model for the underlying behaviour of human memory recalls.

(3) Generalisation capabilities..

In order to test the capability of the network to generalise, we take a segment from one basis pattern vector, concatenated it with a second segment from a second basis pattern vector, selected at random from the exisitng 31 records, and then further concatenated it to a third segment from a third basis pattern vector, again selected at random. Then we input this "amalgamated" vector as the probing vector into the networks.

It is found that none of the network generalise at all. The only possible indication of any generalisation capabilities came from the Hopfield net. We found that, for some cases, it seems to have some generalisation capabilities. For example, we find that

iteration	output of the Hopfield net
1	Proteus"+cocUrTrInGenicil
2	Proteuw`-bocUrtrInPenicil
3	Pvoteuw`-bkcUrtrInPenicil
4	Pvoteuw`-bicUrtrInPenicil
5	Pvoteuw`-bacUrtrInPenicil
6	Pvoteus`-bacUrtrInPenicil
7	Pvoteus`-bacUrtrInPenicil

In this case we find that the network, instead of settling to the pattern vector, Proteus -bacUrTrInGentamy, it settles to a new pattern vector which is not contained in the original database.This is particularly interesting in that the neural net seems to think that it is possible to treat Proteus -bacUrTrIn (stands for Urinary tract infection) with Penicil (stands for Penicillin).

One should not be tempted to conclude from this observation that the network has some kind of generalisation properties. This is because of the fact that the machine seems to have 6 out of 31 records which give different pattern vectors than those that werte originally stored in the database. Much more testing needs to take place before one can comfortably conclude that the Hopfield net has some kind of generalisation properties.

(4) Robustness with respect to the failure of the neurons

It is found that both the Hamming net and the Hopfield net performance degrades "gracefully" with the degradation in the number of neurons which are mulfunctioning. On the other hand the multi-layer perceptron does not degrade well. It is found that the network just wrongly classify the outputs.

3. Conclusions

In this paper we considered the problem of database retrieval and we have employed three typical neural network architectures to see if they can retrieve the records under various conditions.

It is found that all the networks can retrieve the original pattern vectors if they are input as the probing vectors. Furthermore if the probing vectors are contaminated with noise then the original pattern vectors can be retrieved provided that the noise contamination is not high. Both the Hopfield net and the Hamming net appear to degrade their behaviour rather gracefully as the number of malfunctioning neurons increases. Both networks simply do not produce some of the pattern vectors, but they do not affect the retrieval of other unaffected pattern vectors. On the other hand the multi-layer perceptron appears to be very "flakely" to the degradation of the performance. It simply gives up to classify the input pattern vectors. Only the Hopfield net appears to give some generalisation capabilities.

From these experiments, we can conclude:

(1) The neural nets considered are good pattern matchers. They can match a given pattern with patterns already stored in the memory, provided that the noise contamination is limited.
(2) Because of the distributed nature of the storage of the memory, even if some of the neurons failed, the performance of the networks, in general, degrade gracefully.
(3) As a generalisation device, the networks, at least in the way this paper considers, are not good at all.

In general, there is not much to choose among the networks if one's aim is to classify a given set of pattern vectors. Any one of the three neural network architectures are applicable. On the other hand, if one wishes to have some robustness in the hardware, then it is advisable to choose either the Hopfield net or the Hamming net. If one wishes to see how a network mimicks a human brain's behaviour in the memory recall, then it is perhaps best to use the Hopfield net.

It would be interesting to see if the Hopfield net methodology can reproduce some of the findings by psychologist in memory recognition and recalls.

4. References.

[1] Hopfield J., "Neural networks and physical systems with emergent collective computational abilities", Proc. of National Acad. Sciences, Vol. 74, 1982, pp2554-2558.

[2] Hopfield J., "Neurons with graded response have collective computational properties like those of two-state neurons", Proc. of National Acad. Sciences, Vol. 81, 1984, pp3088-3092.

[3] Gardner E., "The space of interactions in neural network models", Journal of Physics A: Math. Gen., Vol. 21, 1988, pp257-270.

[4] Rummelhart D.E., Hinton G.B., Williams R.J., "Learning internal representations by error propagation", in Rummelhart D.E., McClelland J.L. (Eds), Parallel Distributed Processing, Vol. 1, MIT Press, 1987, pp318-362.

[5] Lapedes A., Farber R.,"How neural nets work", in Anderson DZ (Editor), Neural Information Processing Systems, American Institute of Physics, 1988, pp442-456.

[6] Pao Y.H., Adaptive Pattern Recognition and Neural Networks, Addison-Wesley, 1989.

[7] Renals S., "Radial basis function network for speech pattern classification", Electronics Letters, Vol. 25, No. 7, 30 March 1989, pp437-439.

[8] Tsoi A.C., "Multi-Layer Perceptron trained using radial basis functions", submitted, July, 1989.

[9] Anderson D.Z., (Editor), Neural Information Processing Systems, American Institute of Physics, 1988.

Fault Tolerant Behavior of I^2t Parallel Computing Network

STEVEN W. WELCH AND RUSSELL G. BROWN
Systematix, Inc., P.O.Box 110335, Nashville, TN 37222
FRANCIS M. WELLS, A.B. BONDS AND LLOYD W. MASSENGILL
Vanderbilt University, P.O. Box 1824B, Nashville, TN 37235
KENNETH R. FERNANDEZ
NASA, EB44, Marshall Space Flight Center, AL 35812
Supported in part by NASA (NAS8-38049)

1. POTENTIAL FOR USE OF NEURAL NETWORKS IN SPACE ELECTRONICS

Massively parallel networks offer enhanced computational capability in a variety of fields including Space power system design. Cost effective operation of the Space Station will depend on the automation of tasks which have ordinarily been performed by human operators. "The four major aspirations for Space Station automation are (1) maximize productivity, (2) stay within budget, (3) minimize R&D risks, and (4) maximize crew safety."[2][6] To achieve these goals the initial automation schemes will include "both conventional automation schemes (algorithms and event driven logic programs) and expert systems...Conventional automation will handle the bulk of the regulation and high response tasks, as well as any crisis management tasks."[2] The approach will be to upgrade incrementally through several generations of automation systems until "intelligent autonomous systems" can be implemented by "building full-fledged executive controller hierarchical networks."[3] Even within these intelligent systems "algorithmic responses will be absolutely necessary when speed is of the essence, since deducing how to respond using standard AI techniques may require too much time."[3] Moreover, standard AI techniques implemented on a traditional von Neumann architecture suffer fundamental reliability problems since the traditional architecture is susceptible to severely reduced performance when an individual component fails. This is an undesirable property, since such component failure has been shown to be likely under conditions found in Space.

