ISBN 0 444 8874

Artificial Neural Networks and
Statistical Pattern Recognition

621.

NH

4 88741 5

Artificial
Networ
Statistical I
Recog

Machine

Pattern Re

Machine Intelligence and Pattern Recognition

Volume 11

Series Editors

L.N. KANAL

and

A. ROSENFELD

University of Maryland
College Park, Maryland, U S A

NORTH-HOLLAND
AMSTERDAM • LONDON • NEW YORK • TOKYO

A

St

Ol

Edit

Ishv
Way
Detr

Ani
Mic
East

199

NO
AM

Artificial Neural Networks and Statistical Pattern Recognition

Old and New Connections

Edited by

Ishwar K. SETHI
Wayne State University
Detroit, Michigan, U S A

Anil K. JAIN
Michigan State University
East Lansing, Michigan, U S A

1991

NORTH-HOLLAND
AMSTERDAM • LONDON • NEW YORK • TOKYO

ELSEVIER SCIENCE PUBLISHERS B V
Sara Burgerhartstraat 25
PO Box 211, 1000 AE Amsterdam, The Netherlands

Distributors for the United States and Canada:
ELSEVIER SCIENCE PUBLISHING COMPANY, INC
655 Avenue of the Americas
New York, N Y 10010, U S A

Library of Congress Cataloging-in-Publication Data

```
Artificial neural networks and statistical pattern recognition : old
  and new connections / edited by Ishwar K  Sethi, Anil K. Jain.
      p    cm. -- (Machine intelligence and pattern recognition ; v.
  11)
    Includes bibliographical references and index
    ISBN 0-444-88740-7  -- ISBN 0-444-88741-5 (pbk.)
    1  Pattern recognition systems   2  Neural networks (Computer
  science)   I  Sethi, Ishwar K., 1948-    . II. Jain. Anil K., 1948-
     III. Series.
  TK7882 P3A7  1991
  006.3--dc20                                            91-24621
                                                            CIP
```

ISBN: 0 444 88740 7 (hardbound)
0 444 88741 5 (paperback)

Printed in The Netherlands

FOREWORD

The first time I became interested in Neural Nets and Statistical Pattern Recognition was in early 1958 while I was a graduate student in the Moore School of Electrical Engineering of the University of Pennsylvania. My student subscription to the NEW YORKER magazine brought many chuckles from cartoons and stories but the only item from all those many issues that has stayed with me was a column in the December 6, 1958 issue titled "Rival" This covered an interview with Frank Rosenblatt in which he described his hopes for his "artificial intelligences" which would rival humans in perception and problem solving. By the time I read this column I knew a fair amount about Rosenblatt's research on Perceptrons, since as part of a machine learning and recognition research project and a search for a dissertation topic, I had spent much time pouring over his Cornell Aeronautical Laboratory reports. I had also read parts of a book *Stochastic Models for Learning* by Bush and Mosteller (Wiley, 1955) and been studying papers on Statistical Discrimination, in particular papers by C R. Rao and the chapter on Problems of Discrimination in his book *Advanced Statistical Methods in Biometric Research* (Wiley , 1952). About the same time Robert Bush joined the University of Pennsylvania as chairman of Psychology. I chose Bush as my dissertation advisor, and with some support from R. Duncan Luce did a dissertation (for the Ph.D in electrical engineering!) on the analysis of some stochastic processes arising from Luce's nonlinear "Beta" model for learning. This is how learning models, artificial neural networks, and statistical pattern classification came together in my cognizance.

Two years later, when I joined General Dynamics/Electronics (GD/E) in Rochester, New York, as Manager of the Machine Intelligence Advanced Development Laboratory, it seemed as though every company and university laboratory was working on perceptron type machines. At GD/E we also implemented our own version of an adaptive pattern recognizer which was soon called APE (Adaptive Pattern Encoder). There were many other learning machines implemented by various organizations, machines with names such as MINOS, SOCRATES, and of course ADALINE and MADALINE. It was a time for catchy names and audacious claims [see Kanal, Proc IEEE, October 1972]. Clearly PERCEPTRON and ADALINE were the key innovations and they had memorable names, although I have it on good authority that in the 1980' s when the new machine vision company Perceptron was formed, its founders had no idea that the name they had come up with had a previous incarnation. Because of simultaneous exposure to papers on learning models, perceptrons, and statistical discrimination, my attempts at understanding perceptrons and other "bionic" networks were formulated in terms of statistical classification methods, stochastic approximation procedures and stochastic models for learning "Evaluation of a class of Pattern Recognition Networks" presented at the Bionics conference in Ithaca, N.Y in 1961 and reprinted in this book, summarized some of that understanding. It may seem surprising now, but at that time it had been stated by some of the well known researchers writing in the engineering literature on pattern recognition, that the use of a weighted sum of binary variables as done in the perceptron type classification function limited the variables to being statistically independent.

Rosenblatt had not limited himself to using just a single Threshold Logic Unit but used networks of such units. The problem was how to train multilayer perceptron networks. A paper on the topic written by Block, Knight and Rosenblatt was murky indeed, and did not demonstrate a convergent procedure to train such networks. In 1962-63 at Philco-Ford, seeking a systematic approach to designing layered classification nets, we decided to use a hierarchy of threshold logic units with a first layer of "feature logics" which were threshold logic units on overlapping receptive fields of the image, feeding two additional levels of weighted threshold logic decision units. The weights in each level of the hierarchy were estimated using statistical methods rather than iterative training procedures [L.N. Kanal & N.C. Randall, Recognition System Design by Statistical Analysis, Proc. 19th Conf. A.C.M,1964]. We referred to the networks as two layer networks since we did not count the input as a layer On a project to recognize tanks in aerial photography, the method worked well enough in practice that the U.S. Army agency sponsoring the project decided to classify the final reports, although previously the project had been unclassified. We were unable to publish the classified results! Then, enamoured by the claimed promise of coherent optical filtering as a parallel implementation for automatic target recognition, the funding we had been promised was diverted away from our electro-optical implementation to a coherent optical filtering group. Some years later we presented the arguments favoring our approach, compared to optical implementations and trainable systems, in an article titled "Systems Considerations for Automatic Imagery Screening" by T.J.Harley, L.N. Kanal and N.C. Randall, which is included in the IEEE Press reprint volume titled *Machine Recognition of Patterns* edited by A. Agrawala. In the years which followed multilevel statistically designed classifiers and A.I search procedures applied to pattern recognition held my interest, although comments in my 1974 survey,"Patterns In Pattern Recognition: 1968-1974" [IEEE Trans. on IT, 1974], mention papers by Amari and others and show an awareness that neural networks and biologically motivated automata were making a comeback.

In the last few years trainable multilayer neural networks have returned to dominate research in pattern recognition and this time there is potential for gaining much greater insight into their systematic design and performance analysis. Artificial neural networks trained on sample data are nonparametric statistical estimators for densities and classifiers. This leads to many questions about ANN's in comparison to alternate statistical methodologies. Such questions include the information requirements for each approach, the sample sizes for design and test, the robustness to incomplete data and different types of noise, and the generalization capability of competing procedures Additional points of comparison concern the relations of the sizes of feature vectors for each pattern category; the capability for variable-length vector pattern recognition; the capability for fusion of multiple sources or sensors; the ability to incorporate domain knowledge; the ability to work with other pattern recognition paradigms in an integrated fashion; the ability of the methodology to extend to other types of problem solving, e.g., combinatorial optimization, resource allocation, etc., using the same general network architecture; the suitability for easy mapping to VLSI or other parallel architecture. The capability of neural networks to combine adaptation with parallelism in an easy and natural fashion and the ability of learning continuously while working on a problem in a real environment are of particular interest. Finally, the cost of implementation and of training personnel in the methodology will also be determiners of comparative success.

Some of the above questions are beginningto be addressed in the literature and the present volume is also a good start in this direction. I am thankful to Professors Anil K. Jain and Ishwar K. Sethi for their initiative in assembling and editing this volume and to the authors of each chapter for their contribution to this volume. The richness of the artificial neural network paradigm for pattern recognition ensures that, despite the many individuals working in this area, much work remains to be done to gain a true understanding of ANN methodologies and their relation to better understood pattern recognition methods I expect that additional volumes will be assembled and published in this book series on the subject of artificial neural networks and their relation to and interaction with statistical pattern recognition, genetic algorithms, expert systems, and other approaches to the machine recognition of patterns

Laveen N. Kanal
College Park, MD

PREFACE

Artificial neural networks (ANNs) are currently enjoying tremendous popularity across many disciplines. The interest in artificial neural networks is not new; it dates back to the work of McCulloch and Pitts, who, about forty years ago, proposed an abstract model of living nerve cells or neurons. Since then, the field of ANN has charted a bumpy course, with expectations running high in the late 50's with the publication of Rosenblatt's Perceptron model, and going down in the late 60's with the publication of Minsky and Pappart's book, "Perceptrons". While isolated interest in ANNs continued thereafter, a resurgence of interest came in the early 80's, with the work of Hopfield and his associates as well as that of Rummelhart and the parallel distributed processing (PDP) research group.

Two broad groups of researchers have been drawn to the study of artificial neural networks. The first group of researchers mainly consists of scientists who are interested in obtaining answers to some fundamental issues of perception, learning, and memory organization in the human brain through the study and development of different neuron models The artificial neural network models espoused by this group of researchers are required to be as close as possible to the biological neural networks The second group of ANN researchers is drawn mainly from the engineering community These researchers are interested in exploiting the learning and parallel processing capabilities of the ANN to build engineering applications This set of researchers takes a pragmatic approach towards various ANN models; they are not overly concerned about the closeness of the artificial neural systems with their biological counterparts

Pattern recognition applications have emerged as the main beneficiary of the recent developments in ANNs. Pattern recognition tasks such as recognizing a familiar face or voice, identifying objects around us, or noticing relationships in a set of observations that we perform so effortlessly, have proven to be difficult in unconstrained settings for the traditional algorithmic approach, even using very powerful computers. In this respect, ANNs, with their self-organizing and non-algorithmic learning characteristics, offer a great deal of potential for pattern recognition applications.

The pattern recognition related activities using ANNs can be broadly grouped into two categories. The first group of activities consists of using the discriminatory or self-organizing features of various ANN models, such as multilayer perceptrons, neocognitron, ART series, Kohonen's self-organizing feature maps, etc to build systems for recognizing different kinds of shapes, sounds and textures Many such efforts have led to performance levels that are comparable or superior to the existing levels of performance achieved by traditional pattern recognition methods. The second group of pattern recognition related activities centers around mapping traditional pattern classifiers into ANN architectures The aim of such mappings is to utilize some of the key features of ANN models to obtain better classification performance in terms of speed or error rate or both. Most of the commonly used classifiers, such as linear classifiers, quadratic classifiers, tree classifiers, nearest neighbor classifiers, have been exactly or approximately mapped into layered ANN architectures.

With the growing complexity of pattern recognition related problems which are being solved using ANNs, many ANN researchers are grappling with design issues such as the size of the network, the number of training patterns, and performance assessment and bounds These researchers are discovering that many of the learning procedures lack the scaling property, i.e these procedures simply fail or produce unsatisfactory results when applied to problems of bigger size. Phenomena like these are very familiar to researchers in statistical pattern recognition (SPR), where the "curse of dimensionality" is a well-known problem. Issues related to the training and test sample sizes, feature space dimensionality, and the discriminatory power of different classifier types have all been extensively studied in the SPR literature. It appears that many ANN researchers looking at pattern recognition problems are not aware of the ties between their field and SPR, and are therefore unable to successfully exploit the past work that has been done in SPR. Similarly, many pattern recognition and computer vision researchers do not realize the potential of the ANN approach to solve problems such as feature extraction, segmentation, and object recognition.

It is in the context of the above remarks that the idea for this volume originated; we are now delighted to share it with you There are thirteen chapters in this volume, organized into three groups The theme of the 5 chapters in the first group revolves around the connections between ANNs and SPR. Familiarity with work in each of these areas can lead to mutual benefit, as the study of ANNs and SPR share many common goals The first chapter in this group is a paper by Kanal that was originally published in 1961 It is included here because it is one of the first papers, if not the first, to discuss the relationship between perceptrons and statistical classification methods, and also to relate learning algorithms to stochastic approximation methods. The second chapter, by Werbos, provides an overview of artificial neural networks research, especially the back-propagation algorithm for static as well as dynamic situations, and its linkage with statistical pattern recognition. The third chapter in the first group is by Raudys and Jain, who investigate the performance of artificial neural networks designed using only a small set of exemplars. The next two chapters, by Gelfand and Delp, and Sethi, deal with the relationship between tree classifiers and multiple layer perceptron networks

The second group of 5 chapters is devoted to the application of neural networks to various pattern recognition problems involving image and speech data. The first chapter in this group, by Lee, Srihari and Gaborski, provides a theoretical relationship and an empirical comparison between the Bayes decision rule and the back-propagation network, using the problem of handwritten character recognition In the second chapter of this group, Khotanzad and Lu examine the use of multiple layer perceptron networks for shape and texture recognition, and compare the performance of neural net classifiers with some conventional classifiers The third chapter in the application group is by Ghosh and Bovik, who highlight the relationships between conventional and neural techniques for processing of textured images, and suggest discrete 2-dimensional Gabor transforms using a neural network implementation. The next chapter in this group, by Rangarajan, Chellappa and Manjunath looks at the relationship between the Markov random fields and neural networks. This relationship is examined in the context of early or low-level vision processing, suggesting some applications that might benefit from an approach that combines Markov random fields and neural networks The last chapter in the application group is by Bengio and De Mori. After surveying the application of different neural models to automatic speech recognition, Bengio and De Mori present the details of using radial basis functions network for a particular speech recognition task One consistent conclusion in all the five chapters of the application group is that neural net classifiers can serve as a good alternative to conventional classifiers The maximum advantage in the use of neural classifiers occurs when the data are noisy and a large number of training samples are available

The third section of the book deals with the implementation aspects of artificial neural networks. While the most widely used neural network implementations today are software simulators, it goes without saying that the full advantage of neural network paradigm cannot be reaped without hardware implementations. The first chapter in this section, by Hassoun, deals with the architecture of dynamic associative memories, a class of ANNs which utilizes supervised learning procedures to store information as stable memory states Optical implementations of ANNs hold great promise The second chapter, by Vijaya Kumar and Wong, describes such implementations for four associative memory schemes The third and final chapter of the implementation section is on the VLSI implementation of neural networks Salam, Choi and Wang provide an overview of the various issues related to the silicon implementation of neural nets Some notable successes have already been achieved in the silicon implementation of biological devices, the most notable being the silicon retina and cochlea, by Mead and his group, at CalTech According to Mead, analog VLSI neural networks running at 10 quadrillion operations per second are ultimately achievable.

This book could not have been completed without the whole-hearted cooperation of all the authors; we are thankful to them. We are also grateful to Professor Laveen Kanal for his constant encouragement and suggestions We are also thankful to the Plenum Publishing Corporation for its kind permission to reprint Laveen Kanal's paper Our sincere thanks also goes to the staff at Elsevier Science Publishers B.V. for their cooperation and support. We would consider our goal accomplished if this book is able to contribute in some way to greater interaction between the two communities of ANN and SPR researchers. It is to them that this book is dedicated

Ishwar K. Sethi
Detroit, MI

Anil K Jain
East Lansing, MI

TABLE OF CONTENTS

Artificial Neural Networks and Statistical Pattern Recognition
Old and New Connections
I.K. Sethi and A.K. Jain (Editors)
1991 Elsevier Science Publishers B V.

EVALUATION OF A CLASS OF PATTERN-RECOGNITION NETWORKS

Laveen Kanal

General Dynamics/Electronics, Rochester, New York

INTRODUCTION

The realization of devices to perform a given pattern-recognition task can be considered in terms of the problem of providing the following specifications.

A. The observables (measurements or tests) x_i, $i = 1,2,\ldots,N$, by which patterns are to be characterized.

B. The form of the classification function, i.e., the manner in which observables are to be used in assigning a pattern to one of K known groups.

C. The procedures for determining the classification function from samples of patterns from the K different groups.

In any given instance the results of the measurements may be represented by $x = (x_1, x_2, \ldots, x_n)$. Then the universe of patterns can be thought of as being an N-dimensional space and the recognition task becomes one of dividing this N-dimensional space into mutually exclusive regions R_j, $j = 1,2,\ldots,K$, such that when x falls in R_j, the pattern is listed under group j.

Unless an especially auspicious choice has been made of the N characteristics which define the observation space, the x_i will, in general, have to be treated as stochastic variables. The recognition task then becomes the application of statistical inference to the classification of a pattern to one of the K known groups to which it can possibly belong. Information on the probability distributions of observables for the various groups can range from complete ignorance of the functional form of the distribution, to the case where the functional form and all parameters are known.

1. A CLASS OF PATTERN-RECOGNITION NETWORKS

In some recent articles (see, for example, Hawkins, 1961), work on the application of a class of networks to pattern recognition has been reported. Figures 1 and 2 show two networks typical of this class. The measurements x_i which

Previously published in Biological Prototypes and Synthetic Systems (Volume 1) edited by Eugene E Bernard and Morley R. Kare, Plenum Press, New York, 1962

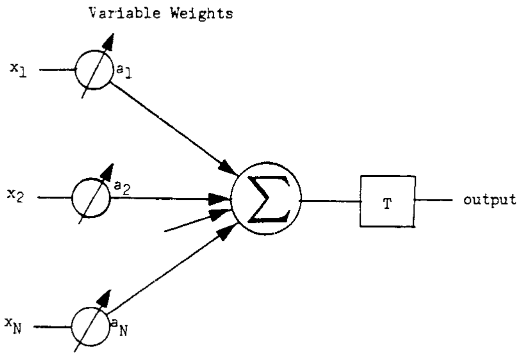


Fig. 1. Example of a neuron-type pattern-recognition network; see equation (1).

characterize the pattern are obtained, for instance, by placing the pattern on an "artificial retina" with the outputs of the retina elements being quantized such that the x_i are either 0 or 1.

In Fig. 1 the classification function takes the form of a weighted sum of the x_i, viz.,

$$\sum_{i=1}^{N} a_{ij} x_i > T_j \qquad j = 1,2,\ldots,K \tag{1}$$

where the set of coefficients a_{ij} which is desired is the one which permits the threshold T_j to be exceeded whenever a pattern from group j is present and not otherwise.

In the network of Fig. 2, subsets of the x_i, selected perhaps in a random manner, are connected through fixed weights (+1, −1) to summation units with thresholds. Let b_{im} be the fixed weights between the retina elements and the summation units, where b_{im} can be 0. Let I_m be the thresholds for the summation units, and y_m be the outputs from the summation units, with y_m being 0 when the thresholds I_m are not exceeded, and 1 otherwise. Further, let a_{mj} be weighting coefficients (variable) between summation units and response units and let T_j be the thresholds for the response units. Then the classification function used by the network of Fig. 2 is

$$\sum_{m=1}^{M} a_{mj} y_m > T_j \qquad j = 1,2,\ldots,K \qquad M < N \tag{2}$$

with

$$y_m = \begin{cases} 1 \text{ if } \displaystyle\sum_{i=1}^{N} b_{im} x_i > I_m \\ 0 \text{ otherwise} \end{cases}$$

Although the network of Fig. 2 uses the derived variables y_m to characterize the patterns, the form of the classification function is the same as that for the network of Fig. 1. An evaluation of this class of networks can be considered in

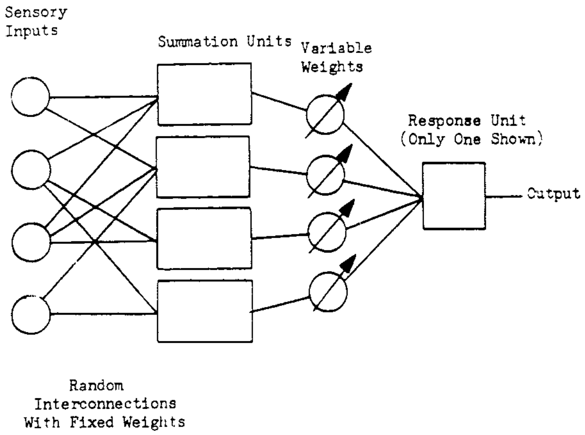

Sensory Inputs
Summation Units
Variable Weights
Response Unit (Only One Shown)
Output
Random Interconnections With Fixed Weights

Fig. 2. Perceptron-type pattern-recognition network (after Hawkins); see equation (2).

the context of comparable classification functions which can be derived from some more or less formal principles proposed in classification theory and practice.

2. A REPRESENTATION OF THE JOINT DISTRIBUTION

Let X denote the set of all points $x = (x_1, x_2, \ldots, x_N)$ with each $x_i = 0$ or 1, and let $p(x) = p(x_1, x_2, \ldots, x_N)$ denote the joint probability of the x_i in a given group. Since there are 2^N points in X, any parametric description of an arbitrary probability distribution will, in general, require $(2^N - 1)$ independent parameters. A particular parametric representation due to Bahadur (1959b) is used here.

Using E_p to denote the expected value when the underlying distribution is $p(x)$, define for each $i = 1, 2, \ldots, N$,

$$m_i = p(x_i = 1) = E_p(x_i) \qquad 0 < m_i < 1$$

$$z_i = \frac{x_i - m_i}{\sqrt{m_i(1 - m_i)}}$$

$$r_{ij} = E_p(z_i z_j) \qquad i < j \tag{3}$$

$$r_{ijk} = E_p(z_i z_j z_k) \qquad i < j < k$$

$$r_{12,\ N} = E_p(z_1 z_2, \ldots, z_N)$$

Further define

$$p_1(x_1, x_2, \ldots, x_N) = \prod_{i=1}^{N} m_i^{x_i} (1 - m_i)^{1 - x_i} \tag{4}$$

so that $p_1(x_1, x_2, \ldots, x_N) = p_1(x)$ denotes the probability distribution of the x_i when (1) the x_i's are independently distributed and (2) they have the same marginal distributions as under the distribution $p(x)$. It is shown by Bahadur that for every $x = (x_1, x_2, \ldots, x_N)$ in X

$$p(x) = p_1(x) f(x) \tag{5}$$

where

$$f(x) = 1 + \sum_{i<j} r_{ij} z_i z_j + \sum_{i<j<k} r_{ijk} z_i z_j z_k + \ldots + r_{12\ldots N} z_1 z_2 \ldots z_N \tag{6}$$

The $2^N - N - 1$ correlations and the N marginal frequencies m_i are the parameters which determine the probability distribution $p(x)$. In order that an arbitrary set of $2^N - N - 1$ real numbers $r_{ij}, r_{ijk}, \ldots,$ serve as the correlation parameters of a probability distribution $p(x)$ for any set of numbers m_i, $0 < m_i < 1$, it is necessary and sufficient that $f(x)$ be non-negative for each x.

The distribution $p(x)$ can now be approximated by distributions of lower order. Thus $p_1(x)$ is a first-order approximation to $p(x)$,

$$p_2(x) = p_1(x) \left[1 + \sum_{i<j} r_{ij} z_i z_j \right] \tag{7}$$

is a second-order approximation to $p(x)$, and so on. For $1 \leq m < N$, the approximation $p_m(x)$ is the only distribution of order not exceeding m under which any set $\{x_{j1}, x_{j2}, \ldots, x_{jm}\}$ of m variables has the same joint distribution as under the given $p(x)$. Of course, approximations to $p(x)$ may also be obtained by retaining various selected terms in the expansion for $f(x)$ and dropping the remaining terms. Because any approximation to $p(x)$ is obtained by dropping terms of $f(x)$, a classification procedure based on it will not do as well as the same procedure when $p(x)$ is used.

3. A CLASS OF CLASSIFICATION FUNCTIONS

A well-known theoretical solution to the problem of classifying an unknown pattern into one of two known groups in such a way as to minimize the probability of making an error in the assignment of inputs to the two groups, or in a manner which equalizes the errors for the two groups, minimizes the expected loss, or is best according to some other criterion, is in terms of the likelihood ratio (see, for example, Anderson, 1958, Chap. 6).

In the present case, the problem is to classify an unknown pattern into one of the several groups to which it can possibly belong. One way to proceed would be to set up likelihood ratios for each pair of the K groups (Anderson, 1958) which would require $K(K-1)/2$ classification functions. For the construction of networks, it is desirable to have only a small number of classification functions and by representing the K groups by, for example, a binary code, much less than K classification functions can be considered; each classification function pools patterns from all K groups into just two groups: those which should produce a 1 and those which should produce a 0. Denote these two groups by group 1 and group 2 respectively and let $p(x/i)$, $i = 1, 2, \ldots,$ denote respectively the probability distribution for x under group 1 and group 2. Then the likelihood-ratio regions for classification are defined by

$$R_1 : L(x) = \frac{p(x/1)}{p(x/2)} > t$$
$$R_2 : L(x) \leq t \tag{8}$$

Thus, if the function $L(x)$ exceeds the threshold t, the pattern is classified as belonging to group 1, otherwise the pattern is classified into group 2. [One way of deriving such a rule is to consider the conditional probabilities that, given a pattern having a certain $x = (x_1, x_2, \ldots, x_N)$, the pattern belongs to group 1 or group 2. The boundary of the two regions for classification can be defined by the equation $p(1/x) - p(2/x) = 0$, which gives the expression $p(x/1)/p(x/2) = p(2)/p(1)$.]

The classification functions and corresponding networks which result when various approximations to $p(x)$ are used in a likelihood-ratio procedure can now be derived. If a first-order approximation is used, $p(x)$ is replaced by $p_1(x)$. This implies an assumption of independence of the x_i. Letting m_i and n_i represent the means of the x_i in groups 1 and 2 respectively, the likelihood ratio is

$$L(x) = \frac{\prod_{i=1}^{N} m_i^{x_i} (1 - m_i)^{1 - x_i}}{\prod_{i=1}^{N} n_i^{x_i} (1 - n_i)^{1 - x_i}} \tag{9}$$

taking the logarithm gives

$$\sum_{i=1}^{N} (a_i x_i + c_i)$$

where

$$a_i = \log \frac{m_i}{n_i} \frac{(1 - n_i)}{(1 - m_i)} \tag{10}$$

and

$$c_i = \log \frac{(1 - m_i)}{(1 - n_i)}$$

The summation over c_i can be absorbed in the threshold and a particular weighted sum is obtained for the classification function. The resulting network is that of Fig. 1, with the coefficients as defined in equation (10). Let

$$z_i = \frac{x_i - m_i}{\sqrt{m_i(1 - m_i)}} \qquad y_i = \frac{x_i - n_i}{\sqrt{n_i(1 - n_i)}}$$

and let $r_{ij}, r_{ijk} \ldots$, and $s_{ij}, s_{ijk} \ldots$, be the correlation parameters for group 1 and group 2 respectively. Then for a second-order approximation to the joint distributions, the logarithm of the likelihood ratio is

$$\sum_{i=1}^{N} a_i x_i + \log\left(1 + \sum_{i<j} r_{ij} z_i z_j\right) - \log\left(1 + \sum_{i<j} s_{ij} y_i y_j\right) \tag{11}$$

plus a constant term. Using the approximation $\log(1 + \theta) \approx \theta$, and letting

$$u_{ij} = \frac{r_{ij}}{\sqrt{m_i(1-m_i)\,m_j(1-m_j)}} \quad \text{and} \quad v_{ij} = \frac{s_{ij}}{\sqrt{n_i(1-n_i)\,n_j(1-n_j)}} \tag{12}$$

expression (11) can be written as

$$\sum_{i=1}^{N}\left[a_i + \sum_{j\neq i}(-m_j u_{ij} + n_j v_{ij})\right]x_i + \sum_{i<j}(u_{ij} - v_{ij})\,x_i x_j \tag{13}$$

Similarly, if third-order correlations are retained, the above derivation gives

$$\sum_{i=1}^{N}\left[a_i + \sum_{j\neq i}(-m_j u_{ij} + n_j v_{ij}) + \sum_{\substack{j\neq i\\ k\neq i\\ j<k}}(m_j m_k u_{ijk} - n_j n_k v_{ijk})\right]x_i$$
$$+ \sum_{i<j}\left[(u_{ij} - v_{ij}) + \sum_{\substack{k\neq i\\ k\neq j}}(-m_k u_{ijk} + n_k v_{ijk})\right]x_i x_j + \sum_{i<j<k}(u_{ijk} - v_{ijk})\,x_i x_j x_k \tag{14}$$

The above expressions come from Bahadur (1959a). Here we note that second-, third-, and higher order approximations give rise to expressions linear in the x_i, under certain conditions. In (13), if $u_{ij} = v_{ij}$, the resulting classification function is represented by a network such as that of Fig. 1, in which the coefficients are given by

$$a_i^{(2)} = a_i + \sum_{j\neq i}(-m_j + n_j)\,v_{ij} \tag{15}$$

Similarly, (14) will lead to a classification function, linear in the x_i, if $u_{ijk} = v_{ijk}$, and

$$u_{ij} = v_{ij} + \sum_{\substack{k\neq i\\ k\neq j}}(m_k - n_k)\,v_{ijk}$$

the coefficients of the classification function being

$$a_i^{(3)} = a_i + \sum_{j\neq i}(-m_j u_{ij} + n_j v_{ij}) + \sum_{\substack{j\neq i\\ k\neq i\\ j<k}}(m_j m_k - n_j n_k)\,v_{ijk} \tag{16}$$

In this manner, a set of classification functions

$$\sum_{i=1}^{N} a_i x_i,\ \sum_{i=1}^{N} a_i^{(2)} x_i,\ \ldots,\ \sum_{i=1}^{N} a_i^{(N)} x_i$$

corresponding to increasing orders of approximation to the joint distributions of the x_i in the two groups, are obtained when specific assumptions are made about the relationships between correlation parameters. The network corresponding to each of these classification functions has the form of Fig. 1; a network which attempted to classify correctly eight groups of patterns could be constructed from three classification functions of the type used in Fig. 1. The performance of such networks relative to corresponding networks based on classification functions of the form (13), (14), and higher order approximations will be determined by the extent to which assumptions about relationships between correlation parameters, necessary to obtain the above linear classification functions, are true for a given experimental situation.

In addition to the above derivation, one can consider various intuitive procedures for deriving linear classification functions. An arbitrary linear combination of the measurements $(x_1, x_2, \ldots, x_N)$ is the function $\sum_{i=1}^{N} a_i x_i$, in which the coefficients a_i are to be chosen to provide maximum discrimination (in some sense) between the two groups. An example of such a procedure is that due to Fisher (1936). Let m_i and n_i represent the means of the x_i for group 1 and group 2 respectively, and let $d_i = m_i - n_i$. The difference in mean values of the linear function for the two groups is $\sum_{i=1}^{N} a_i d_i$. Furthermore, the function $\sum_{i=1}^{N} a_i x_i$ has the variance $\sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j v_{ij}$ where v_{ij} are elements of the covariance matrix, assumed equal for the two groups. The sense in which maximum discrimination between the two groups is provided by Fisher's discriminant function is to choose the coefficients a_i such as to maximize the ratio

$$\frac{(\Sigma\, a_i d_i)^2}{\sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j v_{ij}}$$

Introducing a Lagrangian multiplier λ and differentiating the expression

$$\Sigma\Sigma\, a_i a_j d_i d_j - \lambda\, \Sigma\Sigma\, a_i a_j v_{ij}$$

one obtains a set of linear equations which have the solutions

$$a_i = v^{1i} d_1 + \cdots + v^{Ni} d_N \qquad i = 1,2,\ldots,N$$

where v^{ji} are elements of the inverse of the common covariance matrix. This same function results from a likelihood-ratio procedure for the case of continuous variables with multivariate normal distributions and equal covariance matrices for the two groups, and so it is the optimum discriminant function only when these specific conditions hold; without the assumption of equal covariance matrices, a quadratic function would result from the likelihood ratio (Rao, 1952, Chap. 8; Anderson, 1958, Chap. 6). For the latter case, intuitive procedures which lead to linear discriminant functions can be considered. Examples are the Anderson-Bahadur method (1960) which for the case of arbitrary distributions, maximizes the ratio

$$\frac{\sum_{i=1}^{N} a_i d_i}{\left(\sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j v_{ij1}\right)^{1/2} + \left(\sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j v_{ij2}\right)^{1/2}}$$

i.e., the ratio of the difference between means to the sum of the standard deviations and the linear discriminant functions presented by Kullback (1959, Chaps. 9 and 13) obtained by maximizing three measures of information.

4. DETERMINATION OF COEFFICIENTS FROM SAMPLES

From the discussion up to this point it is apparent that, for the most part, the class of pattern-recognition networks considered in section 1 continue to be excursions in the realm of linear discriminant functions. It is also clear that the use of a weighted sum of the x_i as the classification function does not, as some have suggested, limit the x_i to being independent, but may imply a variety of relationships among correlations and covariances of the type present in the examples of linear discriminant functions given in the last section.

The major departure of the pattern-recognition efforts being discussed from the work in linear discriminant functions is the manner in which samples are used. Rather than obtain the coefficients of the classification function from assumptions concerning the functional form of the probability distributions or from a program of estimation, interest has shifted to starting from an arbitrary initial state $(a_1', a_2', \ldots, a_N')$ and using iteration based on experience with one or more samples on each trial, to go from the initial state to a final state $(a_1, a_2, \ldots, a_N)$ which will produce a desired result.

The problem of using experience to go from some arbitrary initial state of coefficients to a final desired state can be approached in many ways; one way is to use completely random perturbation of the coefficients and some of the adaptive systems presented at this symposium report using this method. One would generally desire somewhat more systematic methods which, at least conceptually, have a better chance of producing a sequence of adjustments which converge in some meaningful sense. The problem may then be stated as one of finding a set of transition operators T to apply to the state vector. In this form, varying degrees of complexity can be introduced into the formulation of the problem, as is discussed by Bellman (1961). However, complexity in formulation which introduces complexity in computation is not very helpful; the procedures which are desirable are those which involve simple calculations after each trial and do not require the storage of old data for use in future computations. Useful iteration procedures can be derived from the point of view provided by the techniques used in stochastic models for learning (Bush and Mosteller, 1955; Bush and Estes, 1959; Luce, 1959; Kanal, 1961; Kanal, 1962) and from the point of view provided by stochastic approximation methods (Dvoretsky, 1956; Sakrison, 1961; Kushner, 1960; Magee, 1960). Typical of a number of other efforts is the approach of minimizing a mean-square error criterion (Widrow, 1960; Widrow and Hoff, 1960; Gabor et al., 1961).

5. SOME COMMENTS ON COMPARING DESIGN PROCEDURES

The embodiment of adaptive procedures in a real-time pattern-recognition system is most desirable when the environment in which the system operates can undergo unsuspected changes. However, when the environment is stationary, the design of a pattern-recognition network corresponding to a given form for the classification function can be carried out on a computer. In this case the coefficients can be obtained either by using the data directly in an iterative procedure, or using statistical estimation of parameters to obtain classification functions such as outlined in section 3. Computer programs for obtaining discriminant functions and other classification functions for application in a variety of fields have been used for a number of years. Typical of some recent applications to speech and character recognition is the work reported in Marill and Green (1960), Welch and Wimpress (1961), and Keith-Smith and Klem (1961), from which an idea of the computation involved can be obtained. One point of comparison between the various methods one may consider is, of course, the relative complexity of the computation.

The error curve corresponding to a classification function can be obtained by computing the probabilities of misclassification for different choices of the threshold. An evaluation of the worth of the classification functions resulting when iteration based on experience is used to obtain the coefficients from samples can be provided by comparing their error curves with error curves obtained from the linear functions corresponding to the different orders of approximations as discussed in section 3, and with the error curves obtained from intuitive procedures such as those of Fisher (1936), Anderson and Bahadur (1960), and Kullback (1959).

6. SOME COMMENTS ON THE CHOICE OF OBSERVABLES, AND ON INVARIANCE PROPERTIES

It has been long recognized that a central problem is the choice of a suitable set of observables and, for the most part, an arbitrary choice has been made, as for example, the choice of dichotomous variables obtained from the elements of an "artificial retina."

Experience indicates that when using procedures which are not optimum, a classification procedure based on dividing the x_i into mutually exclusive subsets s_j, deriving classification functions f_j based separately on the s_j, and using f_j to obtain a final classification function F can, in some situations, do better than a similar function based directly on all the x_i. (In most of the work reported on the type of network represented by Fig. 2, subsets of dichotomous variables have been chosen in a random manner and arbitrary fixed weights have been used to form the first set of classification functions f_j.)

A discussion of the problem of selecting a small number of dichotomous variables from an available large set is presented in the paper by Raiffa (1957).

Some comments on the invariance of two measures of information to nonsingular transformations of the observables, and on the connection between the invariant properties and linear discriminant functions are presented in Kullback (1959). The paper by Ming-kuei Hu (1961) presents a set of moment-invariants. It should be noted that for most situations these moments will themselves be random variables.

ACKNOWLEDGMENT

The author is indebted to C. F. Fey for the computer simulation of various pattern-recognition procedures and to D. F. Smith for his contribution to a comparative study of a number of pattern-recognition devices. He wishes to thank J. Owen for helpful comments made at the presentation of this paper. He also wishes to thank K. K. Maitra for many helpful discussions related to the author's work, and N. F. Finkelstein for his encouragement and support.

REFERENCES

Anderson, T. W. 1958. An introduction to multivariate statistical analysis. John Wiley & Sons, New York.

Anderson, T. W. and Bahadur, R. R. 1960. Classification into multivariate normal distributions with unequal covariance matrices. Paper presented at Inst. Math. Stat. Meeting, Wash., D.C., Jan.

Bahadur, R. R. 1959a. On classification based on responses to *n* dichotomous items. USAF SAM series in Statistics, Randolph AFB Texas.

Bahadur, R. R. 1959b. A representation of the joint distribution of responses to *n* dichotomous items. USAF SAM series in Statistics Rept. 59-42, Randolph AFB, Texas.

Bellman, R. 1961. Adaptive control: a guided tour. Princeton Univ. Press, Princeton, N.J.

Bush, R. R. and Estes, W. K. (ed.) 1959. Studies in mathematical learning theory. Stanford Univ. Press.

Bush, R. R and Mosteller, F. 1955. Stochastic models for learning. John Wiley & Sons, New York.

Dvoretsky, A. 1956. On stochastic approximation. In: Third Berkeley symposium on mathematical statistics and probability Univ. Calif. Press.

Fisher, R. A. 1936. The use of multiple measurements in taxonomic problems. In: Contributions to mathematical statistics, John Wiley & Sons, New York.

Gabor, D., Wilby, W., and Woodcock, R. 1961. A universal nonlinear filter, predictor and simulator which optimizes itself by a learning process. Proc. Inst. Elec. Engrs. 108, Part B.

Hawkins, J. K. 1961. Self-organizing systems—a review and commentary. Proc. IRE, Jan., 31-48.

Hu, Ming-kuei. 1961 Pattern recognition by moment invariants. Letter to the Editor, Proc. IRE, Sept.

Kanal, L. 1961. On a random walk related to a nonlinear learning model. IRE Internat. Convention Record, Part 2, Mar.

Kanal, L. 1962. A functional equation analysis of two learning models. Psychometrika, Vol. 27, Mar.

Keith-Smith, J. E. and Klem, L. 1961. Vowel recognition using a multiple discriminant function. Letter to the editor. J. Acoust. Soc. Am., p. 358, Mar.

Kullback, S. 1959. Information theory and statistics. John Wiley & Sons, New York.

Kushner, H. Efficient iterative methods for optimizing the performance of multiparameter noisy systems. M.I.T. Lincoln Lab. Rept. No. 22G-0043, Oct.

Luce, R. D. 1959. Individual choice behaviour. John Wiley & Sons, New York.

Magee, E. J. 1960. An empirical investigation of procedures for locating the maximum peak regression function. M.I.T. Lincoln Lab. Rept. 22G-0046, Oct.

Marill, T and Green, D. M. 1960. Statistical recognition functions and the design of pattern recognizers. IRE PGEC Trans., Dec.

Raiffa, H. 1957. Statistical decision theory approach to item selection for dichotomous test and criterion variables. Rept. 56-139. SAM Series in Statistics. Randolph AFB, Texas.

Rao, C. R. 1952. Advanced statistical methods in biometric research. John Wiley & Sons, New York.

Sakrison, D. 1961. Application of stochastic approximation methods to optimum filter design. IRE Internat. Convention Record, Part 4.

Welch, P. D. and Wimpress, R. S. 1961. Two multivariate statistical computer programs and their application to the vowel recognition problem. J. Acoust. Soc. Am., Apr.

Widrow, B. 1960. Adaptive sampled-data systems. IFAC Moscow Congress Record. Butterworth Publications, London.

Widrow, B. and Hoff, M. E. 1960. Adaptive switching circuits. Tech. Rept. 1553-1. Stanford Electronics Lab., Stanford Univ., Calif., June.

Artificial Neural Networks and Statistical Pattern Recognition
Old and New Connections
I.K. Sethi and A.K. Jain (Editors)
1991 Elsevier Science Publishers B.V

LINKS BETWEEN ARTIFICIAL NEURAL NETWORKS (ANN) AND STATISTICAL PATTERN RECOGNITION

by Paul J. Werbos
Room 1151, National Science Foundation*
Washington, D.C. 20550

1. Overview

This chapter will try to address the three broad questions: (1) In what ways do ANNs differ from the well-known paradigms of SPR? Are there concepts in ANN for which no counterpart in SPR exists (and vice-versa)?; (2) What benefits can come out of interaction between ANN and SPR researchers?; (3) What advantages, if any, do ANN techniques have over SPR methods in dealing with real-world problems?