Neural networks, in addition to having the property of generalization, are much more tolerant of individual component failure. This property comes from the network architecture and the distributed nature of its parallel computation and database. Several researchers have shown these systems to be remarkably tolerant to the disabling of some fraction of the processing elements.[4][5] Related work by the Space Electronics Research Group at Vanderbilt University has shown similar tolerance to the systematic disabling of both processing elements and signal interconnections. Evidence of catastrophic effects of radiation on Space system reliability has been observed since 1975 with increasing regularity. These have included notable (and expensive) cases such as memory failures in the Pioneer Venus and Voyager probes, as well as numerous satellites. A complete refit of the Galileo probe (at great expense) with radiation hardened electronics has recently been performed after land-based simulations predicted large numbers of memory bit upsets in the Jovian radiation belts. It follows that highly reliable, radiation tolerant parallel computing networks for Space applications are worthy of investigation.

2. MASSIVELY PARALLEL NETWORK FOR COMPUTING I^2t

Simulations were performed on fundamental power system control blocks in order to determine (1) the feasibility of constructing massively parallel network computing structures which could be manufactured with available VLSI technology (e.g. 3 micron analog CMOS processes), and (2) their tolerance to transient disturbance. We modelled I^2t elements, which are used in power systems to protect components from damage due to large current surges caused by short circuit faults. These elements compute a measure of the energy that is applied to a constant impedance series device during a high current transient. One cause of such transients is the presence of intense Space radiation. Such radiation tends to cause semiconductor

power devices to turn on, thereby inducing current surges. A major advantage of a fault tolerant I^2t block would be its ability to function reliably in the presence of such radiation. I^2t blocks are effective only if they have the ability to perform rapid integration of the square of a fast transient waveform (typically 16 millisecond response is required). For this reason, I^2t computation requires the sort of speed which can be provided by parallel computing blocks. Current breadboards of the Space Station automated power distribution system located at Marshall Space Flight Center, Alabama utilize remote power controller modules containing solid state MOSFET switches. These breadboards use analog signal processors to compute the I^2t trip condition.[1] The time available to sense a fault condition and protect a semiconductor switch is critical due to the low thermal capacity of the die and the relatively high thermal resistance between the die and heatsink.

The I^2t neural computer comes in two forms. Both configurations input a 2's complement binary number representing a level of measured current. One configuration produces a 2's complement output which represents the amount of energy (I^2t) accumulated since the specified zero time. The alternate configuration has a single output (possibly with redundancy for fault tolerance) which goes high when the calculated I^2t reaches a given value. This configuration is effectively an alarm which trips when the calculated energy reaches a certain predetermined threshold.

Both configurations of the I^2t unit contain two core neural blocks with a total of three feedforward layers. The first unit takes the binary current level and converts it to a unary representation, with all outputs zero when the input is at its minimum magnitude, all outputs one when the input is at its maximum magnitude, and a proportional number of outputs one for an intermediate magnitude input. This processing block is a two layer network, easily realized with standard back-propagation methods. The second network is a single layer device which takes as input the output of the first block and integrates the square of that input. Its output is also an unsigned unary representation with all outputs zero when the integrated amount is zero, all outputs one when the integrator is saturated, and a proportional number of outputs one for intermediate integration levels. For binary output, the output of the integrator may be reconverted into 2's complement binary with another back-propagation network. The alarm configuration is achieved with a simple network which yields a one when the majority of its inputs are one. For fault tolerance, the inputs to this network are attached to the output representing the desired alarm level, as well as to the outputs surrounding that level. If more outputs are examined there is less chance that a faulty output or

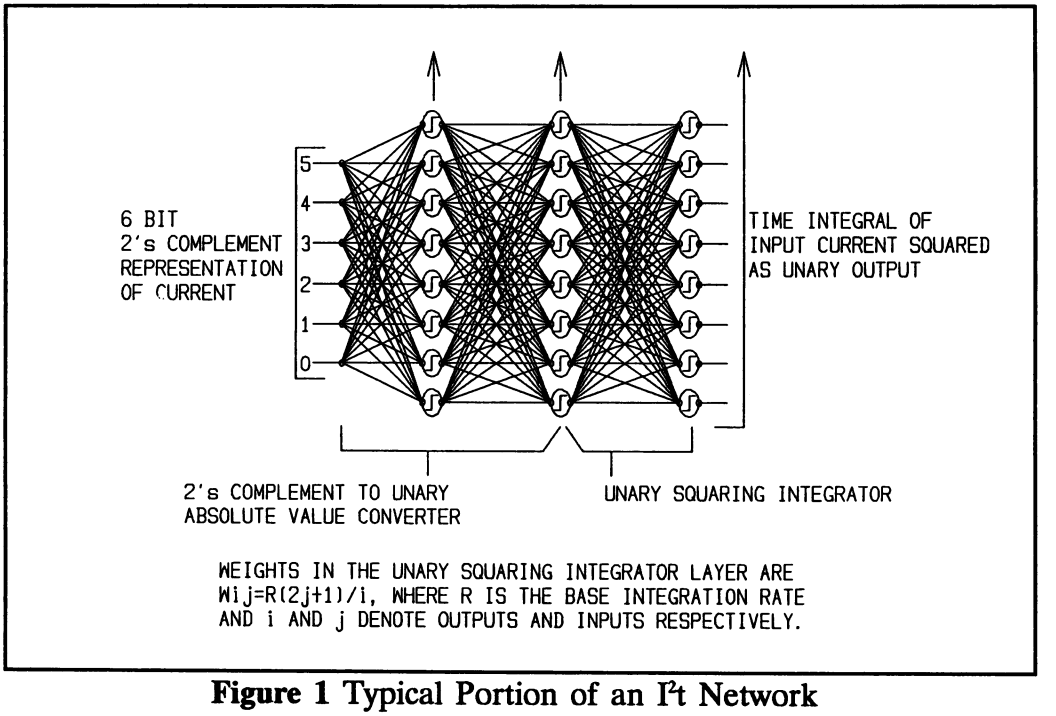


Figure 1 Typical Portion of an I^2t Network

outputs will set off the alarm inappropriately. This same notion of fault tolerance is behind the unary representation fed to the integrator unit. The representation chosen places less weight on each individual signal; whereas a faulty signal on one bit of a binary input can change the input by 50% of full scale, one faulty signal on the unary input to the integrator can change the input by at most one level. The unary input also simplifies the programming of the integrator network. The integrator is a single layer device trained by a direct method. Each individual output element has two principal types of characteristics, namely rates and biases. For each output element, a net is computed by taking the weighted sum of the inputs to that output. This sum is added to the integrated value within the output element at a controlled rate. Whenever the integrated value within the element overrides the built in bias of the element, that element produces a one at its output.

3. FAULT TOLERANCE OF I^2t NETWORK TO INTENSE TRANSIENT NOISE

Cosmic radiation composed of high speed, high energy particles is referred to as single event radiation. Such radiation affects electronic equipment by creating extremely short duration disturbances within that equipment. The character of these transients depends on the distribution of ion atomic weights and linear energy transfer characteristics in the surrounding environment. Disturbances similar to those caused by single event radiation can be simulated in neural nets by presetting a probability that a given unit will be affected by a noise pulse in a given time period. For each time period a random number generator is used to determine for each processing element whether it will be disturbed. When an element is affected, its output is randomly set to either 0 or 1. Note that one way to increase tolerance to such disturbance is to use redundancy in the design of the circuitry. If an output is replicated two or more times, then the effect of transient disturbances can be reduced. Figure 2 shows the outcome of tests of the I^2t computing device. Shown are the input to the device for each test, the output of both the binary and the alarm devices for zero disturbances (dotted lines), and the output of both devices at a disturbance level of 800,000 expected disturbances per thousand processing elements per second. It can be seen that the disturbances degrade

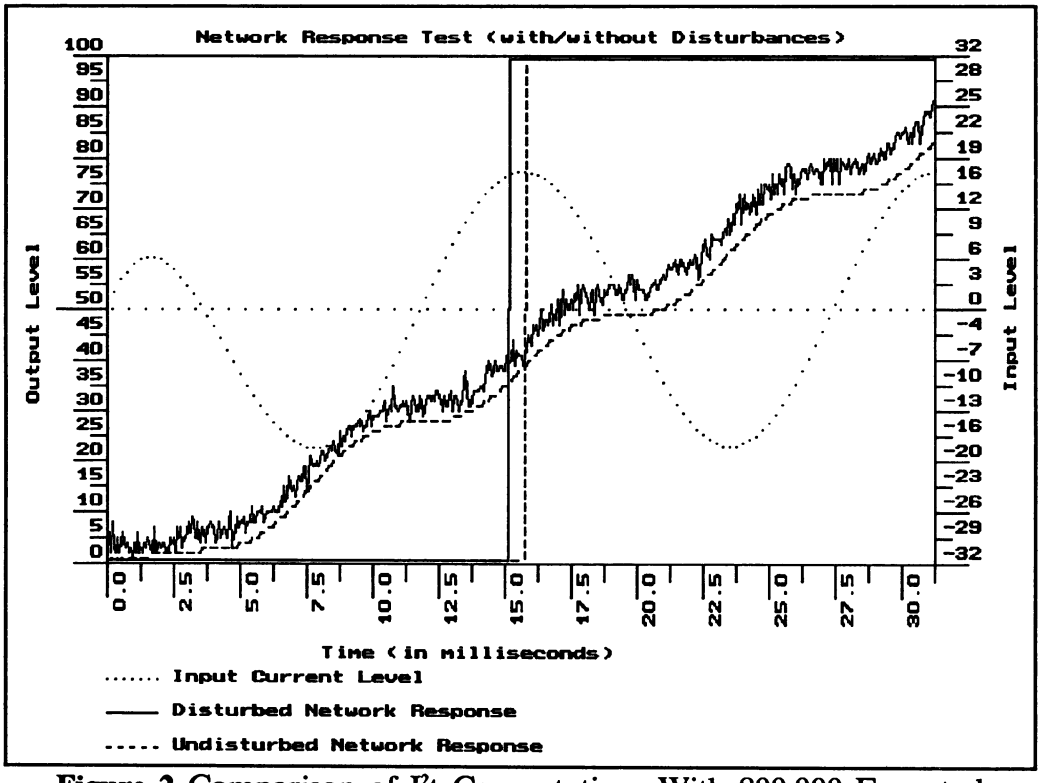


Figure 2 Comparison of I^2t Computations With 800,000 Expected Disturbances per Second per Thousand Elements

the output of the binary output network. It is interesting to note that the tendency of the output level of the disturbed integrator to rise above the output level of the undisturbed integrator is caused primarily by disturbances within the binary to unary absolute value converter, while the instantaneous spikes in the output level of the disturbed integrator are caused primarily by disturbances to the integrating layer itself. Also, the alarm device trips earlier under disturbed conditions than under undisturbed conditions. However, the alarm is still set off sufficiently close to the proper time as to allow for shutting down of appropriate sensitive components. Simulation results indicate that, in the case of the I^2t network, the alarm will, at worst, be set off prematurely. This behavior is desirable, as a premature trip condition is likely to be far less harmful than a delayed trip condition. Table I shows the behavior of the alarm device under a variety of disturbance conditions.