For the most part, ANNs for pattern recognition are to SPR what poetry is to prose -- technically a subset, based on the acceptance of certain design constraints of enormous practical utility, plus an additional source of inspiration which can be very helpful in the design process. Beyond that, the underlying concepts and paradigms are so close, and the range of problems under study overlap to such a degree, that greater mutual communication can minimize duplication of effort and enrich both communities in many other ways. One area of substantive difference is that some parts of the ANN community treat pattern recognition as a subset of systems to perform dynamic system identification and control; this may lead to improvements in pattern recognition, in some cases, when the pattern to be recognized is essentially dynamic (e.g., speech or moving targets). The attempt to understand intelligence in the brain forces us to keep facing up to difficult and important questions, such as the question of how a host of more specific designs and methods can be integrated together in one unified, flexible general-purpose system.

The remainder of this chapter will elaborate on these points. Section 2 will discuss pattern recognition and neural networks in general terms. Section 3 will discuss a few alternative neural network paradigms, and their relation to SPR. Section 4 will briefly discuss dynamic systems and control. The chapter will start on an introductory level, but will try to build up to current research.

2. Neural Networks and Pattern Recognition -- Generalities

Before discussing the pros and cons of ANNs, I should first define just what an ANN really is. However, neuroengineering -- the field which studies ANNs -- is a large and diverse field, using a wide variety of designs. There are popular articles which define very clearly and explicitly what an ANN is, but which single out only <u>one</u> tool out of a

*The views expressed here are those of the author, not those of NSF. Nevertheless, they were prepared in the course of government work and are therefore in the public domain range of hundreds. Biologists, especially, tend to object to such articles, because "neural networks" are often defined in a way which excludes actual physical neurons Section 3 will give examples of a few tools, but this section will give more of an overview.

There are at least two definitions which have been used by different authors, to try to encompass this field The most common definition emphasizes the prefix "neuro" -- pertaining to the brain. In that definition, neuroengineering is that field which tries to copy over the known capabilities of the brain into computer hardware and software More precisely, it develops mathematical designs which could be embodied directly in hardware or simulated in software; usually it is better to do the testing in software.

In developing these designs, neuroengineering tries to make use of what is known about how the brain achieves these capabilities. There are actually two schools of thought here. One school closely follows only what is now known about the brain (in simplified terms), and looks for "emergent computational properties." This may be called the bottom-up or biologically-based school. The other school treats our knowledge of the brain as a very loose constraint, and focuses on the desired capabilities, making heavy use of statistics and control theory The cooperation between these two schools is not as close as it might be, but much of the vitality of the field is due to what cooperation does exist. This vitality is important both as a source of new ideas, and as a force to attract and motivate a new generation of students.

Before going further, I must admit that I tend to follow the top-down school. Even though my initial motivation was to understand the human mind -- and therefore the brain -- I felt that a top-down approach was essential, in order to complement the vast accumulation of raw data in neurophysiology In fields like economic modeling, it is now well-recognized that bottom-up modeling, when based on the desire to explicitly represent every factory in the country, will usually lead to worse forecasts than a top-down approach which tries to get the aggregate behavior straight before adding more micro-level detail. If the dynamics of each and every factory (and factory manager) were fully and exactly known, this would not be possible, but in the real world it is. In studying the human mind, the key aggregate behavior of interest is the ability to learn how to solve complex, real-world problems, using neural-like computational structures. Once again, the goal is to complement the bottom-up research, not to provide an alternative

Three Subdivisions of Neuroengineering

One way of mapping out the complex field of neuroengineering is by considering which capabilities of the brain different researchers are trying to replicate. Broadly speaking, there are three groups One group is trying to understand specific brain pathways and connections which implement specific abilities, without regard to how these abilities are learned For example, some researchers have spent most of their time doing biological experiments on the cochlea of the ear, in order to develop better models of the actual transfer functions used in adult mammals to preprocess speech data; they then implement the same transfer functions directly in VLSI chips which can be used to preprocess speech data.(Shamma 1987, Lyon and Mean 1988) Similar efforts have been carried out for image processing (Mead 1989), for sonar processing (inspired by the sonar system of the bat, among others), for motor control, and for other areas. In a public talk, Bourlard and Wellekens (1988) have reported that the use of more biological preprocessors may improve performance in speaker-independent speech recognition, even in a situation where it has little effect on speaker-dependent performance.

A second group mimics the ability of the brain to learn the solution to specific problems. Traditionally, in neuroengineering, we build systems to perform one of three kinds of learning: supervised learning; reinforcement learning; and unsupervised learning

Supervised learning means learning a static mapping from a vector X to a vector Y, when there is a training set containing data on both vectors If the vector X contains a pattern, and the vector Y contains a classification of that pattern, for each item in the training set, then supervised learning boils down to a task which people in SPR have worked on for years Often, however, ANNs are designed to adjust their parameters one observation at a time, in real-time learning, rather than iterating over an entire database.

Reinforcement learning is similar to supervised learning, except that we do not assume the availability of target vectors, Y, in the training set. Instead, we assume the availability of a performance judge or utility function, which reports how good the current outputs of the network are In essence, supervised learning is like telling a child where to go (in the two-dimensional Y space of a back yard), while reinforcement learning is like telling him that he is "colder" or "hotter" as he gets closer or further from a goal state. Many biologists believe that reinforcement learning is a better description of how animals learn than supervised learning, because animals in nature receive rewards and punishments from the environment but are not told what to do in detail.

Unsupervised learning is often touted as a way of adapting neural networks without giving them any kind of directive feedback at all Some researchers are strongly attracted to this way of describing things, because it suggests the possibility that some neural networks (like themselves?) may truly experience absolute free will, as described by philosophers of the French existentialist school From an engineering perspective, however, it is more useful to focus on the specific tasks which networks in this broad category are actually trained to perform There are at least three main categories of network here: (1) networks designed as "associative memories" (or, more precisely, "autoassociative memories") -- networks which, when given part of a pattern they have seen before, will reconstruct the entire pattern; (2) networks designed as feature extractors, such as "competitive learning" systems (based on clustering) or as "self-organizing maps" (with analogies to factor analysis and principle components analysis); (3) networks designed to model the dynamics of the variables they observe (system identification networks). All three have important applications, and there have been many efforts to combine these three (and supervised learning and reinforcement learning and fixed preprocessors) in various combinations, for different applications.

Finally, there is a third group of neural network researchers, besides the fixed-task group and the learning group Years ago, John Hopfield (Hopfield and Tank 1986) generated great excitement with the observation that certain ANNs, originally derived in the biologically-based literature (Grossberg 1988), could also be used to minimize very complex quadratics or solve even more complicated static optimization problems They also translate relatively easily into hardware implementations, like new VLSI chips or optical computers

One of the classic applications of such Hopfield nets is in computer vision. Some approaches to computer vision would try to identify edges and segments in an image by minimizing a very complex measure of the quality of fit of the segmentation scheme;

many authors have used Hopfield nets to perform this minimization, and some propose to hardwire the networks onto a chip. Hopfield nets are also used in applications like traveling-salesman problems and problems of placing connections in various kinds of (nonneural) networks

From a technical point of view, Hopfield nets are continuous-time gradient-based methods to minimize a user-specified "energy" function. They are often applied to energy functions which do not have a unique local optimum; their energy surfaces often define a pock-marked surface, with many local minima, and it is a serious challenge to find the desired global minimum Two approaches are commonly used to cope with this problem. In one approach, one tries to find an energy function which satisfies the application without having local minima. The other approach, uses stochastic search methods to try to find the global optimum despite the existence of local optima; popular methods along these lines are simulated annealing (e.g, Sun and Hassoun, 1990) and genetic algorithms (Ackley 1990).

An Alternative (Hardware-Based) View of Neuroengineering

Many programs in neuroengineering, both in government and in industry, were not motivated at first by any interest in understanding the brain. For example, a large part of this activity grew out of hard core research in physics, motivated by a desire to exploit the unique computational potential of photons compared with electrons. Leading experts in optics such as Caulfield, Psaltis and Farhat argued that optical computing had the potential to achieve a million-fold improvement in throughput compared with the best digital VLSI technology Given the huge size of the computer industry, and the economic importance of any improvement in throughput, this claim generated very serious further evaluation Carver Mead (1989), who is regarded as the father of VLSI technology, replied that analog, parallel, fixed-function VLSI might also achieve a million-fold increase in throughput, compared with the best general-purpose digital technology. (After all, even the best digital technology typically runs one process stream on one chip, a chip with a million transistors or so; however, an analog multiplication requires only a handful of transistors.)

The critics of optical computing and analog VLSI argued that both of these technologies are far too restrictive to be useful in anything like general-purpose computing. They would fit a very tiny fraction at best of the overall computer market. After all, the vast majority of existing computer programs, in all fields of science and industry, are loaded down with "IF" statement and "DO" loops and long sequences of multiple types of instructions; it would be almost impossible to break them up into parallel calculations performing the same operations (or a tiny choice of operations) over and over again.

The advocates replied that the human brain uses fixed-operation analog distributed hardware, and that it seems capable of handling a fairly wide range of computational tasks. Many research managers agreed with this assessment, and concluded that the challenge before us was to duplicate this kind of capability in computers In brief: the initial goal of many research programs in this field was simply to develop more broad general-purpose algorithms which could take advantage of the capabilities of this kind of computer hardware. The goal was to develop the most efficient possible algorithms -- drawing on everything we know from existing disciplines like SPR, control theory and biology -- subject to the constraint that the algorithms must be inherently implementable in this kind of hardware. From an abstract mathematical point of view, one can always find a "better" (or equally good) algorithm if one throws out the constraint, and searches the entire space of all algorithms which will run on a sequential computer; however, from a practical point of view, algorithms which live within the constraint have a theoretical potential to run at one millionth the cost of those which do not.

This theoretical potential has rapidly become a practical reality, just in the past year (1989/90). This past year, Intel -- among others -- has announced the availability of a new neural network chip, which exploits the potential of analog VLSI David Andes of the U.S. Navy (China Lake) has stated that one handful of the Intel chips has more computational power, for what it does, than all of the Crays in the world put together (The competing chips are also well worth looking at.) Factory managers who use computer vision products often complain that programs exist which could solve many of their production automation problems, but that the existing products would require a Cray at every station (which they cannot afford); if you can mimic or duplicate your existing algorithm with an ANN, you can give such managers the option of buying a small box with a few chips in it instead Because the same functional form can be used to perform a wide variety of tasks, one can achieve economies of scale which would not be possible with more narrow special-purpose chips; one can also reprogram such neural hardware -- simply by sending in new parameter values -- even after it is installed. Similar considerations apply to most military or aerospace markets (like China Lake) In brief, the shift to ANNs could open up enormous new markets for pattern recognition, because of the economics of new hardware.

Is there a limit to what kinds of algorithm can live within this constraint? Last year, Stinchcombe and White (1989) proved that simple feedforward networks -- including the kind of network implemented on the Intel chip -- can work as a "universal approximator" of and decently behaved nonlinear mapping In more recent, unpublished work, they have gone on to prove the ability to approximate functions in a way which makes the derivatives of the approximation match the derivatives of the original function, to any desired degree of accuracy, to any order of differentiation

3. Some Examples of ANN Paradigms

Background

Are there concepts in ANN for which no counterpart in SPR exists (and vice-versa)? This is an extremely difficult question, because it requires knowledge of all the "holes" in both of the two disciplines, as well as a definition of the boundary of both disciplines (which is an exercise in semantics) Furthermore, good ideas in either discipline can be assimilated into the other; thus the holes which exist now can still be filled in the future.

I would guess that about 80% of the work now being done with ANNs could be classified as pattern recognition. In addition to research on image processing and speech recognition, there has been substantial industrial interest in applications like diagnostics, sensor fusion and financial risk assessment. In fact, the latter three probably account for the bulk of the substantial real-world applications to date. Applications to target recognition, sensor fusion and the like account for a majority of current DOD funding of the field, but NSF is currently trying to put more emphasis on other areas which receive less support from DOD

Maren (1990) is making some effort to classify and dissect the full range of paradigms used here, but the sheer volume of research makes it very difficult to do this now. Therefore, in this chapter, I will only give a few examples of important paradigms which I personally happen to know about. For those interested in a broader view, the Proceedings of the International Joint Conference on Neural Networks serve as something like an encyclopedia (with index) of this field.

Overview

Most of the applications of ANNs to pattern recognition still involve the use of basic backpropagation, applied to classical feedforward networks. Therefore, I will begin this chapter with a review of that method, and discuss a practical application from AT&T (Guyon et al, 1989) and its connection to Lie Groups in SPR. Next, I will discuss alternative methods, based on different functional forms Then I will discuss the statistical basis of basic backpropagation, and introduce some alternative methods (and research topics) suggested by the theory.

All of the examples here will involve supervised learning, the learning of a mapping from a vector $\underline{X}(t)$ to a vector $\underline{Y}(t)$ Section 4 will deal briefly with control (which includes reinforcement learning) and neural nets for system identification (a form of unsupervised learning) In the last year or two, many researchers have also begun to use feature-extracting neural networks as preprocessors when classifying patterns; for example, Granger Sutton et al (1990) use a competitive learning (clustering) preprocessor, Foldiak (1989) uses a design which essentially performs principal components analysis, and Hrycej (1990) uses a design more like true factor analysis, related to the work in Werbos (1990d) Grossberg (1987) is probably the most popular overview of competitive learning methods Kohonen (1988) discusses several different feature extraction approaches. Kosko (1990) describes some more recent work related to competitive learning. This chapter will not review any of that work.

Basic Backpropagation: Fundamentals

Guyon et al (1989) provide an excellent example of basic backpropagation, applied to a real-world classification problem

Guyon et al began with a database of 1200 handwritten digits. (The goal was to classify individual ZIP code digits after a preprocessor at the post office had performed the initial segmentation.) In my notation, I would say that the training set consists of $T=1200$ observations For each observation, t, the observation consists of two pieces of information -- the input vectoı, $\underline{X}(t)$, and the target vector, $\underline{Y}(t)$. In this case, $\underline{X}(t)$ was a vector consisting of 256 components, $X_1(t)$ through $X_{256}(t)$, each referring to the grey-scale level of a pixel in the 16-by-16 input image. Likewise, $\underline{Y}(t)$ consisted of 10 components, $Y_1(t)$ through $Y_{10}(t)$, corresponding to the 10 possible classifications. In my notation, I would say that the size of the input vector is $m=256$, and the size of the target vector is $n=10$. Our goal is to initialize and then train a neural network to input $\underline{X}$, and output a prediction of $\underline{Y}$

The most popular way to set up such a network is to use a three-layer structure --

with an input layer, a hidden layer, and an output layer. The outputs of the hidden layer are calculated first, all in parallel, followed by the outputs of the output layer.
The outputs of the hidden layer are calculated by:

$$x_i(t) = s(net_i^-(t)) \qquad i=1,\ ,h \qquad (1)$$

$$net_i^-(t) = W_{i0}^- + \sum_{j=1}^{m} W_{ij}^- X_j(t) , \qquad i=1,\ ,h \qquad (2)$$

where W⁻ is a matrix of weights or parameters, where h is the number of hidden units, and where s is the "sigmoid" function:

$$s(z) = 1 / (1 + e^{-z}) \qquad (3)$$

The outputs of the hidden layer are:

$$\hat{Y}_i(t) = s(net_i^+(t)) \qquad i=1,\ ,n \qquad (4)$$

$$net_i^+(t) = W_{i0}^+ + \sum_{j=1}^{h} W_{ij}^+ x_j(t) \qquad i=1,\ ,n \qquad (5)$$

The notation here is designed to be simple for this structure, where a "-" denotes the lower layer and a "+" denotes the upper or output layer. With more complex ANNs, it is more usual to define <u>net</u>, <u>x</u>, and W_{ij} as more complicated, sparse vectors and matrices. Intuitively, the variable "net_i" is thought of as the voltage input which excites neuron number i; x_i or $\hat{Y}_i$ is thought of as the output frequency or activation of a neuron; W_{ij} is thought of as the strength of the synapse connecting neuron j to neuron i. If the weights become very large, structures like this become more and more like hyperplane classifiers, the classifiers used in discriminant analysis; again, however, the hidden layers provide optimal features for use in that classification, and large weights are not always best.

In basic backpropagation, the goal is simply to adapt the weights W so as to minimize the square error:

$$E = \sum_{t=1}^{T} \sum_{i=1}^{n} \tfrac{1}{2}(\hat{Y}_i(t) - Y_i(t))^2 \qquad (6)$$

From the viewpoint of SPR, what is new here? Certainly not the idea of minimizing square error! From an abstract, mathematical point of view, this is simply a special case of nonlinear regression, which has been in existence for decades. It is an interesting special case, because the functional form here is easy to implement in hardware, and can serve as a universal approximator (even with a little simplification of the function s) What is really new, however, is the <u>way in which</u> error is minimized, which I will get to.

To adapt this kind of network, there are two common approaches, which may be called batch learning and pattern learning (Pattern learning is usually described as "online" learning, which is misleading, because it is usually done off-line in practice. One may also call it observation-by-observation learning.) In <u>both</u> methods, one starts out with initial guesses for the weights W_{ij}, exactly as one does in nonlinear regression. (Most neural network researchers use a random number generator here, but it is better to use prior information, if available and cheap.) One adapts the weights by a conventional sort of iterative update procedure. In batch learning, one begins each iteration by calculating the gradient of E based on the current weights, and then one uses a gradient-based procedure to update the weights. In pattern learning, one cycles through the observations, t; one begins each iteration by calculating the gradient of:

$$E(t) \quad = \quad \sum_{i=1}^{n} \tfrac{1}{2}(\hat{Y}_i(t) - Y_i(t))^2 \quad , \qquad (7)$$

and adapting the weights <u>immediately</u> in response to this gradient, before going on to analyze the next observation.

The use of pattern learning is somewhat novel; however, there are precedents to this as well (albeit less well-known) What is really novel is the method of <u>calculating</u> the gradient of E(t) or E, with respect to all of the weights, in a single pass through the system. Once again, computational <u>efficiency</u> and <u>economics</u> are the defining features of what we are doing

In basic backpropagation, we use the following sequences of equations to calculate the gradients of E(t) with respect to <u>all</u> of the weights in a single pass through the system:

$$F\_\hat{Y}_i(t) \quad = \quad \hat{Y}_i(t) \; - \; Y_i(t) \qquad i=1, \; ,n \qquad (8)$$

$$F\_net_i^+(t) \quad = \quad s'(net_i^+(t)) \, * \, F\_\hat{Y}_i(t) \qquad i=1, \; ,n \qquad (9)$$

$$F\_x_j(t) \quad = \quad \sum_{i=1}^{n} W_{ij}^+ \, * \, F\_net_i^+(t) \qquad j=1, \; ,h \qquad (10)$$

$$F\_W_{ij}^+(t) \quad = \quad x_j(t) \, * \, F\_net_i^+(t) \qquad i=1, \; ,n \quad j=0, \; ,h \qquad (11)$$

$$F\_net_i^-(t) \quad = \quad s'(net_i^-(t)) \, * \, F\_x_i(t) \qquad i=1, \; ,h \qquad (12)$$

$$F\_W_{ij}^-(t) \quad = \quad X_j(t) \, * \, F\_net_i^-(t) \qquad i=1, \; ,h \quad j=0, \; ,m \qquad (13)$$

(By convention, we assume $x_0 = X_0 = 1$.) The arrays $F\_W_{ij}^+$ and $F\_W_{ij}^-$ contain the desired gradients. Note that we have to start from equation (8) and work through the other equations in order to perform these calculations; we are calculating a kind of error feedback in a direction <u>backwards</u> from the original calculations in equations (1) through (7). ("F_" stands for "feedback to.") For pattern learning, we adjust the weights immediately in response to these gradients. For batch learning, we simply <u>add</u> these gradients across all observations t, and <u>then</u> respond. Even though equations (8) through (13) must be calculated in that order, more or less, the calculations associated with any one equation can all be done in parallel, as was true with the original system.

Intuitively, one may think of these F_ . quantities as the derivatives of error with respect to various intermediate calculations. For very simple neural networks (like the case above), one may rationalize the feedback equations by appeal to the usual chain rule for differentiation. However, with complex networks, it becomes very tricky to do this in a safe and rigorous way.