Table I Comparison of Trip Times of Disturbed I^2t Network

Disturbance Frequency (per second per 1000 elements)	Mean Trip Time (msec)	Standard Dev. (msec)
0	15.8	0.000
200,000	15.7	0.310
400,000	15.4	0.057
800,000	14.9	0.122
1,600,000	13.0	3.090

It was shown that the neural I^2t possesses significant fault tolerance to transient electrical disturbances which are potentially similar to the type of single event Space radiation of interest in Space electronic applications. This device, as well as other parallel computing blocks, is currently being studied to determine its behavior when exposed to simulated single event radiation phenomena as well as other types of radiation damage of interest in both Space and military applications.

REFERENCES

[1] Paul M. Anderson, James A. Martin, and Cindy Thomason,"Automated Power Distribution System Hardware," *Proceedings of the 24th Intersociety Energy Conversion Engineering Conference*, Washington, D. C.: August, 1989.

[2] James L. Dolce and Karl A. Faymon,"Automating the U.S. Space Station's electrical power system," *Optical Engineering*, Vol. 25 No. 11, November 1986.

[3] William K. Erickson and Peter C. Cheeseman,"Issues in the design of an executive controller shell for Space Station automation," *Optical Engineering*, Vol. 25 No. 11, November 1986.

[4] G. E. Hinton and T. J. Sejnowski,"Learning and Relearning in Boltzmann Machines," *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, Vol. 1, D. E. Rumelhart and J. L. McClelland (Eds.), Cambridge, MA: MIT Press, 1986.

[5] Terrence J. Sejnowski and Charles R. Rosenberg, "NETtalk: a parallel network that learns to read aloud, " *The John Hopkins University Electrical Engineering and Computer Science Technical Report*, JHU/EECS-86/01, 1986.

[6] W. F. Zimmerman, J. Bard, and A. Feinberg, "Space Station man-machine automation trade-off analysis," *Publication 85-13*, National Aeronautics and Space Administration, Jet Propulsion Laboratory, Pasadena, CA, 1985.

An Adaptive Neural Algorithm for Traveling Salesman Problem

X. Xu
W. T. Tsai

Department of Computer Science
University of Minnesota
Minneapolis, MN 55455

ABSTRACT

Recently, several researchers tried to solve Traveling Salesman Problem (TSP) using Hopfield's model (also called McCulloch-Pitts' model) (Grossberg, 1988; Hopfield, 1982, 1984; Hopfield & Tank, 1985; Foo & Szu, 1988; Wilson & Pawley, 1988). Unfortunately, the results obtained so far are not good enough compared to the ones obtained by conventional heuristic algorithms. This paper presents a new approach. It is based on a generalized neural network (Xu, Tsai, & Huang, 1988), and has a comparable performance with conventional heuristic algorithms. On 40 randomly generated planar TSP instances (each has 100 cities), with a confidence level $\alpha = 0.0069$, the algorithm improves the 2-OPT (25 runs) by at least 1.1%. On TSP instances with random distance matrix, the algorithm has drastic improvement over 2-OPT. The higher complexity of the generalized network is responsible for the power of the algorithm.

1. Introduction

Traveling Salesman Problem (TSP) has been studied extensively in the literature for many years. Recently, several researchers have tried a new approach to TSP (Hopfield & Tank, 1985; Foo & Szu, 1988; Wilson & Pawley, 1988), i.e., using Hopfield's model (also called McCulloch-Pitts' model (Grossberg, 1988)) to obtain optimal or near-optimal solutions. One possible advantage of a such approach is the utilization of massive parallelism of such neural networks. Unfortunately, the results obtained so far are less encouraging. In (Hopfield & Tank, 1985), even for a 30-city problem, the result produced is more than 17% longer than the one produced by a well-known heuristic algorithm (Lin & Kernighan, 1973). The performance is thus rather poor considering that many other heuristic algorithms can obtain solutions within few percentage of the best known solutions even for larger-size problems (Golden & Stewart, 1985).

In this paper, we present a new neural network algorithm, which has a competitive performance compared with conventional heuristic algorithms.

2. Algorithm

We first describe the coding scheme used in this algorithm. Like in (Hopfield & Tank, 1985), the neural network is organized as an $n \times n$ matrix (although half of the matrix is sufficient, using the entire matrix is more convenient), each neuron v_{xi} corresponds to the link from city x to city i. If $v_{xi} = 1$, then the solution will use link (x, i), otherwise link (x, i) is not present in the solution.

With the above coding scheme, one possible energy function is:

$$E = \frac{A}{2}\sum_{x=1}^{n}\left(\sum_{i=1}^{n} v_{xi} - 2\right)^2 + \frac{B}{2}\sum_{i=1}^{n}\left(\sum_{x=1}^{n} v_{xi} - 2\right)^2 + C\sum_{x=1}^{n-1}\sum_{i=x+1}^{n} v_{xi} d_{xi} \qquad (1)$$

with $v_{ii} = 0$, d_{xi} is the distance between city x and i.

the meaning of each term is clear, term A and B require that each city has two links in the solution, and

term C is the cost of the solution.

The above coding scheme is natural and easy to implement on neural networks. The essential difficulty with the above coding scheme is the subtour problem, i.e., a global minimum point of (4) may not correspond to a minimum-length Hamiltonian circle but to a set of smaller cycles. To overcome the subtour problem, we employ the so called generalized neural network (Xu, Tsai, & Huang, 1988), the generalized neural network allows more complex interactions among the neurons and thus is more powerful. Theoretically, the generalized neural network can guarantee to produce the global optimal solution, but at the cost of using an exponential number of coefficients (i.e., the size of the neural network is large), which is not practical for even small-size problem. However, by allowing only some specific interactions among the neurons, one can get measurable improvement from neural network defined by (1) without increasing the neural network size much. The technique used in the algorithm is called Adaptive Subtour Elimination.

2.1. Adaptive Subtour Elimination

While no effective mean is known to eliminate subtours under the coding scheme using Hopfield's neural network, there are many possible ways to eliminate subtours by using the generalized neural network. One solution would be adding to the energy function (1) all terms that correspond to all possible Hamiltonian tours, i.e., terms in the form:

$$-\sum_{\pi} v_{i_1 i_2} v_{i_2 i_3} \cdots v_{i_n i_1} \tag{2}$$

which is the total enumeration of all possible Hamiltonian tours, or equivalently, one can add all terms that correspond to all possible subtours (tours with fewer than n links), it is obvious that such enumeration is too expensive and impractical.

Although the above simple method of eliminating subtours has little practical usefulness, it can be useful if we do the partial enumeration instead of the total enumeration. This is justified by the following reasoning:

Suppose neural network 1 starts from a state S_i, it will converge to a stable state S_f which consists of several cycles, $CYL = \{ cyl_1, ..., cyl_k \}$, where:

$$cyl_i = (node_{j_{i_1}}, ..., node_{j_{i_l}}), \qquad i = 1, ..., k$$

If we add only the terms that correspond to those cycles to the energy function (1), we have a new neural network with energy function:

$$E = \frac{A}{2}\sum_{x=1}^{n}\Big(\sum_{i=1}^{n} v_{xi} - 2\Big)^2 + \frac{B}{2}\sum_{i=1}^{n}\Big(\sum_{x=1}^{n} v_{xi} - 2\Big)^2 + C\sum_{x=1}^{n-1}\sum_{i=x+1}^{n} v_{xi} d_{xi}$$

$$+ F\sum_{i=1}^{k} v_{j_{i_1} j_{i_2}} \cdots v_{j_{i_k} j_{i_1}} \tag{3}$$

Equation (3) defines a new neural network, if we start the new neural network again from S_i, it will converge to a stable state S'_f, the cycles in S_f will not appear in S'_f, since term F penalizes those subtours. However generaly, state S'_f will consist of a new set of subtours.

We can repeat the above procedure until we get a neural network that will produce a satisfactory result. This is called the adaptive subtour elimination since each time when we construct a new neural network, only terms that penalizes subtours of the immediate previous solution are added.

After the adaptive subtours elimination, there may still be a small number of subtours, those are merged by simple operations such as the ones in (Karp & Steele, 1985), and finally the resulting solution is improved by 2-OPT (Golden & Stewart, 1985).

3. Performance Analysis

Forty random planar TSP instances have been generated and tested on our algorithm (each problem has 100 cities), and the same instances are feed to an 2-OPT using random initial solutions, the results are summarized in Table 1. The results from 2-OPT are the best selected from 25 runs starting from random initial solutions. The reason to compare the algorithm to 2-OPT is that on some well-studied instances, 2-OPT has performed consistently within 3.3% of the optimal solution (Golden & Stewart, 1985).

Definition: Let A be an algorithm for TSP, the percentage of improvement of algorithm A over 2-OPT (25 runs) on an TSP instance is

$$pi(A) = \frac{Sol(2\text{-}OPT) - Sol(A)}{Sol(A)}$$

where $Sol(A)$ is the tour length obtained by algorithm A (on that instance).

Problem	$pi(our)$	Problem	$pi(our)$	Problem	$pi(our)$	Problem	$pi(our)$
1	-2.188	11	-0.789	31	1.704	41	2.690
2	3.080	12	0.361	32	1.791	42	0.309
3	7.170	13	2.021	33	-0.791	43	4.939
4	2.144	14	3.240	34	-1.122	44	0.518
5	1.178	15	2.828	35	1.668	45	1.323
6	2.239	16	-0.599	36	2.521	46	0.800
7	0.885	17	2.655	37	3.750	47	1.880
8	2.574	18	0.783	38	-1.785	48	2.319
9	3.778	19	1.110	39	4.342	49	3.792
10	1.202	20	4.319	40	0.831	50	4.278

Table 1 Percentage of improvement (of our algorithm) over 2-OPT.

Based on the data of Table 1, a large sample statistical test enables us to say that our algorithm improves 2-OPT (25 runs) by at least 1.1%, with a confidence level $\alpha = 0.0069$.

Due to the unavailability of the problem instances and the heuristic algorithms of (Golden & Stewart, 1985) (we have only problem # 24), a direct comparison between our algorithm and those heuristic algorithms cannot be made. We again select 2-OPT (25 runs) as a "standard" algorithm to compare with. H10 in Table 2 is the heuristic algorithm CCAO, i.e., Convex hull, Cheapest insertion, Angle selection and Or-OPT.

Based on the data in Table 2, we also made a statistical test on CCAO, and it revealed that CCAO improves 2-OPT (25 runs) by at least 1.1%, with only a confidence level $\alpha = 0.3707$.

From the two tests, one may get the impression that our algorithm is better than CCAO, this is rather illusionary, because the number of experiment data is too small in test two, while the first test is much accurate. It is much fairer to say that our algorithm is competitive with CCAO or other heuristic algorithms listed in Table 2.