Werbos (1974) -- which is now usually cited as the first paper on backpropagation -- describes a more rigorous way of understanding these calculations, the core of which has been reprinted in Werbos (1989) These derivatives are understood most easily as "ordered derivatives," a species of partial derivative defined with respect to the order of calculations used in a system of differentiable equations (like equations (1) through (7)). Werbos (1989) shows how a new chain rule for ordered derivatives can be used quite easily, on any ordered system of differentiable equations, so as to yield the required gradient in a single sweep with a rigorous assurance of a correct result. Backpropagation in this more general sense is not restricted to ANNs, and it can be used to calculate the derivatives of other things besides error (see Section 4) Indeed, the concept of ordered derivative provides a coherent and unified way to understand a wide variety of related but specialized concepts in fields from economics through to nuclear engineering; it is the kind of concept which belongs in basic calculus textbooks

After calculating a gradient, we still have to decide how to adapt the weights in response to the gradient. Most workers simply use steepest descent, with a fixed and arbitrary learning rate used across all the weights. This takes many iterations to converge, but less time that it might take to hand-craft a set of optimal features for the hidden layer. Werbos (1988a,1989) discuss a variety of ways to speed up convergence, and Shanno (1990) has described new numerical methods which may be adapted to do even better. (There is a huge literature on this subject, but considerably more fundamental work to be done)

The application of backpropagation to neural networks as such was first put forwards as an idea in Werbos (1981) and Parker (1982), though it was mentioned briefly in Werbos (1977), and the crude intuition which led to this idea -- and to certain adaptive critic control designs -- appeared in Werbos (1968). Werbos (1981) -- a condensed version of a longer paper written under the review authority of Charles Smith -- also mentioned applications to energy modeling, such as sensitivity analysis, robust time-series identification, and the like; such possibilities are discussed as well in Werbos (1990e) Rumelhart, Hinton and Williams (1986) simplified and popularized these ideas, in a seminal article which had a dramatic effect in encouraging interest in this field (and which did acknowledge the earlier role of Parker(1982), LeCun (1985) and Charles Smith). This experience helps to underline the tremendous value of artificial neural networks as way of communicating and explaining generalized concepts in mathematics which otherwise might have remained obscure and difficult

In the mid-1970's, there were many scientists who argued that cheap derivatives were of little real importance, since they do not change the result which is ultimately computed, and since computer time was getting cheaper and cheaper. Now, in the 1990's, we recognize that our ability to handle larger problems becomes ever more sensitive to issues of computational cost and complexity; cost ratios on the order of N (the number of variables) become ever more important as advanced hardware allows us to increase the value of N.

Basic Backpropagation: More on the Example (and Lie Groups)

Guyon et al (1989) -- like most researchers with long experience in ANNs -- did not rigidly follow the pattern of connections shown in equations (1) through (5). The transfer function, s(z), was scaled, to make its inputs and outputs vary between -1 and +1, instead of 0 and 1 The target vector, $\underline{Y}$, was recoded to represent comparisons between alternative classifications (i.e., each Y_i represents the notion that some class A fits the pattern better than some class B).

Before adapting an ANN, they did their best to develop a classical pattern-recognition scheme (their Networks 1 through 4), represented as neural networks with fixed weights. They used this classical analysis to give them a starting structure. They moved on, in the next stage, to a multilayer network, with the lower layers fixed, and the upper layer adaptive. They did try to use a "fully connected" structure in which (as in equations (1) through (5)) each layer received input from every other neuron, but they found that this led to poor results. In general, when people adapt ANNs containing a large number of weights, relative to the training set, the networks behave poorly in new cases outside of the training set (This is called "poor generalization.") Problems with local minima are also more likely to occur. Recognizing this, they used some very interesting methods to try to reduce the number of weights.

Drawing on their knowledge of image processing, they used hidden neurons limited to taking input from certain windows in the input grid For example, one might write:

$$x_{ij} = s(net_{ij}) \qquad (14)$$

$$net_{ij} = \sum_{k=i-2}^{i+2} \sum_{l=j-2}^{j+2} W_{ijkl} X_{kl} \; , \qquad (15)$$

where each neuron is now indexed by two integers, representing coordinates in a two-dimensional grid of pixels Hidden units -- x_{ij} -- were not calculated for every value of i and j; for example, one might have hidden units defined only for even values of i and j, so that the next layer can be more parsimonious. To reduce the number of weights even further, one can take the drastic step of replacing equation (15) with:

$$net_{ij} = \sum_{k=i-2}^{i+2} \sum_{l=j-2}^{j+2} W_{i-k,j-l} X_{kl} \qquad (16)$$

In this case, there are only 25 weights in all for this entire layer of neurons. If this is too drastic, one can define two sets of hidden neurons in parallel, x_{ij}^A and x_{ij}^B, each of which obey equation (16), only with different weights

Guyon et al (1989) did not describe all the details of how they applied these methods, which led to much fewer weights and better generalization. However, in public talks, they have presented such details. They have noted that it took about 20-40 passes through the training set (using pattern learning) to converge. Similar techniques have been extended by AT&T for pattern recognition; however, at the time of those earlier talks, the details were proprietary because of the likelihood of a commercial product.

The chain rule for ordered derivatives can be applied easily to these kinds of structures, just as it can to equations (1) through (5). Here, the gradient of error with respect to $W_{i-k,j-l}$ is essentially the <u>sum</u> of the derivatives of error with respect to the corresponding W_{ijkl}.

These techniques may be seen, more fully, as a way of implementing the notion of translational invariance, as described by Kanal in another chapter of this book. This is simply a special case of the Lie-Group invariance concept. Rotational invariance could easily be imposed on equation (16) to go even further with this concept (i.e., to <u>enforce</u> weights which obey the symmetry restriction, by cutting out excess degrees of freedom). Giles and Maxwell (1987) have applied similar concepts to single-layer, polynomial-based ANNs. Rumelhart (1990) pointed out that this same kind of architecture could be used to recognize letters in arbitrary positions in a visual field (because of the property of translational invariance), thereby avoiding the usual segmentation problem. In general, this example is one more case where a neural network formulation can be communicated more easily than the more general -- but more difficult -- mathematics which underly it.

It would be nice if similar symmetry properties could be applied uniformly to all kinds of patterns, and if higher neural networks in the brain had the ability to detect such symmetries However, after a few years of looking at this, I could find no really plausible way that the brain might be doing this, except indirectly, through the use of explicit symbolic reasoning and building up ANNs which serve as dynamic models of the external environment It is plausible, after all, that the human use of symbolic reasoning may help to explain why humans are better at generalization, on some level, that are other mammals, despite the similarity in brain structure. The retina -- a special-purpose system -- may be different, because it could enforce symmetries which are unique to vision. Still, Hebb (1949) stressed the point that even humans lose their ability to recognize patterns, when <u>movements</u> of the eye are inhibited; dynamic modeling may be crucial even to <u>our</u> abilities to recognize visual images In this connection, it is interesting to remember how vision -- more than the other senses -- played a crucial role in the evolution of the cerebral cortex of mammals and the precursors to it in birds.

<u>Alternative Functional Forms and Pruning</u>

Basic backpropagation, as described above, is simply a matter of minimizing square error with a particular functional form As with conventional linear regression, many users simply dump the input vectors (independent variables) and target vectors (dependent variables) into a computer package, and let the computer derive the relationship. <u>Unlike</u> linear regression, however, this procedure is capable of learning any arbitrary nonlinear relationship, if given enough hidden units.

Despite the practical advantages of this approach, it has at least three theoretical limitations:

- o It uses least squares in a situation where a different error function may be better.

- o It does not allow for the possibility of developing more <u>parsimonious</u> networks which, as in the previous example, can be expected to perform better in generalization.

o It does not allow for prior information, which may point towards <u>different</u> functional forms.

Regarding the first point, the idea of minimizing square error is usually justified by appeal to maximum likelihood theory, on the assumption that errors are governed by a normal distribution. However, the normal distribution generates numbers in the whole range between minus infinity and plus infinity; it is not logically consistent with Y and $\hat{Y}$ restricted to the range between 0 and 1 or -1 and +1. <u>If</u> the inputs and outputs are logically binary, and the outputs can be interpreted as probabilities (e.g. of class membership), one can simply use the classical Bernoulli measure of error (Wonnacott and Wonnacott 1977) instead of equation (6):

$$E(t) = \log(\hat{Y}(t)Y(t) + (1-\hat{Y}(t))(1-Y(t))) \qquad (17)$$

Differentiating, this replaces equation (8) by:

$$F\_\hat{Y}(t) = \frac{2*Y(t) - 1}{\hat{Y}(t)Y(t) + (1-\hat{Y}(t))(1-Y(t))} \qquad (18)$$

but keeps the rest of the scheme intact. Hinton (1990) has reported good results with this approach. When the output is not intrinsically binary, some authors prefer to assume a normal distribution in "net"; in other words, they define:

$$E(t) = \tfrac{1}{2} \sum_i (net_i - net_i^*)^2 \ , \qquad (19)$$

where net* is a target value calculated back from the target output. Many such alternative error functions are possible. The use of such alternative error functions changes equation 8 (or even 9), but does <u>not</u> affect the basic feedforward structure of the network <u>after</u> training; thus it does not interfere with the use of chips like Intel's for pattern recognition <u>after</u> training has taken place On-chip learning is a complicated matter at the present time, with or without these changes

From a statistician's point of view, these different error functions simply represent different models of the random disturbance; the only basis for preferring one over another is prior knowledge about the particular application in hand, or empirical data showing that one fits better than another. Werbos (1990d) addresses the additional problem of how to account for correlations in the disturbances across multiple target variables; this is difficult at present within the cost constraints of ANNs, but may be unnecessary for most current applications.

Parsimony is a more complex and difficult subject. Even though simple feedforward nets may be able to approximate any well-behaved function, there is certainly no guarantee that they will do so in the most parsimonious fashion possible. As in the last section, it is desirable to achieve more parsimony when this is possible; therefore, in most applications, a network will generalize better if you choose the particular functional forms and structures which are most likely to perform well with the minimum number of weights. The cost of doing this may outweigh the benefits, when training data is plentiful and prior information is expensive, but this is not always the case.

Some naive researchers have written papers suggesting that their new functional form (or model of the neuron) is "better" than another, in a truly universal way; however, as in any statistical modeling, different functional forms fit better on different problems, and apriori generalizations across all problems tend to be dangerous. It is true, however, that the standard feedforward networks tend to be better at classification problems where different patterns can be separated easily by hyperplanes; when it is better to use hyperspheres or ellipses, there can be significant benefits in using a slightly different functional form. Yau and Manry (1990) and DeClaris (1990) have reported very good results using backpropagation with functional forms altered in this way, with appeal to SPR. There is also a literature on "radial basis functions," which do the same sort of thing, but usually rely on a hand-crafted lower layer with fixed parameters. (Moody and Darken 1989). In SPR, it is common to use Bayesian methods with the assumption that patterns in different classes are governed by normal distributions; work on "supervised competitive learning" addresses a similar situation (Sutton et al 1990, Kohonen 1988). In general, all of this work illustrates the general principle that it helps to use prior information whenever such information is available and not too expensive. Unfortunately, this will complicate the effort to use standardized chips, until the various tradeoffs and domains of application are better mapped out.

Some researchers suggest that we may look to the brain, to arrive at alternative neuron models which may work better in practice One group proposes to use neurons which calculate crossproducts and even higher-order polynomials in the input variables; certain neurons in the brain do indeed have "modulatory" (multiplicative) inputs, but these mainly involve external timing or synchronization inputs rather than a true calculation of a local quadratic(Foote and Morrison 1987). They also lead to problems in deciding which crossproducts to use and in hardware implementation. Another group (Kuperstein 1987) suggests that true zero-to-infinity inputs should be coded in a kind of logarithmic/decibel scale, which is probably an excellent way to preprocess many forms of data (It is already common in SPR to represent an intensity I as log(I), but the idea here is to code an intensity input into <u>several</u> neurons, which respond logarithmically in their major range but saturate outside that range; different neurons would have different but slightly overlapping regions) A third group suggests the use of neural networks with <u>simultaneous-time recurrence</u> This would allow a network to implement the idea of a relaxation algorithm easily and naturally, without time-lags It is possible to implement backpropagation very efficiently for such a network (Werbos 1988b), but the mathematics are significantly more complex than they are for simple ordered systems. It is unknown as yet how useful this feature is in practical applications, and it is also unknown exactly what the brain is doing along these lines.

One final way to achieve parsimony is simply to prune out unnecessary weights in an ordinary feedforward network. After all, this worked very well for Guyon et al (1989). The brain itself is clearly a highly sparse structure Werbos (1987) presented a few preliminary thoughts on the subject of automatic pruning (and regrowth), but a large number of researchers -- including DeClaris, McAvoy and DeFiguerido (1990) -- claim success in real applications, based on approaches which should be published in the next year or two. From a statistician's point of view, this is clearly just a nonlinear extension of the well-known field of stepwise regression, which statisticians like Dempster (1977) discussed long ago. Sparse networks can be run on chips simply by setting a lot of weights to zero; however, there may be more efficient ways to implement this idea in future generations of chips or optical computers.

Many advocates of neural networks look to ANNs to provide a high degree of fault tolerance. Unfortunately, a high degree of parsimony tends to reduce fault tolerance, ceteris paribus. It may well be that certain neural network structures and training regimes (e.g., adding noise to hidden nodes which are otherwise constrained to be similar, to simulate faults <u>during training</u>) may help reduce the degree of conflict between these objectives, but this is basically a topic for future research.

<u>Statistical Issues and Bedrock and Associative Memory</u>

The preceding parts of this section -- like a majority of the work in neuroengineering and SPR -- is rooted in the classical concepts of maximum likelihood theory and Bayesian estimation theory (Wonnacott and Wonnacott 1977). In those theories, one begins by assuming that the classification model (or neural network model) is "true" for some values of the parameters (or weights) One searches for those values of the weights which have the maximum probability of being the true weights. More precisely, one searches for the values of the weights with the highest conditional probability of being true, conditioned upon the training data One exploits Bayes' Law to deduce:

$$Pr(weights \mid data) = \frac{Pr(data \mid weights)Pr(weights)}{Pr(data)} \qquad (20)$$

The first term on the right -- Pr(data|weights), the likelihood term -- is a straightforward computable function of the model, the weights and the data, so long as the model does include a model of the (probability distribution of the) error disturbances When errors follow a normal distribution, independent of each other, then maximizing the likelihood term is essentially equivalent to minimizing square error. The term in the denominator -- Pr(data) -- does not affect the <u>relative</u> choice between different sets of weights, and may be ignored here. The other term -- Pr(weights) -- is the prior probabilities term, which has been an embarrassment to students of learning and statistics ever since the time of Immanuel Kant or earlier; it represents our knowledge about likely values of the weights <u>before</u> we have access to any empirical data.

In maximum likelihood theory, we usually assume that all possible sets of weights are equally likely apriori Thus to maximize equation (20), we simply maximize the likelihood term Some engineers have argued that this procedure is firmly grounded in theory and scientific, because it leads to precise and predictable results <u>without</u> relying on subjective things like prior probabilities, unlike the more "ad hoc" methods used by some people in SPR and neuroengineering In certain fields, like economics, the maximum likelihood approach is indeed more practical in most cases than explicit Bayesian regression (where the computer asks the user to provide Pr(weights)), because it is easier for a human to trade off empirical data versus prior information in his head when he knows which is which

Unfortunately, the assumption of equal probabilities apriori is still an arbitrary assumption, with no more guarantee of truth than any other arbitrary assumption one might make. The real world is far messier. Least squares can be very useful, when the training set is larger than the number of weights, as discussed above; however, as the training set grows smaller, or the number of variables grows larger, it becomes ever more important to make our prior assumptions as realistic as possible. No one is immune to this problem -- engineers, statisticians, neuroengineers or SPR practitioners. Neuro-engineers and SPR practitioners have faced up to this problem more than most groups have, because we both deal so often with enormous input vectors. Likewise, for very similar reasons, we need to face up to the issue which statisticians call "robustness" -- the problem of how to work with models or networks which are unavoidably oversimplified representations of external reality Actually, these problems are central to econometrics as well, but econometricians have many ad hoc fixes available (Werbos (1990c).

For larger training sets, these issues of prior probabilities and robustness help to explain the need for parsimony. Long ago, philosophers like the Reverend Occam argued that humans would be unable to learn from experience without somehow giving greater credence to a simpler model, rather than a complex model, in cases where both fit experience equally well Solomonoff (1964) formalized this notion, and showed how it is still consistent with notions such as "open-mindedness" which we would want our learning systems to possess. Later workers in complexity theory formulated notions of estimation which are closely related or even equivalent to Solomonoff's proposal, and are working to refine these concepts in the context of neural networks. Sara Solla of AT&T and Papantoni-Kazakos are working along these lines, but the subject is quite difficult. (Tishby, Levin and Solla 1989, Papantoni-Kazakos 1989).

In the limit as the training set becomes extremely small, the best way to predict the classification of a new pattern is usually by analogy (If the input vector is really just one number, one usually relies on classical interpolation, which is similar.) This helps to explain the widespread popularity of nearest-neighbor schemes in SPR, and the popularity of associative memory approaches to supervised learning in ANNs. (Kosko has called these systems "heteroassociative memory", and stresses that they converge faster than backpropagation.) The same principle is at work in both fields.

Classical statisticians have suggested that such situations can still be understood within the scope of maximum likelihood theory One can formulate a model which predicts new patterns to be similar to their neighbors, plus noise, and then use maximum likelihood methods One could even use such methods -- with backpropagation -- to adapt a similarity metric, which may even be modeled as a function of system inputs rather than a fixed matrix. To make this approach more consistent with conventional modeling, we could simply use this arrangement to forecast the errors of a more conventional least-squares forecasting network Werbos (1977) described this general approach, under the name of "syncretism," but no one has had a chance to pursue it as yet. A good adaptation of the similarity metric would still require a large database, but the approach may be of value in cases where similarity-based forecasting works better than ordinary approaches Apriori probabilities would still affect the results, because of the importance of the initial (or default) value of the similarity metric.

There is an interesting similarity between syncretism, on the one hand, and variations of basic backpropagation in which different weights use different learning rates to respond to gradients Both the weight-specific learning rates and the corresponding similarity metric terms serve as measures of "attention" -- the degree to which a specific input is used to explain any errors in classification Most biologists believe that time-varying attention in some form is essential to learning in higher organisms.

Syncretism is simply one of many ways to try to come up with a more rational, flexible and technically sound way to reconcile the capabilities of nearest-neighbor classification and least-squares classification. There are many other ways, and there is much room for further research, both in the SPR and ANN communities. Werbos (1987) described another such approach, which generalizes the classical statistical method of ridge regression (Dempster 1977), in which the norm of the weight vector is added to the error function E(t).

Among the host of associative memory and vector quantization methods actually used in engineering applications, the ones encountered most often are related to the CMAC architecture (Albus 1971) and the ideas of Kohonen (1988). Lukes et al (1990) used a simple associative memory for a simulated control application, and Miller (Kraft and Campagna 1990) used a CMAC to control a physical robot. Both researchers have reported informally that they have successfully implemented differentiable versions of these designs, which yield more accurate results and open up new applications. (For example, one can build a two-layer structure in which the upper, fast layer is an associative memory, and the bottom layer is adapted by backpropagation. This would fit the idea that humans learn new patterns quickly and new features more slowly. There are important control applications as well.) Nestor, Inc, of Providence, Rhode Island, has also found many clients for its memory-based supervised learning system. Fahlman (1990) has developed a new learning system which is intermediate in spirit between backpropagation and associative memory.

4. Dynamic Systems and Control

So far we have discussed pattern recognition as a static problem, mapping a vector $\underline{X}(t)$ to a vector $\underline{Y}(t)$. In actuality, there are many classification problems -- like speech or target recognition in a moving viewscreen -- which are essentially dynamic in nature. Furthermore, pattern recognition in mammals occurs in subsystems, within the context of a larger system (brain) whose overall function is one of control. For this reason, the most important area for fundamental research in neuroengineering involves systems of ANNs for control or system identification applications. This work has been mapped out very carefully and reviewed at length (Werbos 1990b, Miller, Sutton and Werbos 1990).

How does the work in that area impinge upon pattern classification?

At the simplest level, ANNs for speech recognition and the like need to be more than static maps, in order to classify speech accurately. It is important that the classification at time t reflect information from earlier times as well. In fact, it is very easy to modify feedforward networks to include time-lagged inputs or memories (a form of "recurrence"), and to use backpropagation in training them. Werbos (1990a) gives a tutorial on how to do this, using the same methods originally discussed in Werbos (1974). Lang et al (1990) have reported reasonable success in classifying speech based on a special case of this general approach. Neural networks with this kind of recurrence are essentially just a nonlinear generalization of the ARMA models which have been used in the past in speech recognition. Lockwood Reed (1990) -- who organized the interagency group on speech recognition in the United States government -- has argued that conventional approaches to speech recognition have reached a plateau, from which large improvements will be ever more difficult; he has argued that neural networks should be able to get beyond that plateau, after a few years of catching up. Carnegie-Mellon -- where Lang and Waibel are working -- also maintains the best performing public domain speech recognition programs based on conventional methods (hidden Markov models, HMM), according to some observers; these methods are said to be roughly as good as the best proprietary packages, and should make an excellent basis for comparison as this work progresses.

In actuality, there are certain difficulties in applying these methods directly to speech data. The information available in speech classification is far less than the information available in the entire speech process; in other words, the sequence of speech labels has less information content than does the entire time-series of speech, by far. Thus from a statistical point of view, it may be desirable to try to exploit the information (variance) in the speech data itself by developing <u>models</u> of the speech process. In effect, this is one more case where an unsupervised learning strategy can serve as a feature generator (at the very least) for a classification process One can do this dynamic modeling, once again, by using backpropagation through time, where the target vectors are the state of the speech process at time t+1. Considerably more robust results can them be had by using more sophisticated approaches (Werbos 1990b), which tend to be better in generating long-term memories Levin (1990) has shown how a hybrid HMM/neural system can be built, which can always outperform conventional HMM, by treating part of the recognition problem as a control problem; this could be extended much further, using advanced neurocontrol methods.

In those recognition problems which do feed directly into a control problem, there may be ways to use the control information (Werbos 1987) to focus <u>attention</u> on key variables, and improve the real-world value of the recognition subsystem; however, I am not aware of any ANN implementations as yet of this idea.

<u>5. Conclusions</u>

ANNs and SPR are both large, diverse areas of research. They both share a large number of paradigms, such as least squares estimation, exploitation of symmetry, clustering, and so on ANNs in pattern recognition may be viewed as a subset of SPR, a subset which is carefully designed so as to allow maximum computational efficiency -- something which is of enormous importance to most practical applications. ANN researchers have also carried out efforts to understand fixed pattern recognition systems in the brain, which can be useful as preprocessors for ANNs and SPR both. ANN researchers also try to maintain links with our understanding of learning in biological brains -- a subject of enormous importance In recent work, many ANN researchers have focused on dynamic systems and control problems, which could allow new and more powerful approaches to pattern recognition as well.

REFERENCES

Ackley, D.H. (1990), Learning from natural selection in an artificial environment. In IJCNN Proceedings (January, Washington D C) Hillsdale, NJ: Erlbaum, p.I-189 to I-193.

Albus, J.S. (1971)., A theory of cerebellar function. Mathematical Bioscience, No. 10, p.25-61.

Bourlard,H. and Wellekens (1988), A link between Markov models and multilayer perceptrons (abstract). In Neural Networks, Volume 1, Supplement 1, p.290.

Box, George E. and Jenkins (1970). Time-Series Analysis: Forecasting and Control. San Francisco, CA: Holden-Day.

N.DeClaris (1990), On the use of higher order nonlinear neural-type junctions in adaptive systems. In K.Narendra (ed.), Proceedings of the Sixth Yale Workshop on Adaptive and Learning Systems. New Haven, Conn: Narendra, Yale U. p.120-126.

R.DeFigueride (1990), personal communication

A.Dempster et al (1977), A simulation study of alternatives to ordinary least squares, Journal of the American Statistical Association, Vol. 27, March.

Fahlman,S (1990), The recurrent cascade-correlation architecture Draft. (Dept. of Computer Science, Carnegie-Mellon U)

Foldiak, P. (1989), Adaptive network for optimal linear feature extraction. In IJCNN Proceedings. IEEE Catalog No. 89CH2765-6 New York: IEEE. p. I-401 to I-405

Foote, S. and Morrison, Extrathalamic modulation of cortical function(1987). Annual Review of Neuroscience Palo Alto, CA: Annual Reviews Inc.

Giles, C.L. and Maxwell (1987), Learning, invariance and generalization in high order neural networks, Applied Optics. Vol. 26, No. 23.

Grossberg, S. (1987), Competitive learning: from interactive activation to adaptive resonance, Cognitive Science, 11, p 23-63

Grossberg, S. (1988), Nonlinear neural networks: principles, mechanisms, and architectures, Neural Networks. Vol 1, No. 1, 1988, p.17-61.

Guyon, I , Poujaud, Personnaz, Dreyfus, Denker and LeCun(1989), Comparing different neural network architectures for classifying handwritten digits. In IJCNN Proceedings. New York: IEEE

Hebb,D.O. (1949), The Organization of Behavior. New York: Wiley.

Hinton, G. (1990), An unsupervised learning procedure that discovers surfaces in random-dot stereograms Oral version of a talk included in IJCNN Proceedings, Hillsdale, NJ: Erlbaum.

Hopfield, J. and D.Tank (1986),Computing with neural circuits: a model, Science. Vol 233, p. 626, August 8

Hrycej, T (1990) A modular architecture for efficient learning. In IJCNN Proceedings (June, San Diego). New York: IEEE p 557-562

Kohonen, T. (1988), Self-Organization and Associative Memory, Second Edition. New York: Springer-Verlag.

Kosko, B. (1990), Stochastic competitive learning IJCNN Proceedings. New York: IEEE, p. II-215 to II-226.

Kraft, L.G. and D. Campagna (1990), A summary comparison of CMAC neural network and traditional adaptive control systems. In Miller, Sutton and Werbos (1990).

Kuperstein, M. (1987), Adaptive visual-motor coordination in multijoint robots using parallel architecture. In Proceedings of the IEEE Conference on Robotics and Automation. Washington D.C.: IEEE Computer Society.

Lang, K., Waibel and Hinton (1990), A Time-delay neural network architecture for isolated word recognition, Neural Networks. January.

LeCun, Y. (1985) Une procedure d'apprentisage pour reseau a seuil assymetrique. Proceedings of Cognitiva 85, p. 599-604. Paris.

E.Levin (1990), Modeling time-varying systems using a hidden control neural network architecture.
In K.Narendra (ed), Proceedings of the Sixth Yale Workshop on Adaptive and Learning Systems New Haven, Conn : Narendra, Yale.

Lukes, G., Thompson and Werbos (1990), Expectation-driven learning using an associative memory. In IJCNN Proceedings (January, Washington D.C.). Hillsdale, NJ: Erlbaum. p. I-521 to I-524.

Lyon, R.F. and Mead, An analog electronic cochlea. IEEE Trans. ASSP, July 1988.

Maren, A. (1990), Handbook of Neural Comp. Appl. Academic Press, scheduled for circa July 1990.

Mead, C. (1989) Analog VLSI and Neural Systems. Reading, Mass.: Addison-Wesley.

Miller, Sutton and Werbos (1990), Neural Networks for Robotics and Control. Cambridge, Mass.: MIT Press (Scheduled for fall catalogue.)

Moody,J. and Darken (1989), Fast learning in networks of locally-tuned processing units,Neural Computation, Vol 1, p 281-294.

Papantoni-Kazakos (1989), personal communication. (University of Virginia, Charlottesville)

Parker, D.B. (1982), "Learning-Logic", Invention Report S81-64, File 1, Office of Technology Licensing, Stanford University, October 1982.

L.Reed (1990), oral presentation, Fort Monmouth (U.S. Army), New Jersey.

Rumelhart, D (1990), unpublished address, IJCNN, January, Washington D.C.

Rumelhart, D , Hinton and Williams (1986), Learning internal representations by error propagation In Rumelhart and McClelland, eds , Parallel Distributed Processing, Volume 1. Cambridge, Mass : MIT Press.

Shamma, S. (1987), Neural networks for speech processing and recognition. In IJCNN Proceedings. New York: IEEE. p IV-397 to IV-405.

Shanno, D. (1990), Recent advances in numerical techniques for large-scale optimization. In Miller, Sutton and Werbos (1990).

Solomonoff, R.J , A formal theory of inductive inference, Inform. Contr., March/June.

Sun,J. and Hassoun (1990), A fast algorithm for finding global minima of error functions in layered neural networks In IJCNN Proceedings (San Diego, June). New York: IEEE. p.I-715 to I-720.

Stinchcombe, M. and H. White (1989). Universal approximation using feedforward networks with non-sigmoid hidden layer activation function. In IJCNN Proceedings. New York: IEEE.

Sutton, G.G, Reggia and Maisog, Supervised and reinforced competitive learning In IJCNN Proceedings (San Diego, June) New York: IEEE. p. I-563 to I-567.

Tishby, N, Levin and Solla (1989), Consistent inference of probabilities in layered networks: predictions and generalization. In IJCNN Proceedings. New York: IEEE p.II-403 to II-409.

Werbos, P. (1968) Elements of intelligence, Cybernetica (Namur), No. 3. p.131-143 and p.170-178.

--------- (1974). Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Harvard U. Ph.D. thesis, committee on applied mathematics, November 1974

--------- (1977), Advanced forecasting methods for global crisis warning and models of intelligence, General Systems Yearbook.

__________ (1978), Learning how the world works: specifications for predictive networks in robots and brains. In IEEE/SMC Proceedings, Volume 1. IEEE No. 87CH2503-1. New York: IEEE.

--------- (1981), Applications of advances in nonlinear sensitivity analysis. In Drenick and Kozin, eds., System Modeling and Optimization: Proceedings of the 10th IFIP Conference (September 1981). New York: Springer-Verlag, 1982.

----------(1988a). Backpropagation past and future. In ICNN Proceedings. New York: IEEE. A transcript of the talk, which is more colloquial, a bit broader in scope, and contains the slides, is available from the author upon request.

--------- (1988b). Generalization of backpropagation with application to a recurrent gas market model, Neural Networks, October

--------- (1989), Maximizing long-term gas industry profits in two minutes in Lotus using neural network methods, IEEE Trans. SMC March/April.

--------- (1990a), Backpropagation through time: what it does and how to do it, Proceedings of the IEEE. October special issue on neural networks.

--------- (1990b), Neurocontrol an related techniques. In A Maren (ed.), Handook of Neural Comp. Appl. Academic Press, 1990

--------- (1990c), Econometric techniques: theory versus practice, Energy: The International Journal. March/April

---------- (1990d), Stochastic modeling and representations of reality: a linear starting-point. Submitted to Neural Networks, spring 1990.

__________ (1990e), Neurocontrol and fuzzy logic: connection and options, International Journal of Approximate Reasoning, fothcoming special issue on fuzzy logic and neural networks.

Wonnacott, T.H. and Wonnacott (1977), Introductory Statistics for Business and Economics, Second Edition. New York: Wiley

Yau, H.C and Manry (1990), Sigma-pi implementation of a nearest-neighbor classifier In IJCNN Proceedings (San Diego, June) New York: IEEE.

Artificial Neura
Old and New C
I.K. Sethi and A
© 1991 Elsevie

Small sa
neural n

Šarūnas Rau

[a]Department
Vilnius 2326

[b]Department
USA

Abstract

Small tra
network clas
sented, and p
number of ne

1. INTROD

Artificial
nique for pat
feedforward n

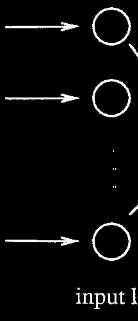


Figure 1: An

[1]Research s

Artificial Neural Networks and Statistical Pattern Recognition
Old and New Connections
I.K. Sethi and A K. Jain (Editors)
© 1991 Elsevier Science Publishers B V All rights reserved

Small sample size problems in designing artificial neural networks[1]

Šarūnas Raudys[a] and Anil K. Jain[b]

[a]Department of Data Analysis, Institute of Mathematics & Cybernetics, Akademijos 4, Vilnius 232600, Lithuania (USSR)

[b]Department of Computer Science, Michigan State University, East Lansing, MI 48824, USA

Abstract

Small training sample effects common in statistical classification and artificial neural network classifier design are discussed A review of known small sample results are presented, and peaking phenomena related to the increase in the number of features and the number of neurons is discussed.

1. INTRODUCTION

Artificial neural networks are now widely recognized as a useful classification technique for pattern recognition [31] A typical artificial neural network (ANN) classifier (a feedforward network) consists of several layers of neurons (see Figure 1) Each (say i^{th})

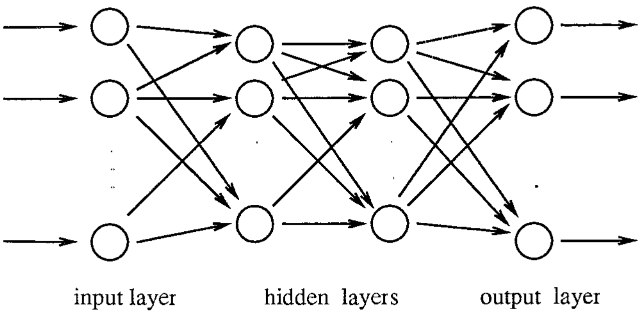


Figure 1: An ANN With Two Hidden Layers

[1]Research supported in part by NSF grant CDA-8806599

neuron has several (say d) inputs $x_1, x_2, \ldots, x_d$, one output y_i, and performs an operation

$$y_i = f_i(\sum_{j=1}^{d} w_{ij}x_j + w_{i0}), \tag{1}$$

where f_i is a nondecreasing and differentiable activation function, and w_{ij} is the weight assigned to the j^{th} input of the i^{th} neuron Examples of such activation functions are hard limiting or soft limiting threshold functions and Huber's and Tukey's functions [33]. The neurons in the input layer correspond to the components of the feature vector to be classified. In the feedforward network which will be discussed here, the inputs to the neurons in each successive layer are the outputs of the preceeding layer The neurons in the output layer are usually associated with pattern class labels.

The important design issues in building an ANN classifier are to find an appropriate network topology (number of hidden layers, number of neurons in each layer) and to learn the weights w_{ij} for each neuron from the given training samples. If an one-layer ANN classifier with a single neuron and hard limiting threshold activation function (a simple perceptron [46]) is used, then a linear discriminant function is realized and the resulting decision surface is a hyperplane [32] On the other hand, a multilayer ANN with soft limiting threshold activation function can realize an arbitrarily complex decision surface [6, 17, 30, 31, 53]. A number of methods exist to train an ANN [18, 31, 32, 47, 52] These training methods differ in the error function and in the optimization technique used to determine the weights in the neural network Let y_i be the actual output and o_i be the desired output of the i^{th} neuron in the output layer of ANN. The most popular error function is the mean square error function defined as

$$MSE = \sum_{j=1}^{n}\sum_{i=1}^{p} \varepsilon(y_{ij} - o_{ij}) = \sum_{j=1}^{n}\sum_{i=1}^{p}(y_{ij} - o_{ij})^2, \tag{2}$$

where n is the number of training samples, p is the number of neurons in the output layer, and $\varepsilon(\ )$ denotes the error function

The ANN classifier can be analyzed as a special case of statistical pattern classifiers which are "data-driven", in the same spirit as Parzen-window classifiers and K-NN classifiers [8] It is well known that, in a finite training sample case, the expected classification error EP_N of a statistical pattern classifier can increase as number of features are increased due to the inaccuracies in estimating the parameters of the classifier The finite number of training samples causes the following practical difficulties and constraints in designing a classifier [3, 8, 9, 19, 21, 24, 37, 42]:

1 The resubstitution error rate has an optimistic bias.

2 A peaking in classification performance is often observed as the number of features increases

3 A simple classification algorithm (e.g., a linear discriminant function) may outperform a more complex classification algorithm (e g., a quadratic discriminant function).

4. A nonparametric decision rule may outperform a parametric decision rule even if the assumed parametric model is correct

5. There is an optimum value of K in the K-NN decision rule

6. The choice of window width is critical in the performance of a Parzen window classifier.

7. In the case of unequal numbers of training samples per class, the decision boundary may need to be "balanced"

We believe that ANN classifiers will also encounter similar difficulties and constraints when the number of training samples is small The purpose of this paper is to analyze the small sample size effects that occur in the design of ANN classifiers

In Section 2 we present some known results concerning the influence of the number of training samples on the accuracy of several well known parametric and nonparametric statistical classifiers. These results will be useful in analyzing similar small sample size effects for ANN classifiers. In Section 3 we discuss the classification accuracy and training time of ANN classifiers Section 4 deals with the problem of estimating classification error. In Sections 5 and 6, we analyze the "peaking" phenomena which arises due to an increase in the number of inputs and the number of hidden layer nodes in multilayer ANN classifiers Section 7 consists of discussion, and suggestions for future research.

2. FINITE SAMPLE PROBLEMS IN STATISTICAL PATTERN RECOGNITION

One of the most popular and simplest statistical pattern classifier is the Fisher linear discriminant function (LDF)

$$g(x) = \sum_{i=1}^{d} w_i x_i + w_0, \tag{3}$$

where $x_1, x_2, \ldots, x_d$ denote the d features, and $w_0, w_1, \ldots, w_d$ are constants

For a two-class problem, if $g(x) > 0$ then the feature vector $x = (x_1, x_2, \ldots, x_d)^T$ is allocated to class π_1, otherwise to class π_2 The linear discriminant function linearly maps the training patterns from each class on the real line defined by the weight vector $w = (w_0, w_1, w_2, \ldots, w_d)^T$. Fisher [11] chose the weight vector such that the mean squared deviation of the projected training patterns around their class mean vectors (within-class scatter) is minimized with respect to the separation between the projected class mean vectors (between-class scatter). The weights of this LDF are identical to those obtained from the "plug-in" decision rule for the case of two Gaussian class-conditional density functions when the unknown mean vectors and the common covariance matrix are replaced by their maximum likelihood estimates The same weight vector can also be obtained by the least-mean-square-error adaptation algorithm with an equal number of training patterns from both classes [14, 26] Therefore, the linear discriminant function is, in fact, an one-layer perceptron trained by the standard delta rule learning algorithm [47]

The expected probability of misclassification, EP_N, of LDF can be written as,

$$EP_N = q_1 P\{g(x) \leq 0 | x \in \pi_1\} + q_2 P\{g(x) > 0 | x \in \pi_2\}, \tag{4}$$

where q_1 and q_2 are prior probabilities of classes π_1 and π_2, respectively The expected probability of misclassification, EP_N, of the LDF depends on the number of training samples per class N_1 and N_2 , dimensionality of the feature vector d, and the asymptotic probability of misclassification, P_∞,

$$P_\infty = \lim_{N \to \infty} EP_N$$

It is not easy to obtain a simple analytic expression for EP_N, so Raudys and Pikelis [43] provided a table showing this dependence for various values of N, d, P_∞, and the Mahalanobis distance between the two class-conditional density functions. The relative increase in the expected probability of misclassification (EP_N/P_∞) increases with an increase in dimensionality and the asymptotic probability of misclassification, and decreases with an increase in the number of training samples.

Pikelis [35] compared a number of asymptotic expansions of the expected probabilty of misclassification for LDF and found that Deev's expansion [7] is the most exact. In the case of Gaussian class-conditional density functions with a common covariance matrix, the second term on the right hand side of Equation (4) can be computed as follows [7]

$$P\{g(X) > 0 | X \in \pi_2\} = \phi \left\{ -\frac{\delta}{2} \frac{1 + \frac{N_2 - N_1}{N_1 N_2} \frac{d-1}{\delta^2} + \frac{2(d-1)}{\delta^2 N_i} \frac{N_1 + N_2 - d - 1}{N_1 + N_2 - 2}}{\sqrt{\frac{N_1 + N_2 - 1}{N_1 + N_2 - d} \frac{N_1 + N_2 + 1}{N_1 + N_2} \left(1 + \frac{N_1 + N_2}{N_1 N_2} \frac{d-1}{\delta^2}\right)}} \right\}, \tag{5}$$

where N_i is the number of training samples from class $\pi_i, \phi(c)$ is Laplace's probability integral and δ^2 is the squared Mahalanobis distance, $\delta^2 = (\mu_1 - \mu_2)^t \sum^{-1} (\mu_1 - \mu_2)$ The first term in Equation (4)

$$P\{g(X) \leq 0 | X \in \pi_1\} = 1 - P\{g(X) > 0 | X \in \pi_1\}$$

can be obtained from Equation (5) by interchanging N_1 and N_2

Equation (5) shows that if $d \to (N_1 + N_2 - 2)$, i.e. when the estimated covariance matrix becomes singular, the misclassification error increases enormously Due to the nonlinear nature of the Laplace integral $\phi(c)$, the term

$$\left(\frac{N_i - N_j}{N_i N_j}\right) \left(\frac{d-1}{\delta^2}\right), \; i = 3 - j, \; j = 1, 2 \tag{6}$$

will increase the classification error when $N_1 \neq N_2$ This degradation in classification performance due to unequal numbers of training samples from different classes is significant when a quadratic dicriminant function is used in high dimensional cases [38, 42]

In the standard delta learning rule, the weights $w_0, w_1, \ldots, w_d$ of the linear discriminant function are found by minimizing the mean-square-error (Equation(2)) Other, more complicated, criteria can also be used [48, 49, 50]. For example, a minimization of the modulus criterion,

$$\varepsilon(c) = |c|, \tag{7}$$

in a second-order perceptron (where new derived features involving quadratic or exponential terms are used) can result in an optimal Bayes decision boundary [48]

In many learning algorithms, the weights $w_0, w_1, \ldots, w_d$ are changed only when a training vector X_p is incorrectly classified (error-correction algorithms). For example, in the relaxation algorithm REL,

$$\varepsilon(c) = \begin{cases} c^2, & \text{when } X_p \text{ is incorrectly classified,} \\ 0, & \text{otherwise,} \end{cases} \tag{8}$$

and in the fixed increment algorithm FIX

$$\varepsilon(c) = \begin{cases} |c|, & \text{when } X_p \text{ is incorrectly classified,} \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

When an empirical probability of misclassification is minimized then we use a hard-limiting threshold function

$$\varepsilon(c) = \begin{cases} 1, & \text{when } X_p \text{ is incorrectly classified,} \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

Smith [49] derived the following approximate formula to calculate the expected probability of misclassification of MSE, REL and FIX adaptation algorithms for linear discriminant functions

$$EP_N^{\alpha} = P_{\infty}^{\alpha} + \frac{1}{2N}(A^{\alpha} + B^{\alpha}(d-1)), \tag{11}$$

where parameters A^{α} and B^{α} depend on the type, α, of the error function (MSE, REL, FIX) and on the asymptotic probability of misclassification (see Table 1) Consider a

Table 1: The coefficients A and B in Equation (11) [49]

Algorithm		MSE		REL		FIX	
δ	P_{∞}	A	B	A	B	A	B
1.0	0.309	.0880	4400	0886	.0441	0152	505
2.0	0.159	.1210	2420	.1400	.2610	.1960	.317
4.0	0.023	.0540	.0675	.1680	.1820	.2180	.232

classification problem for which the Mahalanobis distance $\delta = 2$, dimensionality d=10, and training sample size $N = N_1 = N_2 = 20$, then Equation (11) results in $EP_N^{MSE} = 0.2165$ for the standard delta rule with MSE criterion (the exact value from Pikelis' table [35] is 0.219 and Deev's [7] main term in Equation (5) gives 0.217); $EP_N^{REL} = 0.221$ for the relaxation error criteria (Equation (8)), and $EP_N^{FIX} = 0.235$ for the fixed increment criteria (Equation (9)). With an increase in the Mahalanobis distance δ (or a decrease in the asymptotic probability of misclassification P_∞), the role of training samples in determining the weights of the linear discriminant function is diminished and, as a result, the differences between the expected errors EP_N^{MSE}, EP_N^{REL} and EP_N^{FIX} increase. When one uses minimum empirical classification error criteria (Equation (10)), the classification error is significantly higher than that of the classifiers which use the MSE, REL and FIX criteria [51]. Therefore, the standard delta rule where the mean-square-error criterion is minimized is the most preferable learning rule to design ANN classifiers in terms of its sensitivity to the training sample size.