Problem	24	25	26	27	28
$pi(H1)$	0.9686	1.5671	-0.5442	2.5216	0.7613
$pi(H2)$	0.1684	1.0889	-1.0728	1.2054	0.0194
$pi(H3)$	0.7373	1.5671	-0.5442	2.9098	0.7613
$pi(H4)$	0.2976	1.5871	-0.0199	1.5038	3.0545
$pi(H5)$	-1.3657	1.6272	-1.0046	3.0947	0.2038
$pi(H6)$	-0.3057	1.5171	-2.0084	2.1161	1.6242
$pi(H7)$	0.5469	1.6974	-0.6818	2.5929	2.1571
$pi(H8)$	-0.0593	-0.0388	-0.0696	0.9087	0.6238
$pi(H9)$	0.6470	1.4571	0.0199	2.8074	2.2381
$pi(H10)$	1.1100	2.0600	0.0100	2.2779	0.6827

Table 2 Percentage of improvement over 2-OPT (From Golden & Stewart, 1985).

4. Conclusions

In this paper we presented a neural network algorithm with Adaptive Subtour Elimination. The algorithm is based on a generalized neural network model (Xu, Tsai, & Huang, 1988). The results of those neural networks are post-processed by a simple iterative algorithm, known as 2-OPT. By using the generalized neural networks, our algorithm has produced comparable performance with other conventional heuristic algorithms on TSP. From the experiment data, we can say, with confidence level $\alpha = 0.0069$, that our algorithm is better than 2-OPT (25 runs) by at least 1.1%. This is a good result considering that on 5 TSP instances, the 2-OPT (25 runs) has produced solution within 3.3% over the optimal solution (Golden & Stewart, 1985). By demonstrating that neural network can produce competitive results with conventional heuristic algorithms, this paper indicates that the neural network approach is indeed a viable alternative to solve hard optimization problem.

5. Reference

Foo, Y. S., & Szu, H. (1988). "Binary neurons with analog communication links for solving large-scale optimization problems", *Abstracts of the First Annul INNS Meeting*, Vol. 1, Supplement 1, pp. 437.

Golden, B. L., & Stewart, W. R. (1985). "Empirical analysis of heuristics," in E. L. Lawler et al, eds., The Traveling Salesman Problem, Chapter 7, pp. 207-249.

Grossberg, S. (1988). "Nonlinear neural networks: principles, mechanisms, and architectures", *Neural Network*, Vol. 1., pp. 17-61.

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.

Karp, R. M., & Steele, J. M. (1985). "Probabilistic analysis of heuristics," in E. L. Lawler et al, eds., The Traveling Salesman Problem, Chapter 6, pp. 181-206.

Lin, S, & Kernighan, B. W. (1973). "An effective heuristic algorithm for the traveling salesman problem", *Operations Research*, Vol. 21, pp. 498-516.

Wilson, G. V., & Pawley, G. S. (1988). "On the stability of the traveling salesman problem algorithm of hopfield and tank", *Biological Cybernetics*, Vol. 58, pp. 63-70.

Xu, X., Tsai, W. T., & Huang, N. K. (1988). "A generalized neural network model" *Abstracts of the First Annul INNS Meeting*, Vol. 1, Supplement 1, pp. 150. Also available as CSci TR 88-30, Department of Computer Science, University of Minnesota.

Fuzzy rule realization on associative memory system

Toru Yamaguchi*, Naoki Imasaki**, Kazuhito Haruki**

* LIFE : Laboratory for International Fuzzy Engineering research
Siber Hegner Buildihg 4FL., 89-1 Yamashita-cho,
Naka-ku, Yokohama-shi, 231 JAPAN
** Systems & Software Engineering Laboratory, Toshiba Corp.
70, Yanagi-cho, Saiwai-ku, Kawasaki-shi, 210 JAPAN

1. Introduction

Fuzzy control has been known as a successful application of fuzzy theory(1), and is used for many real systems(2). Fuzzy control realization needs the three steps; 1) we acquire the fragment of expert's uncertain knowledge, 2) write the knowledge in fuzzy rules, and 3) refine the fuzzy rules. To support the latter two steps, we proposed SNN-AM (Structured Neural Network - Associative Memory system)(3). SNN-AM can infer suitable rules from the expert's uncertain knowledge with its associative function and can refine fuzzy rules with a special learning method. We use a short-term memory system (STM) for inference and a long-term memory system (LTM) for learning. We also use a structured neural network architecture which is advancement of BAM (Bidirectional Associative Memory)(4) architecture. SNN-AM's inference might activate some stored memories which fit for an input condition. If the input condition isn't suitable for any stored conditions, SNN-AM makes up a rule image suitable for the input condition from stored rules. SNN-AM's learning is fast, because neural networks in the structured neural networks are separately trained.

In this paper,we first propose SNN-AM, and discuss its architecture and the framework of STM and LTM. Second, we discuss how the fuzzy rules are stored in SNN-AM, how the suitable rules are inferred from the stored fuzzy rules, and how the stored rules are refined. Finally, we reports the elevator group control application, which uses SNN-AM in predicting control effects, and the usefulness of SNN-AM.

2. SNN-AM: Structured Neural Network - Associative Memory system

SNN-AM memorizes concepts given by expert's uncertain knowledge on a bidirectional associative network, which connects structured neural networks.

SNN-AM's architecture is shown in Fig.1. It consists of structured neural networks S0~SN. (It is possible to represent hierarchical concepts by applying the same architectures to S0~SN.) S0 has a concept name set which consists of concept named neurons, and S1~SN have attribute name sets which consist of neurons characterizing the concepts in S0. S0 is connected to S1~SN in respect to special uncertain relationships. These connections are constructed by BAM. BAMs in SNN-AM activate the stored concepts which fit for an input condition.

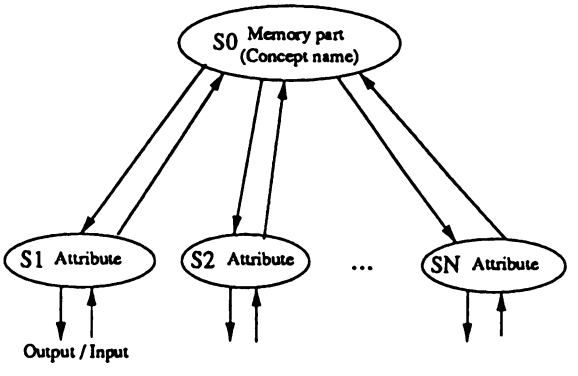


Fig.1 Concept representation

There is a special architecture that a concept set in S0 conducts attribute sets in S1~SN, and its architecture makes the network dynamics stable because the major macro factor (concept set) drives micro factors' (attribute sets') actions to be stable. This architecture is well known as a holonic system(5). It will be mentioned below that SNN-AM has an inference function and a learning function.