The above results concerning the linear discriminant function and the mean-square-error delta adaptation algorithm are valid when only a single-layer ANN performs the classification, and when a hard-limiting threshold activation function is used in Equation (1). When one uses a soft limiting activation function in a two-layer ANN, then Equation (1) in fact performs a feature extraction procedure. In other words, the outputs of the hidden layer define a new feature space where simple classes can sometimes be comparatively easily separated by adjusting the weights of neurons in the output layer.

It is well known that a multi-layer ANN classifier can form complex decision boundaries similar to nonparametric Parzen window or K-Nearest Neighbor classifiers [17, 30, 31]. Therefore, a knowledge of the sensitivity of the nonparametric statistical classifiers to the finiteness of the training sample size can serve as a guide in the analysis of multi-layer ANN classifiers. Unfortunately, very little is known about the behavior of nonparametric statistical classifiers in the finite sample case.

Raudys [38] obtained some results by means of simulation studies for nonparametric Parzen window classifiers. The classification problem involved two multivariate Gaussian populations with identity covariance matrices. He used the following window function:

$$K(\frac{X - X_p}{\lambda}) = C * \exp\{-\frac{(X - X_p)^T(X - X_p)}{\lambda^2}\}, \tag{12}$$

where λ is the window width, X_p is a training sample, and C is a constant.

The relative increase in the classification error (EP_N/P_∞) is presented in Table 2 for two values of the smoothing parameter ($\lambda = 0.1, 0.8$). The increase in the error rate of the Parzen window classifier to the training sample size drops when the value of the smoothing parameter increases. When $\lambda \to 0$, a Parzen window classifier with the Gaussian window function performs no "smoothing" and its performance becomes similar to an 1-NN classification rule. Thus, values of $\kappa = EP_N/P_\infty$ presented for $\lambda = 0.1$ are practically the values of κ for the 1-NN classifier. Note that the increase in the error rate of the nonparametric statistical pattern classifiers with an increase in number of features d is more significant than for parametric MSE adaptation rule [42]. The decrease of the classification error with an increase in the training sample size is also slower for nonparametric classifiers. Therefore, in order to design complex decision boundaries in a high dimensional feature space with the help of nonparametric statistical classifiers, a very large number of training samples is required. We suspect that in a two-category case, the complex multi-layer neural network classifiers will also have similar behavior.

Table 2: The values of the relative increase in the expected probability of misclassification $\kappa = EP_N/P_\infty$ of Parzen window classifier for two Gaussian classes with common identity covariance matrices [38].

N	d=3		N	d=5		N	d=8	
P_∞	0.1	0.01	P_∞	0.1	0.01	P_∞	0.1	0.01
δ	2.56	4.65	δ	2.56	4.65	δ	2.56	4.65
2	1.97/1.96	3.53/3.51	3	2.15/2.12	3.55/3.51	5	2.28/2.26	4.29/4.21
3	1.90/1.80	2.92/2.86	5	1.98/1.94	3.24/3.15	8	2.13/2.05	3.48/3.41
6	1.78/1.68	2.61/2.51	10	1.87/1.80	3.07/2.87	16	1.95/1.90	3.18/3.08
15	1.64/1.46	2.32/2.18	25	1.71/1.58	2.56/2.38	40	1.91/1.76	3.72/2.46
30	1.50/1.23	2.15/1.71	50	1.66/1.44	2.16/1.90	80	1.84/1.65	2.28/2.10
150	1.39/1.06	1.53/1.20	250	1.62/1.12	1.86/1.27	400	1.81/1.29	2.14/1.54

The search for the appropriate architecture and the weights of an ANN classifier is an optimization problem for a given error function. This optimization problem itself in fact involves selecting a variant of an ANN classifier among an immense number of all possible ANN classifiers with a given architecture. Raudys [41, 44, 45] analyzed a problem where the best model needs to be selected from an infinite general population ($\mathcal{M}$) of the models. Let the i^{th} model $M_i \in \mathcal{M}$ be characterized by some value of the error function P_i, and there exists an estimate $\hat{P}_i$ of P_i. It is assumed that $P_1, P_2, \ldots$ are random variables with a density function $f(P)$. Similarly, $\hat{P}_1, \hat{P}_2, \ldots$ are random variables with the conditional density function $f(\hat{P}_i|P_i)$. Due to inaccuracies in the estimates $\hat{P}_1, \hat{P}_2, \ldots$, the selection of the best model according to the estimates $\hat{P}_1, \hat{P}_2, \ldots$ results in an increase in the value of the true error function P_{true} compared with the value of the error P_{ideal} in an ideal selection procedure which uses the true values $P_1, P_2, \ldots$.

In Table 3 we present estimates of the relative mean increase in classification error $\kappa = EP_{true}/EP_{ideal}$ obtained for a mathematical model when true probabilities of misclassification $P_1, P_2, \ldots$ have a generalized Beta distribution with parameters $P_{min}, P_{max}, \gamma = 4.0, \eta = 4$ and the best model is selected from 10^6 models randomly chosen from the general population $\mathcal{M}$ [41]. The estimates $\hat{P}_1, \hat{P}_2, \ldots$ were obtained by using n test samples, where $n\hat{P}_j$ is Binomially distributed with parameters P_j and n. This methodology was also applied to the problem of selecting the best subset of features [45].

The theoretical values of the relative increase in the classification error due to an inaccurate selection of the best model provide only guidelines for real model selection tasks. They show, however, that when the sample size is small, the increase in the true classification error is rather significant and there is only a small chance that a good model will be selected. The same conclusion is valid for the ANN optimization problem. The performance of the complex multi-layer ANN classifier trained on a small number of samples will differ significantly from that of the ANN classifier ideally trained on an arbitrarily large number of samples.

Unfortunately, the above theoretical model does not allow us to evaluate the influence of dimensionality, number of hidden layers, and other parameters which define the architecture of ANN classifiers. Table 3 also shows that the relative increase in the clas-

Table 3: The values of the relative increase in the classification error $\kappa = EP_{true}/EP_{ideal}$ when inexact sample-based estimates of the classification error [41] were used to select the best model.

	P_{min}/P_{max}			
N	0.2/0.3	0.1/0.2	0.03/0.5	0.01/0.1
25	2.3	2.7	4.5	8.9
50	2.1	2.5	3.8	7.1
100	1.8	2.2	3.1	5.2
200	1.5	1.8	2.4	3.8
500	1.3	1.5	1.7	2.6

sification error due to inexact sample-based optimization criterion depends on the error rate of the optimized classifier P_{min}. When P_{min} is small, the relative increase is large. The absolute value of the increase in the classification error is of the same order as the standard deviation of the estimates $\hat{P}_1, \hat{P}_2, \ldots$ used as the criterion to find the best weights of an ANN classifier

3. THE CLASSIFICATION ACCURACY AND TRAINING TIME OF ARTIFICIAL NEURAL NETWORKS

Multi-layer neural network classifiers represent a wide class of classification rules. By changing the shape of activation function $f(\ )$ in Equation (1), and the architecture of the ANN, one can obtain classification algorithms with different derived features.

It is generally known [31] that traditional multi-layer artificial neural network classifiers trained by the back propagation algorithm require a great number of sweeps of the training sample data in order to minimize empirical classification error

For a given pattern recognition problem and a given set of training samples, the complexity of the decision region required to obtain a good classification accuracy of the training samples is fixed On the other hand, an ANN with a large number of hidden layers and a large number of neurons in the hidden layers will have more degrees of freedom and will require less accurate determination of the weights to achieve the desired classification accuracy. Therefore, for a given set of training samples, it will be easier to train a complex neural network than a simple one. This observation is supported by numerous experimental studies [2, 16, 28, 34, 47]. In an analysis of the XOR problem with a two-layer neural network, Rumelhart et al. [47] propose an empirical equation to determine the number, S, of presentations of training samples as a function of the number H of the hidden nodes:

$$S = 280 - 33 \log_2 H. \tag{13}$$

Kung and Hwang [28] present a graph of the convergence time (i.e., number of training sweeps) versus the number of hidden nodes per layer when various numbers of hidden layers are used There were n=8 pairs of randomly generated 12-dimensional input and

7-dimensional output training patterns They observed that a network with more hidden nodes per layer and with more layers led to a smaller number of training sweeps. They also noted an abrupt reduction in the number of training sweeps around n-1 hidden nodes (n is the number of training patterns). This observation agrees with a well-known fact in statistical pattern recognition theory, that in a d-dimensional case, a hyperplane can discriminate perfectly any set of d+1 points with arbitrary class labels [4]. Therefore, when the number of hidden units is equal to n-1 there is no need to adjust the weights of the hidden layer units! We repeat the experiment on two-layer ANN classifiers and show the result in Figure 2.

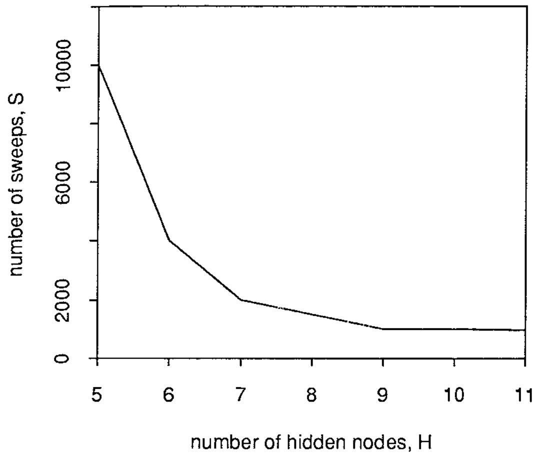


Figure 2: The convergence time (i e , number of training sweeps S) versus the number of hidden nodes per layer in the two-layer ANN classifiers.

With an increase in the number of training samples, the complexity of the decision boundaries required for perfect discrimination of the training samples increases Therefore, "training times are typically longer when complex decision boundaries are required and when networks have more hidden layers" [31]

We pointed out in Section 2, that in selecting the best classifier on the basis of inaccurate sample-based estimates $\hat{P}_1, \hat{P}_2, \ldots$, the difference between the true (test sample estimate) classification error of the "best" classifier and the ideal one (obtained when selection is performed on the basis of exact values $P_1, P_2, \ldots$) decreases with an increase in the number of training samples used to obtain the estimates $\hat{P}_1, \hat{P}_2, \ldots$ Besides the true classification error P_{true} and the ideal classification error P_{ideal} in the selection process, there exists an apparent error $P_{apparent}$, i.e. the minimal value among the estimates $\hat{P}_1, \hat{P}_2, \ldots, \hat{P}_m$ (here m is the number of classifiers compared empirically). The mean value of the apparent error ($EP_{apparent}$) is less than that of the ideal error EP_{ideal} The difference, $(EP_{ideal} - EP_{apparent})$, decreases with an increase in the training sample size used to obtain estimates $\hat{P}_1, \hat{P}_2, \ldots, \hat{P}_m$ In the best classifier selection problem, while using the random optimization search procedure, the values of the true, ideal, and apparent errors depend on the number m, i e , the size of the set from which the "best" classifier is selected All three error rates decrease with an increase in m; the apparent error decreases most quickly, and the true error most slowly (see, for example, curves in Figure 3 obtained for the Beta-Binomial distribution model discussed in section 2). In experiments

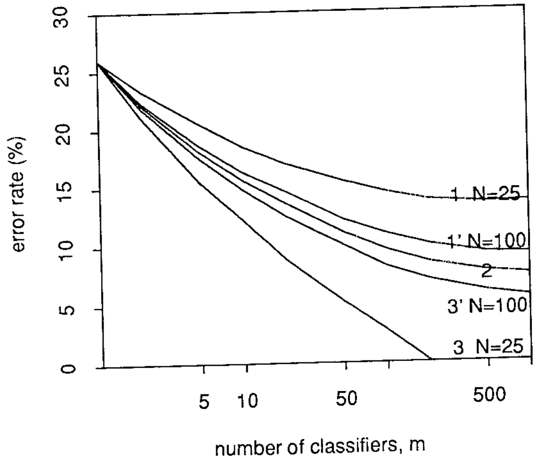


Figure 3: The mean values of the true error EP_{true} (1,1'), ideal error EP_{ideal} (2) and apparent error $EP_{apparent}$ (3,3') in best classifier selection versus number, m, of classifiers compared (Beta-Binomial distribution of $(\hat{P}_j, P_j)$ [41]).

with feature selection, when the number m is sufficiently large, the true error ceases to decrease, and merely oscillates [41] Therefore, with finite training sample size, there is no need to analyze a very large number of classifiers if they were randomly chosen from the set of all possible classifiers The same conclusion can be drawn for ANN optimization The optimization itself is a selection of the best classifier from the infinitely many possible classifiers with a given architecture Simulation studies confirm this theoretically obtained conclusion In Figure 4, we present two such pairs of graphs obtained by means of simulation Similar graphs were obtained by le Cun et al [5] while solving a handwritten digit recognition problem by means of a 3-layer artificial neural network.

Therefore, in the ANN training problem, an excessive amount of optimization of the neural network weights is not necessary if the training sample size is small The optimal number of sweeps required to minimize the true classification error (the test sample estimate) depends on the number of training samples. It increases with an increase in sample size; however, theoretically-based recommendations for training effort required in

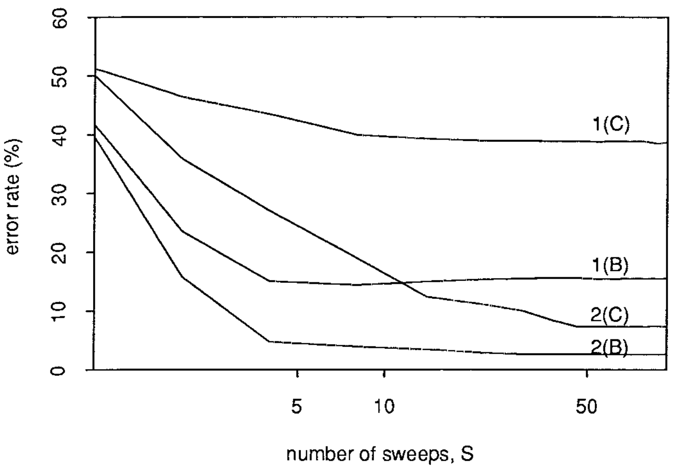


Figure 4: True error (1(B),1(C)) and apparent error (2(B),2(C)) versus number of sweeps in the back propagation algorithm ($H = 4, d = 12, N = 100$, two spherically Gaussian populations $N(\mu_i, I\delta_i^2)$; $\mu_i = (\mu_{i1}, \ldots, \mu_{id})^T$; data set B: $\mu_{ij} = (-1)^i/j, \delta_i^2 = 1, i = 1, 2; j = 1, \ldots, d$; data set C: $\mu_1 = \mu_2, \delta_i^2 = 4^{i-1}, i = 1, 2$)

practical problems do not yet exist.

The analysis presented above shows that instead of minimizing the apparent (training sample) error, one should minimize the true (test sample) error

4. ESTIMATION OF THE CLASSIFICATION ERROR

A number of techniques exist to estimate the classification error in statistical pattern recognition [13, 15, 20]; It is well known that the resubstitution estimate is optimistically biased Use of the resubstitution method to estimate error rate of an ANN classifier in the small training sample case will also result in a biased estimate Therefore, in a finite design sample case, the hold-out method (where independent test samples are used to estimate the classification error) is preferable.

Dutta and Shekhar [10] present the following resubstitution ($\hat{P}_R$) and hold-out ($\hat{P}_H$) estimates, obtained for a three-layer ANN classifier with 6- and 10-dimensional feature vectors when the total number of training samples from 4 classes was 30 and the test sample size was 17

$\hat{P}_R = 0.20$ for d=6 and $\hat{P}_R = 0.076$ for d=10,
$\hat{P}_H = 0.235$ for d=6 and $\hat{P}_H = 0.176$ for d=10

We notice a significant difference between $\hat{P}_R$ and $\hat{P}_H$ in the 10-dimensional case. The difference $(\hat{P}_H - \hat{P}_R)$ can serve as a criterion to evaluate the sufficiency of the training sample size. Thus, for d=6 we can conclude that the training sample size is sufficient, since $\hat{P}_R = 0.2$ and $\hat{P}_H = 0.235$, and for d=10 the training sample size is not sufficient, since $\hat{P}_R = 0.076$ and $\hat{P}_H = 0.176$.

It is important to remember that the estimates $\hat{P}_R$ and $\hat{P}_H$ are random variables. Their standard deviations can be approximately evaluated by the following equation [12, 36, 39].

$$SD(\hat{P}_e) = \sqrt{\frac{E\hat{P}_e(1 - E\hat{P}_e)}{n_t}}, \tag{14}$$

where E denotes the expection operator and n_t is the number of samples used to obtain the error estimate $\hat{P}_e$ (here $e = R$ or H).

In solving practical pattern recognition problems, a researcher needs to select the "best" classifier and evaluate its performance empirically. Even when independent test samples are used to estimate the classification error of the best classifier, the result will still be optimistically biased [41, 44]; the test samples take the role of additional training samples. If several classifiers are evaluated and compared, then the bias mentioned above can become significant; the bias can exceed the standard deviation (Equation (14)) of the estimate $\hat{P}_H$ used to select the best version [45]. Therefore, one has to remember that in evaluation of the performance of the ANN classifier, an independent test sample, *never* used before, should be used to obtain an unbiased error rate.

5. PEAKING IN THE CLASSIFICATION PERFORMANCE WITH INCREASE IN DIMENSIONALITY

It is well known that introducing additional features never increases the classification error of the optimal Bayes classifier. However, in a finite training sample case, or when one uses an inappropriate statistical model for the real data, the addition of new features can actually increase the classification error. The additional discriminatory information provided by the new features is outweighed by the increase in the inaccuracy of parameter estimates needed in the classification rule. Thus a peaking phenomenon is observed: addition of new features decreases the classification error at first, then the error levels off, and begins to increase [1, 8, 9, 19, 22, 24, 29, 37, 40]. The peaking phenomenon is also observed in the design of ANN classifiers. A single linear threshold element trained by the delta rule in a finite training sample case will have the same behaviour as the standard linear discriminant function. In the latter case, the optimal number of features, d_{opt}, is a function of the asymptotic probability of error P_∞, the number of features d, and the training sample size N [22]. If "best" features (providing the most discriminatory information) are added first and these best features are significantly better than the worst ones, then d_{opt} will be small. If the discriminatory information provided by the individual features is approximately equal, or if we include them in the classifier in a random order then $d_{opt} \approx N - 1$ [22].

When one uses a piecewise linear classifier with H linear hyperplanes, then the "effective" number of training samples used to determine the weights of each linear boundary segment is approximately [23]

$$N^* = N/H, \tag{15}$$

where N is the number of training samples per class. A two-layer neural network classifier with a hard-limiting activation function gives a piecewise-linear decision boundary. One can, therefore, expect that when H is small, the number of observations used to adjust the weights of each hidden layer element will be approximately equal to $N^* = N/H$, where H is the number of neurons in the hidden layer

In Figure 5 we present several graphs that illustrate peaking phenomena. A two-layer ANN classifier with H neurons in the hidden layer was trained by back propagation rule with a sigmoidal activation function and was used to classify two spherically Gaussian pattern classes. A set of 1000 independent test samples was used to estimate the error rate of the ANN classifier Graphs were obtained by averaging the results of sixty Monte Carlo trials with different training sample sets of fixed size and different initial weights. Similar results, which demonstrate the peaking with increase in dimensionality, were obtained by Kohonen et al [27] Graph B in Figure 5, obtained for a case where the discriminating power of the features drops very slowly with the increase in the number of features, does not exhibit the peaking phenomenon

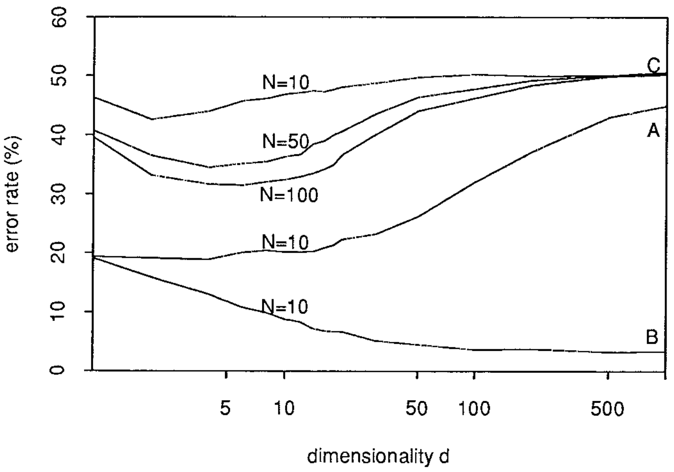


Figure 5: Expected probability of misclassification EP_N versus dimensionality d (Two classes of spherically Gaussian data $N(\mu_i, I\delta_i^2)$, $\mu_i = (\mu_{i1}, \ldots, \mu_{id})^T$; data set A: $\delta_i = 1, \mu_{ij} = (-1)^i/j, i = 1,2; j = 1, \ldots, d$, $H = 2$; data set B: $\delta_i = 1$, $\mu_{ij} = (-1)^i/\sqrt{j}$, $i = 1,2; j = 1, \ldots, d$, $H = 2$; data set C: $\mu_1 = \mu_2, \delta_i^2 = 4^{i-1}, i = 1,2$, $H = 8$)

6. EFFECT OF THE NUMBER OF NEURONS IN THE HIDDEN LAYER ON THE PERFORMANCE OF ANN CLASSIFIERS

It is obvious that the classification error of an ideally trained neural network classifier cannot be increased by introducing new hidden layer neural elements. With an increase in the number of hidden layer elements, the classification error of the ideally trained ANN classifier, P_∞, will fall sharply at first, then more slowly, and eventually, the addition of new elements will not effect P_∞ However, for finite number of training samples, numerical evaluations indicate the existence of the peaking phenomenon as the number of neurons in the hidden layer increases Several graphs that illustrate the peaking phenomenon while increasing the number of hidden neurons for two classes of Gaussian data are presented in Figure 6 Khotanzad and Lu [25] trained a two-layer ANN classifier to recognize the English alphabet Training data consisted of differently positioned, scaled, and oriented 64×64 binary images of each of twenty six English characters, 12 images per character. The performance of the ANN classifier peaked at approximately 50 neurons (2% error) in the hidden layer, and with an increase in H, saturated at an error rate of 5%

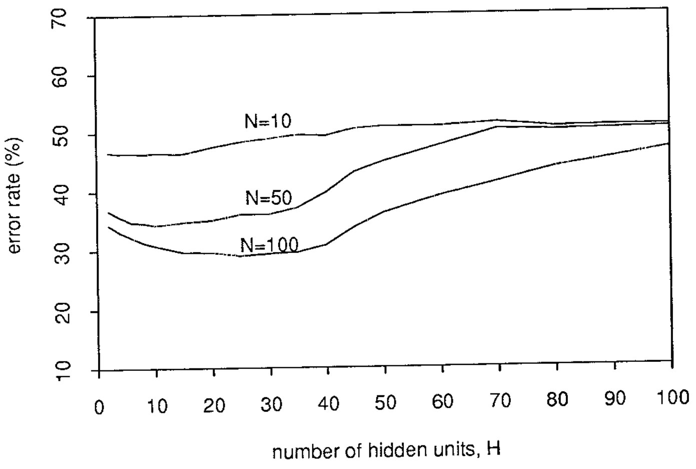


Figure 6: Expected probability of misclassification EP_N versus the number of neurons H in the hidden layer (Two classes of spherically Gaussian populations, $N(0, I\ 4^{i-1})$, $i = 1, 2$, number of features d=8, training sample size N=10,50,100)

7. DISCUSSION

Artificial neural network classifiers can be analyzed as a special class of statistical pattern recognition algorithms In the case of small number of training samples, several unexpected and counterintuitive behavior can sometimes be observed in the design of the ANN classifiers Some of these are listed below

1 Increase in the true (test sample) classification error due to small size of the training samples;

2. Optimistic bias in the apparent (training sample) error rate;

3. Increase in the number of sweeps required to minimize the true and apparent classification error rates of the ANN classifier with an increase in the number of training samples;

4 An optimistic bias in the error rate of the best version of an ANN classifier selected from several competing models based on small number of test samples;

5. A peaking in classification performance with an increase in the number of features and the number of neurons in hidden layers of the ANN classifier.

In spite of the extensive development of ANN theory and applications, many important theoretical problems are far from being solved, and very few quantitative results are available There are several small sample problems which need to be solved:

1 The role of the shape of the nonlinear activation function and the number of units in the hidden layer on the sensitivity of two-layer ANN classifier to the finiteness of the training sample.

2. Dependence of the true (test sample) error and the apparent (training sample) error on the training time (number of sweeps) and the architecture of the ANN in the finite training sample case.

3 Designing fast training algorithms which minimize the true error instead of minimizing the apparent error.

4 A procedure to determine the optimal number of neurons in the hidden layer, in accordance with the training sample size and the problem complexity

Acknowledgements — Jianchang Mao carried out several experiments reported in this paper The first author would like to acknowledge the visiting scholarship provided by the Department of Computer Science, Michigan State University.

References

[1] D C Allais The problem of too many measurements in pattern recognition. *IEEE Int. Con. Rec.*, (part 7):124–130, 1966

[2] D J. Burr Experiments on neural net recognition of spoken and written text *IEEE Trans Acoust , Speech, Signal Process* , 36(7):1162–1168, July 1988

[3] B Chandrasekaran and A K Jain On balancing decision functions *J Cybernet Inform. Sci.*, 2:12–15, 1979

[4] T M Cover Geometrical and statistical properties of systems of linear inequalities with application in pattern recognition *IEEE Trans Elect Comp.*, 14:326–334, 1965

[5] Y.L. Cun, L.D Jackel I Guyon, D. Henderson, B. Boser, R E Howard, J S. Denker, W Hubbard, and H P Graf. Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine*, pages 41–46, Nov 1989.

[6] G Cybenko. Approximation by superpositions of sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4), 1989

[7] A D. Deev. Representation of statistics of discriminant analysis and asymptotic expansions in dimensionalities comparable with sample size *Report of Academy of Sciences of the USSR*, 195(No 4):756–762, 1970. (in Russian).

[8] D A Devijver and J. Kitter. *Pattern Recognition: A Statistical Approach* Wiley, N Y , 1983.

[9] R.P.W Duin *On the Accuracy of Statistical Pattern Recognizers* Dutch Efficiency Bureau, Pijnacker, The Netherlands, 1978

[10] S Dutta and S Shekhar. Bond rating: A non-conservative application of neural networks. In *Proc. 1988 IEEE Int. Conf on Neural Networks*, pages II443–II450, San Diego, California, July 1988.

[11] R.A Fisher. The use of multiple measurements in taxonomic problems. *Ann of Eugenics*, 7(No. 2):179–188, 1936

[12] D H. Foley Considerations of sample and feature size *IEEE Trans Inf Theory*, IT-18(No 5):618–626, 1972

[13] K. Fukunaga. Statistical Pattern Recognition In T.Y. Young and K S. Fu, editors, *Handbook of Pattern Recognition and Image Processing*, pages 3–32. New York: Academic, 1986.

[14] P Gallinari, S. Thiria, and F. Fogllman Soulie. Multilayer perceptrons and data analysis. In *Proc. 1988 IEEE Int Conf on Neural Networks*, pages I391–I399, San Diego, California, July 1988

[15] D.J Hand. Recent advances in error rate estimation *Pattern Recognition Letters*, 5:335–346, 1986.

[16] J Higashino, B.L. Degreefand, and E H J. Person. Numerical analysis and adaptation method for learning rate of back propagation. In *Proceedings of IEEE IJCNN'90*, pages I627–I630. Lawrence Erlbaum Associates, Inc., 1990

[17] W M Huang and R P. Lippmann. Neural net and traditional classifiers In D Anderson, editor, *Neural Info Processing Syst* , pages 387–396. American Institute of Physics, New York, 1988

[18] A G Ivakhanenko *Perceptrons* Kijev, Naukova Dumka, 1975.

[19] A K Jain and B. Chandrasekaran Dimensionality and sample size considerations in pattern recognition practice. In P R Krishnaiah and L.N. Kanal, editors, *Handbook of Statistics 2*, pages 835–855 North Holland, 1982

[20] A K Jain, R.C. Dubes, and C C. Chen Bootstrap techniques for error estimation *IEEE Trans Pattern Anal Machine Intell* , PAMI-9(9):628–630, 1987

[21] A K. Jain and M.D Ramaswami Classifier design with Parzen windows. In E.S. Gelsema and L N. Kanal, editors, *Pattern Recognition and Artificial Intellegence*, pages 211–228. Elsevier, 1988.

[22] A K Jain and W G Waller On the optimal number of features in the classification of multivariate Gaussian data *Pattern Recognition*, 10:365–374, 1978.

[23] K. Juskevicius Investigation of the sensitivity of a minimum distance piecewise linear classifier to the limitation of learning sample size. In *Statistical Problems of Control, Issue 61*, pages 89–129. Inst. of Math. and Cyb Press, Vilnius, 1983.

[24] L Kanal and B Chandrasekaran On dimensionality and sample size in statistical pattern recognition. *Pattern Recognition*, 3:238–255, 1971.

[25] A Khotanzad and J. H Lu Distortion invariant character recognition by a multilayer perceptron and back-propagation learning In *Proc. 1988 IEEE Int Conf on Neural Networks*, pages I625–I632, San Diego, California, July 1988.

[26] J.S. Koford and G.F Groner The use of an adaptive threshold element to design a linear optimal pattern classifier. *IEEE Trans. Inf Theory*, IT-12:42–50, 1966.

[27] T. Kohonen, G. Barna, and R Chrisley Statistical pattern recognition with neural networks: Benchmarking studies In *Proc 1988 IEEE Int Conf on Neural Networks*, pages I61–I68, San Diego, California, July 1988.

[28] S Y Kung and J.N Hwang. An algebraic projection analysis for optimal hidden units size and learning rates in back propagation learning In *Proc 1988 IEEE Int Conf on Neural Networks*, pages I363–I370, San Diego, California, July 1988

[29] G.S Lbov On representativeness of the sample size while choosing the effective measurement system In N.G Zagoraiko, editor, *Computing Systems Issue 22*, pages 39–58 Inst of Math Press, Novosibirsk, 1966

[30] R P Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, 4 (2):4–22, Apr 1988.

[31] R. P Lippmann Pattern classification using neural networks *IEEE Communications Magazine*, pages 47–64, Nov. 1989

[32] M. Minsky and S Papert *Perceptrons: An introduction to computational geometry* Cambridge, MA:MIT Press, 1969.

[33] J R Movellan Error functions to improve noise resistance and generalization in back propagation networks In *Proceedings of IEEE IJCNN'90*, pages I557–I560 Lawrence Erlbaum Associates, Inc , 1990

[34] Y Nisikawa, H. Kita, and A. Kawamura NN/I: A neural network which divides and learns environments In *Proceedings of IEEE IJCNN'90*, pages I684–I687 Lawrence Erlbaum Associates, Inc , 1990.

[35] V. Pikelis. Comparison of methods of computing the expected classification errors *Automation and Remote Control*, (No.5):59–63, 1976. (in Russian).

[36] V. Pivoriunas and S. Raudys. On the accuracy of "learning-one-out estimate". In *Statistical Problems of Control, Issue 27*, pages 53–70. Inst of Math. and Cyb Press, Vilnius, 1978. (in Russian)

[37] S. Raudys On the problems of sample size in pattern recognition In *Proc 2nd All-Union Conf on Statistical Methods in control theory*, pages 64–67, Moscow: Nauka, 1970 (in Russian)

[38] S. Raudys. *Statistical Classification in the Case of Essentially Limited Sample Size* PhD thesis, Inst of Math. and Cyb.,Lithuanian Acad of Sci , Vilnius, 1977. (in Russian).

[39] S Raudys. Comparison of the estimates of the probability of misclassification In *Proc 4th Int Joint Conf Pattern Recognition*, pages 280–282, Kyoto, Nov. 1978.

[40] S. Raudys Determination of optimal dimensionality in statistical pattern recognition *Pattern Recognition*, 11:263–270, 1979

[41] S Raudys. Influence of sample size on the accuracy of model selection in pattern recognition. In *Statistical Problems of Control, Issue 50*, pages 9–30 Inst of Math and Cyb Press, Vilnius, 1981 (in Russian)

[42] S Raudys and A K Jain Small sample size effects in statistical pattern recognition: Recommendations for practitioners. *To appear in IEEE Trans. on Pattern Anal Machine Intell*, 1991.

[43] S Raudys and V Pikelis. On dimensionality, sample size, classification error and complexity of classification algorithm in pattern recognition *IEEE Trans. Pattern Anal Machine Intell.*, PAMI-2(3):242–252, 1980

[44] S. Raudys and V. Pikelis Collective selection of the best version of a pattern recognition system. *Pattern Recognition Letters*, 1:7–13, 1982.

[45] S Raudys, V Pikelis, and D. Stasaitis An influence of the number of initial and final features, and correlation between them on the accuracy of feature selection In *Statistical Problems of Control, Issue 74* Inst of Math. and Cyb. Press, Vilnius, 1986. (in Russian)

[46] R Rosenblatt *Principles of Neurodynamics* New York, Spartan Books, 1959.

[47] D E Rumelhart, G E Hinton, and R. J. Williams Learning internal representations by error propagation. In D E. Rumelhart and J.L McClelland, editors, *Parallel Distributed Processing* Cambridge, MA:MIT Press, 1986.

[48] A G Senin. Analysis of optimality criteria used in pattern recognition *Avtometrika No 5, Novosibirsk*, pages 20–25, 1971

[49] F W Smith Small-sample optimality of design techniques for linear classifiers of Gaussian patterns *IEEE Trans Inf Theory*, IT-18(No 1):118–126, 1972.

[50] Y Z Tsypkin. Adaptation, learning and self-learning in automatic systems *Automatic and Remote Control(USSR J)*, (No.1):23–61, 1966

[51] V.N Vapnik. *Estimation of Dependences based on Empirical Data* Nauka Moscow, 1979, Springer NY, 1982

[52] B. Widrow and R.C Winter Neural nets for adaptive filtering and adaptive pattern recognition. *IEEE Trans. Comput*, pages 25–39, March 1988.

[53] A Wieland and R Leighton Geometric analysis of neural network capabilities In *IEEE 1st Int'l Conf on Neural Networks*, pages III–385, June 1987

Artificial Neural Networks and Statistical Pattern Recognition
Old and New Connections
I.K. Sethi and A K Jain (Editors)
© 1991 Elsevier Science Publishers B V All rights reserved

On Tree Structured Classifiers

Saul B. Gelfand and Edward J. Delp

Computer Vision and Image Processing Laboratory, School of Electrical Engineering, Purdue University, West Lafayette, Indiana 47907

Abstract

Classification trees constitute an important and increasingly popular form of hierarchical classifiers. We first describe and compare classification trees and feedforward neural network classifiers. We then focus on the specific problem of obtaining right-sized trees, i.e., trees which neither underfit nor overfit the data. A new efficient iterative method is proposed to grow and prune classification trees. This method divides the data sample into two subsets and iteratively grows a tree with one subset and prunes it with the other subset, successively interchanging the roles of the two subsets. Numerical results on a waveform recognition problem are presented.

1. INTRODUCTION

The goal of this paper is to examine certain problems with some well-known methods of constructing classification trees, and to suggest some new methods which overcome these problems. Although the topic under consideration here is really classification trees, it is useful to compare classification trees and feedforward neural network classifiers (some work has already been done along these lines; see [1], [15]). Both of these methodologies are currently being applied to complex pattern recognition problems and both are active areas of research. A comparison of classification trees and neural network classifiers suggests how it might be possible to combine the two methodologies in such a way that they complement each others capabilities. The classification tree construction methods described in this paper can be applied to the design of classification trees in general, and to the design of classification trees which are used in combination with neural networks in particular. In further work, we explore the application of these tree construction algorithms to classification trees which employ neural network feature extraction (see [7]).

This research was partially supported by the Whirlpool Foundation, and by the National Science Foundation through grant No CDR-8803017 to the Engineering Research Center for Intelligent Manufacturing Systems

The pattern recognition problem we consider in this paper is in the statistical framework, Hence we are given a data set which consists of a random sample of feature vectors and their corresponding class labels. These features were presumably identified based on problem-specific knowledge and exploratory data analysis. The problem is to design a classifier which has appropriate predictive or generalization capability, i.e, which can accurately classify samples not amongst the training data. Furthermore, the problem is completely nonparametric as we make no assumptions concerning the nature of the underlying class distributions.

A typical binary decision tree for classification is shown in Figure 1. The circular nodes are binary decision nodes whose two descendents are determined by a threshold τ_k on a specified feature value x_k. The square nodes are terminal nodes and are assigned a class label. Note that the same feature may occur in different parts of the tree associated with a different threshold. Note also that a feature may be a node-dependent function (linear or nonlinear) of the original features; following [4] we refer to such functions as transgenerated features. When an unlabelled feature vector is submitted for classification, the vector is assigned the class label of the terminal node it lands in. The classification capability of a tree classifier arises from its ability to partition the feature space into complex regions by making a sequence of simple decisions at each of its nodes.

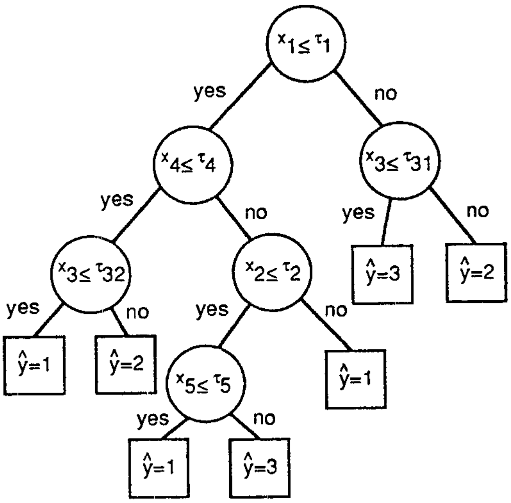
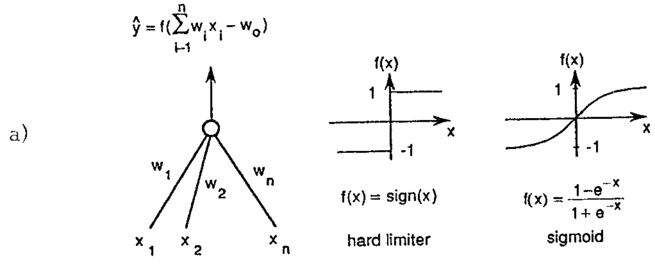


Figure 1. A classification tree for a 5 dimensional feature space and 3 classes.

There have been many approaches to constructing classification trees [2],[3],[4],[6],[8],[11],[12],[14],[16],[17],[18]. Classification trees are usually constructed top-down, i.e., an appropriate feature and threshold are first selected at the root node, and then the data set is propagated down through the root node and appropriate features and thresholds are selected at the nodes at the second level of the tree, and so on. Classification tree construction involves three steps: splitting nodes (selecting features and thresholds), determining which nodes are terminal nodes, and assigning class labels to terminal nodes. Now it is straightforward to assign class labels to terminal nodes, and it is generally agreed that for a fixed set of features the performance of a classification tree does not vary significantly over a wide range of reasonable splitting criteria (c.f. [2]). Hence there are really two fundamental problems in classification tree design: extracting good transgenerated features and selecting the right-sized tree. With regard to transgenerated features there are certain tradeoffs. An appropriate choice of transgenerated features can lead to simplier trees with lower error rates; however, transgenerated features can also lead to large increases in the computation required to construct the classification tree, and can reduce or eliminate the ability to interpret a split at a node. In previous work, transgenerated features have typically consisted of linear combinations of the original features [2],[4],[16]. It seems reasonable that under certain conditions transgenerated features which consist of nonlinear combinations of the original features might be appropriate. With regard to right-sized tree selection, the issue here is that trees that are too large or too small can overfit or underfit the data, respectively. Early approaches to selecting terminal nodes were based on stopping rules, e.g., under specified conditions the recursive partitioning is simply halted. The difficulty with such approaches is that partitioning is halted too soon at certain nodes and too late at others. More recent approaches to selecting terminal nodes involve growing a large tree with pure terminal nodes (i.e. terminal nodes which contain data samples from only a single class) and selectively pruning it upwards [2].