STM (short-term memory system) and inference

STM has an inference function which temporarily keeps the network dynamics stable. The series of active states' transition explains an infering transition with some meanings. In this transition, the most fitting concept's activation value gradually increases toward the highest level, while other concepts' activation values decrease. SNN-AM controls this activation values. To imitate some fuzzy inference methods which are used in fuzzy control, SNN-AM stops the transition before the state sufficiently converges. SNN-AM's inference activates some concepts fitting for the input condition with its certainty factor, as the fuzzy inference does.

LTM (long-term memory system) and learning

LTM consists of the connection weights between S0~SN and the connection weights in each network S0~SN. These connection weights are refined by following two learning methods.
The learning method at the connection weights between S0 and S1~SN is based on an idea similar to Hebbian learning law, and is shown as follows.

$$W_{AC} = -W_{AC} + X_C \cdot R_A , \quad (1)$$

where, W_{AC} is the connection weight between a concept C and an attribute A; if the concept C has a relation to the attribute A, X_C's value is 1, and otherwise, X_C's value is 0; R_A is A's certainty factor to attribute A's labeled meaning. We arrange Eq.(1) for A's independent and discrete timing learning and the representation of the learning times k on considering Eq.(1)'s learning curve, and make Eq.(2).

$$\Delta W_{AC}(k) = D_1[\exp(-T(k-1))-\exp(-T(k))] , \quad (2)$$

where, D_1 is a coefficient, T(k) means the k-th learning time.
The B.P. learning method(6) can be used as the learning method for the connections in S0~SN. Other methods (like L.V.Q, etc.) are also applicable as the learning method. SNN-AM's learning can be fast because S0~SN are trained independently. In LTM, the learning effect is changed in proportion to the matching grade between attribute A's meaning and its learning data, and inversely proportion to the rate of learning about attribute A.

3. Fuzzy rule representation

Fig.2 shows fuzzy if-then rules, which corresponds to Fig.1. Fuzzy rules are memorized in S0 as concepts. These fuzzy rules' concepts have "if" and "then" attributes. S1 is "if" attribute set. And S2 is "then" attribute set. Fuzzy model(7), which has high representation ability, is formed by SNN-AM. Ordinary fuzzy model rules are written as

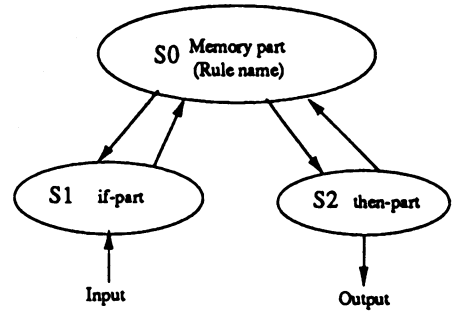


Fig.2 Fuzzy rule representation

$$\text{Rule i:If } ((u1=Ci1)\&\cdots\&(uj=Cij)\&\cdots\&(ur=Cir)) \text{ then } yi=fi(u), \quad (j=1\sim r,\ i=1\sim N), \quad (3)$$

where, $u=[u1,\cdots,ur]^T$ is a input vector, Cij is a fuzzy set like "Big","Medium", or "Small", fuzzy set are characterized by membership functions which represent each label's meaning, yi is a output vector. fi(·) is ordinarly a linear function which shows a sub-model. We use neural networks for sub-models, so sub-models are non-linear function in this paper. Fuzzy model is useful for representing structured neural network model.

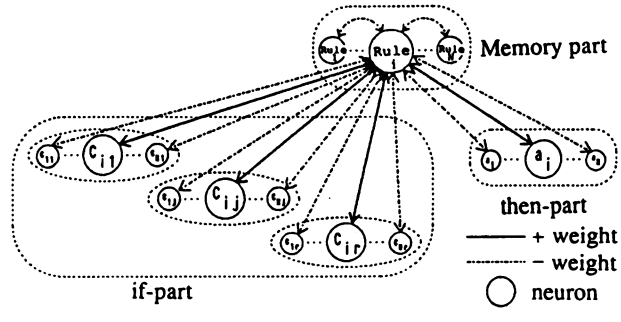


Fig.3 Memorizing method

This fuzzy model's rule are represented in Fig.3, where we explain how to memorize a fuzzy rule on Rule i. Each circle shows a neuron which represents each structured neural network. The memory part is at the center of SNN-AM, and connected to if-part and then-part. Positive relations have plus weighted bidirectional connections, and negative relations have minus weighted connections. No connection if there's no relation. In the memory part, rule neurons have minus weighted connections each other.

SNN-AM's fuzzy inference dymanics is shown as following discrete equations.

$$\begin{aligned} v(k+1)&=f\{Wvc\cdot c(u(k))+Wvv\cdot v(k)+Wvx\cdot x(k)\},\\ x(k+1)&=f\{Wxv\cdot v(k+1)+Wxx\cdot x(k)+Wxz\cdot z(k)\}\ ,\\ z(k+1)&=f\{Wzx\cdot x(k+1)+Wzz\cdot z(k)\}, \end{aligned} \tag{4}$$

where, v, x, and z are neuron's states, v is a r-vector for the if-part, x is a N-vector for the memory part, z is N-vector for the then-part, $f\{\cdot\}$ is a sigmoid function, $c(\cdot)$ shows membership functions Cij for input u, Wij shows weight matrix from nodes i to nodes j.
When T is the inference transition stopping time, the output activation vector of then-part is shown as a(u)=z(T). The output vector y is given by

$$y=\left(\sum_{i=1}^{N} ai(u)\cdot yi\right)/\left(\sum_{i=1}^{N} ai(u)\right)\ . \tag{5}$$

SNN-AM's learning at Wxz and Wzx is given by follows from Eq.(2).

$$\Delta Wij=\begin{cases} +D_1[\exp(-D_2(Ti(k-1)+D_3))-\exp(-D_2(Ti(k)+D_3))]\ , & (i=j)\\ -D_1[\exp(-D_2(Ti(k-1)+D_3))-\exp(-D_2(Ti(k)+D_3))]\ , & (i\neq j) \end{cases} \tag{6}$$

$$\Delta Ti(k)=Ti(k)-Ti(k-1)=\min\left(1, D_4\left(\sum_{i=1}^{N} ai(u)/Wii\right)\right)\ , \tag{7}$$

where, D_1, D_2, D_3 and D_4 are coefficients. Eq.(7) represents the learning effect feature described in LTM.

4. Application to elevator group control system

The fuzzy rules designed for elevator group control are

$$\begin{aligned} &\text{Rule 1:If d=BIG} && \text{then } y1=f_B(x),\\ &\text{Rule 2:If d=MEDIUM} && \text{then } y2=f_M(x),\\ &\text{Rule 3:If d=SMALL} && \text{then } y3=f_S(x), \end{aligned} \tag{8}$$

where, BIG, MEDIUM and SMALL are membership functions, $f_B(\cdot)$, $f_M(\cdot)$ and $f_S(\cdot)$ are sub-models which show predictive effects, yi and x are 3dim-vector. Eq.(8) shows expert's rules where we get predictive control effects yi when the demand is d and control parameter is x. Output y is given by Eq.(5).
In this application explained in Fig.4, sub-models f_B and f_S are certain but f_M is uncertain at the beginning.

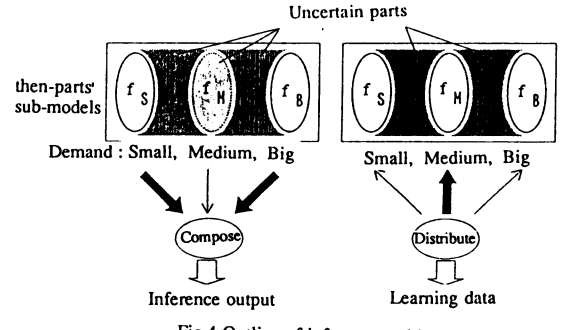
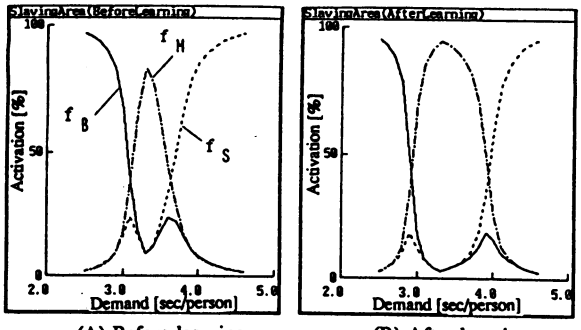


Fig.4 Outline of inference and learning

If an inference at the condition d=MEDIUM is done, output y is given by f_M, f_B, and f_S because f_M isn't sufficiently trained. The f_M's activation area becomes wider by the learning, as shown in Fig.5. Fig.6 shows that the error rate is decreased by the learning. The transition of active state rates at inference is shown in Fig.7.



(A) Before learning (B) After learning

Fig.5 Active states for demands

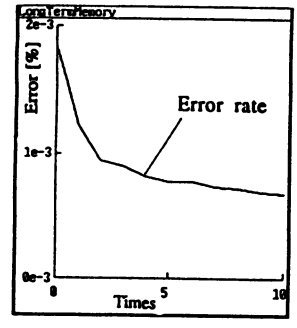


Fig.6 Error transition at learning

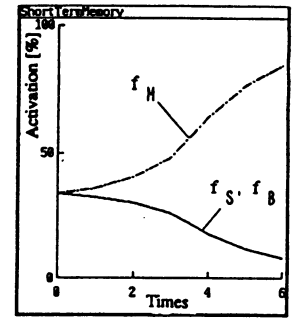


Fig.7 Active state transition at inference

5. Conclusion

This paper presented how to realize fuzzy rules within an associative memory system, how to infer suitable memorized rules, and how to refine the memorized rules.

REFERENCES

(1)Zadeh,L.A.: Outline of a New Approach to the Analysis of Complex Systems and Decision Processes, IEEE Trans. SMC-3-1, pp.28-44 (1973).

(2)Yamaguchi,T.,Endo,T. and Haruki,K.: Fuzzy Predict and Control Method and Its Application, IEE Int. Conf. CONTROL88 C.P. No.285, pp.287-292 (1988).

(3)Yamaguchi,T.,Imasaki,N.,Haruki,K.: Learning Fuzzy Neural Network and its application, SICE 9th Knowledge engineering symposium, pp.1-6 (1989) (in Japanese).

(4)Kosko,B.: Adaptive Bidirectional Associative Memories, Applied Optics, Vol.26, No.23, pp.4947-4960 (1987).

(5)Shimizu,H.,Yamaguchi,Y.,Tsuda,I. and Yano,M.: Pattern Recognition Based on Holonic Information Dynamics: Towards Synergetic Computers, Complex System - Operational Approaches, pp.225-239 (1985).

(6)Rumelhart,D.E.,Hinton,G.E. and Williams,R.J.: Learning Internal Representations by Error Propagation, Parallel Distributed Processing, PP.318-362 (1986).

(7)Takagi,T. and Sugeno,M.: Fuzzy Identification of Systems and Its Applications to Modeling and Control, IEEE SMC-15, pp.116-132 (1985).

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)

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