Some advantages of classification trees are that they have a form which can be compactly stored; they efficiently classify new data; and they demonstrate good generalization capability on a variety of problems. Also, the construction procedure includes a means of selecting the right-sized tree to avoid overfitting or underfitting the data. Some disadvantages of classification trees are that for certain problems the use of only single or linear combinations of features and a stepwise level-by-level construction methodology can be myopic and lead to large trees and/or poor error rates. Also, most approaches to constructing classification trees are nonadaptive, i.e., the entire data set of feature vectors and their class labels must be available before construction begins (but see [3],[18]). It would be desirable to have a method for updating trees as new data becomes available, possibly in real-time.

In contrast to a binary decision tree, a typical multilayer feedforward neural network is shown in Figure 2. The network consists of interconnected identical simple processing units called neurons (we do not call them nodes here to avoid confusion with nodes of a tree). An individual neuron sums its weighted inputs and passes the result through a threshold unit. In a multilayer feedforward network the neurons are organized into layers with no feedback or lateral connections. Layers of neurons which are not in the output layer are called hidden layers. The feature values x_k are the inputs to the network, and the weights and thresholds w_i are the parameters which are adjusted via training. When an unlabelled feature vector is submitted for classification, the vector is assigned the class label corresponding to the largest output value. The classification capability of a feedforward net arises from its ability to implement complex mappings due to its multilayer structure and the use of nonlinear threshold functions (like the sigmodal nonlinearity).



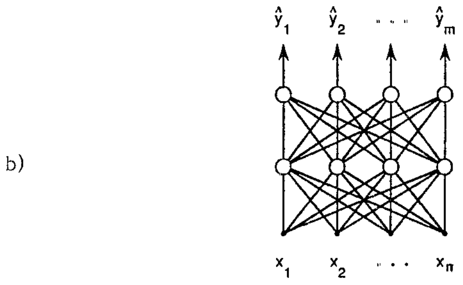


Figure 2 (a) A single neuron and some threshold functions; (b) a two-layer feed forward neural network.

The most popular approach to multilayer feedforward network classifiers is to fix the size of the network (i.e, the number of hidden layers and the number of nodes per layer), and to train the network using the so-called backpropagation algorithm [10], [13]. Backpropagation is a (stochastic) gradient algorithm which recursively updates the weights and thresholds so as to minimize the mean square error between the true and desired network outputs. There are two fundamental problems in the design of feedforward net classifiers: finding efficient and reliable training algorithms, and selecting the right-sized net. With regard to training algorithms, iterative algorithms related to backpropagation have been proposed to reduce the training time at the cost of increased complexity. There has also been research into noniterative algorithms which transform the network structure and/or the input data. With regard to right-sized net selection, the issue is that nets that are too large or too small can overfit or underfit the data, respectively. There has been alot of research into the capabilities of feedforward nets as they depend on the network size, but heuristic approaches are usually used to select a net for a particular problem [9].

Some advantages of feedforward neural network classifiers are that, like classification trees, they have small memory requirements; they also efficiently classify new data and demonstrate good generalization capability. Also, feedforward nets can be trained by iterative algorithms like backpropagation which are adaptive, i.e., each feature vector and class label is successively used to updated weights and thresholds in the net. Some disadvantages are the extremely long training times and the possibility of trapping in local minima. Also, no systematic procedure exists for selecting the size of the net. Furthermore, these problems become more severe for large complex problems.

In view of the characteristics of tree and feedforward neural network classifiers as described above, we suggest that it might be possible to combine the two methodologies in such a way as to improve the performance obtainable by either alone. Now from a certain point of view, the advantages and disadvantages of the tree and feedforward net classifiers can be associated with either too much structure (in the case of the tree) or too little structure (in the case of the net). If the tree and the net could be combined into a classifier which had some intermediate degree of structure, then perhaps only the advantages of the two approaches might be retained. One way to do this is to use a classification tree which has a small multilayer perceptron at each node to extract a nonlinear transgenerated feature. These small feedforward nets would be of fixed size with one hidden layer and a single output (corresponding to only a binary decision at each node). By using such nets the problems of training time, local minima, and selecting the size of the net might be controlled. At the same time, the nets can generate rich nonlinear transgenerated features potentially leading to smaller trees and better classification performance. Also, it might be possible to develop an adaptive version of such a classification tree, since some version of the back propagation algorithm could be used to train the nets at the nodes of the tree.

The rest of this paper develops some new methods for growing and pruning classification trees. The methods described here are useful regardless of what type of transgenerated features are used (including nonlinear features generated by feedforward nets at the nodes of the tree). However, here we only consider splits on single feature coordinates. In [7] we have extended these methods to incorporate transgenerated neural network feature extraction, along the lines described above; we remark that the results in terms of decreased tree size and improved classification performance are very encouraging.

The starting point for our work are two related methods for growing and pruning (and also estimating the error rate) of classification trees from [2]. In the first method the data set is divided into independent training and test sets, and a large tree with pure terminal nodes is grown based on the training set. Then a pruned subtree is selected by minimizing an estimate of the misclassification rate based on the test set over a parametric family of pruned subtrees. The test set is also used to estimate the misclassification rate of the selected subtree. This method is not desirable for small data sets because it only uses part of the data to grow the tree and part of the data to prune it. In the second method a large tree with pure terminal nodes is grown based on the entire data set. Then a pruned subtree is selected by minimizing a cross validation estimate of the misclassification rate over a parametric family of pruned subtrees. These methods are based on an efficient pruning algorithm which generates the parametric family of pruned subtrees. These methods have been incorporated into a program known as CART (Classification And Regression Trees) which has achieved wide-spread popularity.

There are some significant problems with the CART methods of growing and pruning classification trees. The most serious problem with the CART methods is that a pruned subtree is selected by minimizing over a parametric family of pruned subtrees, and this parametric family may not include the optimal (or even a good) pruned subtree. For reasons which will become clear in the sequel, we believe this is likely to be the case for difficult classification problems which require large trees to approximate complex decision regions. A further problem with the CART cross validation method is that it can be very expensive computationally as it requires the growing and pruning of auxiliary trees. Based on these problems we propose two new methods for growing and pruning classification trees.

In the first proposed method the data set is divided into independent training and test sets, and a large tree with pure terminal nodes is grown based on the training set. Then a pruned subtree is selected by minimizing an estimate of the misclassification rate based on the test set over *all* pruned subtrees. The test set is also used to estimate the misclassification rate of the selected subtree. Again, this method is not desirable for small data sets because it only uses part of the data to grow the tree and part of the data to prune it. In the second proposed method, the first method is iterated, alternately using the test (training) set to grow a tree off of the terminal nodes of the previously selected pruned subtree, and the training (test) set to select a new pruned subtree. It can be shown that this iterative method converges. These methods are based on an efficient pruning algorithm which generates a particular pruned subtree.

Because the proposed methods select a pruned subtree by minimizing over *all* pruned subtrees, as opposed to the CART methods which select a pruned subtree by only minimizing over a parametric family of pruned subtrees, we expect them to perform better, i.e., have a lower misclassification rate. In addition, examination of the various methods suggests that the proposed methods should require much less computation than the CART methods. We try the various methods on a waveform recognition problem from [2], which is a difficult problem for classification trees. The results show that for this problem the proposed methods do infact perform better and require less computation than CART.

The paper is organized as follows. In Section 2 we develop notation for decision rules and classification trees. In Section 3 we review the CART methodologies for growing and pruning classification trees, propose new methodologies for growing and pruning classification trees, and compare them. In Section 4 we examine and compare the optimal pruning algorithms which are used in CART and our proposed methods. In Section 5 we give numerical results for the various methods on a waveform recognition problem.

2. DECISION RULES AND CLASSIFICATION TREES

Let $(\underline{X},Y)$ be jointly distributed random variables with $\underline{X}$ taking values in $\mathbb{R}^q$ and Y taking values in the integers $\{1,\ldots,J\}$. $\underline{X}$ is a *pattern* or *feature vector* and the components of $\underline{X}$ are *features,* and Y is the associated *class label.* The problem is to estimate Y based on observing $\underline{X}$. In order to do this we shall assume that a random sample $L^{(0)} = \{(\underline{X}_n, Y_n),\ n = 1,\ldots,N^{(0)}\}$ of feature vectors and their associated class labels are available. Here the $(\underline{X}_n, Y_n)$'s are independent identically distributed random variables, independent of and distributed like $(\underline{X},Y)$. $L^{(0)}$ will be called the *data set.* Hence the problem is to estimate Y based on observing $\underline{X}$, given knowledge of $L^{(0)}$.

A *decision rule* is a function d(•) which maps $\mathbb{R}^q$ into $\{1,\ldots,J\}$. When the feature vector $\underline{X}$ is observed the estimated class is $d(\underline{X})$. Suppose a decision rule d is to be constructed based on $L^{(1)}\subset L^{(0)}$, and to be evaluated based on $L^{(2)}\subset L^{(0)}$. In this case $L^{(1)}$ and $L^{(2)}$ will be called the *training* and *test* sets, respectively. The *true misclassification rate* of d is

$$R^*(d) = P(d(\underline{X}) \neq Y)$$

Let $L\subset L^{(0)}$. The *estimated misclassification rate* of d based on L is

$$R(d) = \frac{M}{N},$$

where M is the number of samples in L such that $d(\underline{X}_n) \neq Y_n$, and N is the total number of samples in L. When $L = L^{(1)}$, R(d) is a *training sample estimate* of $R^*(d)$, and when $L = L^{(2)}$, R(d) is a *test sample estimate* of $R^*(d)$. Let R_B denote the (minimum) Bayes misclassification rate.

We briefly describe some necessary terminology for discussing trees (see [2] for more detail). A *tree* is a finite nonempty set T of positive integers and two functions left (•) and right (•) from T to $T\cup\{0\}$ such that

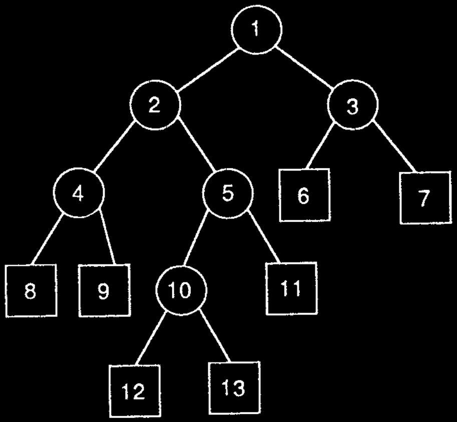
(i) For each $t\in T$ either left (t)=0 and right (t)=0, or left (t) > t and right (t) > t

(ii) For each $t\in T$, other than the smallest integer in T, there is a unique $s\in T$ such that either t=left (s) or t=right (s)

T will itself be called a *tree,* and each element of T is a *node.* Figure 3 shows a tree and the corresponding values of $\ell(t)$=left (t) and r(t)=right (t).

Let T_1 be a non empty subset of T and let $\text{left}_1(\bullet)$ and $\text{right}_1(\bullet)$ be the restriction of left(•) and right(•) to T_1, respectively. T_1 is a *subtree* of T if the triple T_1, $\text{left}_1(\bullet)$, $\text{right}_1(\bullet)$, forms a tree. T_1 is a *pruned subtree* of T if T_1 is a subtree of T with the same root node as T; this is denoted by $T_1 \leq T$ or $T \geq T_1$.

We now show how to associate a decision rule with a tree. Let T be a tree, and suppose that $U(t)\subset\mathbb{R}^q$ and $j(t)\in\{1,\ldots,J\}$ for $t\in\tilde{T}$. Furthermore suppose that $\{U(t),\ t\in\tilde{T}\}$ is a partition of $\mathbb{R}^q$. A *classification tree* consists of the tree T together



t	$\ell(t)$	r(t)
1	2	3
2	4	5
3	6	7
4	8	9
5	10	11
6	0	0
7	0	0
8	0	0
9	0	0
10	12	13
11	0	0
12	0	0
13	0	0

Figure 3 A tree and the corresponding values of $\ell(t)$ = left(t) and r(t) = right(t)

with the partition $\{U(t):t\in\tilde{T}\}$ and class labels $\{j(t):t\in\tilde{T}\}$. T will itself be called a classification tree. The decision rule d corresponding to the classification tree T is given by

$$d(\underline{x}) = j(t) \quad \text{if} \quad \underline{x}\in U(t) .$$

More generally, we can associate a collection of decision rules with the pruned subtrees of a tree. Let T be a tree and suppose that $U(t)\subset\mathbb{R}^q$ and $j(t)\in\{1,\ldots,J\}$ for $t\in T$. Furthermore suppose that for every pruned subtree $T_1 \leq T$, $\{U(t), t\in\tilde{T}_1\}$ is a partition of $\mathbb{R}^q$. Then the decision rule d_1 corresponding to the classification tree T_1 is given by

$$d_1(\underline{x}) = j(t) \quad \text{if} \quad \underline{x}\in U(t),\ t\in\tilde{T}_1 .$$

We denote the *true misclassification rate* of a classification tree T by $R^*(T)$, and the *estimated misclassification rate* based on $\underline{L}$ by R(T). An important fact is that R(T) can be expressed as an additive function on $\tilde{T}$, i.e.,

$$R(T) = \sum_{t \in \tilde{T}} R(t) , \quad R(t) = \frac{M(t)}{N} , \tag{2.1}$$

where M(t) is the number of samples in L such that $\underline{X}_n \in U(t)$ and $Y_n \neq j(t)$, and N is the total number of samples in L. We will also consider a risk function which penalizes the complexity of the classification tree as well as its misclassification rate. The *estimated complexity-misclassification rate* of a classification tree T based on L is defined as

$$R_\alpha(T) = R(T) + \alpha |\tilde{T}| .$$

Here α is a constant which weights the number of terminal nodes in T, which is a measure of the complexity of T. Note that $R_\alpha(T)$ can also be expressed as an additive function on $\tilde{T}$, i.e.,

$$R_\alpha(T) = \sum_{t \in \tilde{T}} R_\alpha(t) , \quad R_\alpha(t) = R(t) + \alpha . \tag{2.2}$$

Let T and its pruned subtrees be classification trees as described above. Consider the problem of minimizing the estimated misclassification rate, or more generally the estimated complexity-misclassification rate, based on L, over the pruned subtrees of T, i.e. finding a $T_1 \leq T$ such that

$$R_\alpha(T_1) = \min_{T' \leq T} R_\alpha(T')$$

Since the pruned subtrees of T are a finite set, the minimum exists but may not be unique. Call any such minimizing T_1 an *optimally pruned subtree* of T with respect to R_α. In [2, p. 285] it is shown that there exists an optimally pruned subtree $T_1 \leq T$ such that $T_1 \leq T'$ for any other optimally pruned subtree $T' \leq T$. Call this T_1 the (unique) *smallest optimally pruned subtree* of T with respect to R_α, and denote it by $T(\alpha)$. We remark that $T(\alpha)$ will exist for any tree T and any cost function $R_\alpha(T)$ which can be expressed as an additive function on $\tilde{T}$ as in (2.2); there is nothing special about classification trees and misclassification rates in this regard.

3. CLASSIFICATION TREE CONSTRUCTION AND ERROR RATE ESTIMATION

The basic idea behind most algorithms for generating classification trees is to recursively partition the feature space in such a way as to recursively generate the tree. If t is a nonterminal node, let t_L, t_R be its immediate descendents. At each nonterminal node t the region $U(t) \subset \mathbb{R}^q$ is split into two regions $U(t_L)$ and $U(t_R)$ which correspond to the nodes t_L and t_R, respectively. Splitting continues in this way until some stopping criterion is met and the node becomes a terminal node. At each terminal node t a class label j(t) is assigned. This procedure yields a classification tree as defined in Section 2. In fact, if all nodes (terminal and nonterminal alike) are assigned a class label then the procedure yields a tree such that every pruned subtree is a classification tree, again as defined in Section 2.

The construction of a classification tree requires three steps:

1. The selection of splits (features and thresholds).
2. The decision as to which nodes are terminal nodes.
3. The assignment of class labels.

We assign class labels and select splits as described in [2]. Briefly, we assign a class label $j(t)$ to each node $t \in T$ by minimizing the estimated misclassification rate at node t. We select a split $s(t)$ at each nonterminal node $t \in T - \tilde{T}$ by minimizing the resulting change in node impurity at nodes t_L and t_R relative to node t; we use the Gini criterion as our node impurity measure. Our splits consist of thresholds on the individual features $s = \{\underline{x}: x_k \leq \tau\}$ where τ ranges over the real numbers and $k = 1, \ldots, q$. The effect of using such splits is to partition the feature space with hyperplanes orthogonal to the component axes. In other words, we do not consider transgenerated features here. See [2] for a detailed discussion of the Gini and other impurity functions. Also, see [4],[8],[12],[14] for other approaches to splitting.

We now consider the problem of deciding whether a node should be terminal or nonterminal node, and how to estimate the misclassification rate. Suppose we construct the classification tree T based on a data subset $L \subset L^{(0)}$ using the splitting and labelling methods described above, and we continue the splitting until every node has pure class membership (assume this can be done for the moment). Then T correctly classifies every sample in L and $R(T) = 0$. But if class distributions overlap T should not correctly classify every sample in L and $R^*(T) \geq R_B > 0$. It is seen that relatively pure terminal nodes will lead to large trees which overfit the data, while relatively impure terminal nodes will lead to small trees which underfit the data; the problem is to select a right-sized tree and to obtain honest estimates of its misclassification rate. Early approaches to selecting terminal nodes were based on stopping rules, In [2] it is suggested that instead of using stopping rules, it is better to grow a large tree with terminal nodes which have pure (or nearly pure) class membership, and selectively prune it upwards.

Next, we describe four methods for growing, pruning and estimating the error rate of classification trees. These descriptions formalize the discussion in the Introduction (Section 1) We first review the two CART methods from [2]. Based on certain problems with these methods, we propose two new methods. Let L be a subset of the data set $L^{(0)}$. Let N(t) be the number of samples in L which land in node t, and let $N_j(t)$ be the number of samples in L which land in node t and belong to class j (also recall the definitions of R(t) and R(T) from Section 2). In the sequel, superscripts on quantities like $N_j^{(i)}(t)$, $N^{(i)}(t)$, $R^{(i)}(t)$, and $R^{(i)}(T)$ mean that the quantities are based on some data subset $L^{(i)} \subset L^{(0)}$ instead of $L \subset L^{(0)}$. Such notation is necessary when we discuss algorithms which operate on multiple data subsets $L^{(i)} \subset L^{(0)}$, e.g., when $L^{(1)}$ is a training set and $L^{(2)}$ is a test set, or when $L^{(1)}, \ldots, L^{(V)}$ are subsets of $L^{(0)}$ which are used in a V-fold cross validation procedure.

Method I: CART Independent Training and Test Set Method [2]

1) Divide $L^{(0)}$ into $L^{(1)}$ and $L^{(2)}$ such that $L^{(1)}$ and $L^{(2)}$ have approximately equal numbers of samples in each of the classes.

2) Use $L^{(1)}$ to generate a tree T by splitting until all terminal nodes $t \in \tilde{T}$ have $N_j^{(1)}(t) = N^{(1)}(t)$ for some j, or $N^{(1)}(t) \leq N_{min}$, or $\min[N^{(1)}(t_L), N^{(1)}(t_R)] = 0$ for all possible splits of t, and assign class labels to all nodes.
Let $T(\alpha)$ be the smallest optimally pruned subtree of T with respect to $R_\alpha^{(1)}$.

3) Generate a nested sequence of pruned subtrees $T = T_0 \geq T_1 \geq \ldots \geq T_K = \text{root}(T)$ of T such that $T(\alpha) = T_0 = T$ for $\alpha < \alpha_1$, $T(\alpha) = T_k$ for $\alpha_k \leq \alpha < \alpha_{k+1}$ and $k = 1,\ldots,K-1$, and $T(\alpha) = T_K = \text{root}(T)$ for $\alpha \geq \alpha_K$, for some numbers $-\infty < \alpha_1 < \alpha_2 < \ldots < \alpha_K < \infty$ (CART contains a pruning algorithm for computing the α_k's and the T_k's; see Section 4.)

4) Select the smallest $T^* \in \{T_0,\ldots,T_K\}$ such that

$$R^{(2)}(T^*) = \min_k R^{(2)}(T_k)$$

5) Estimate the misclassification rate of T* by

$$\hat{R}(T^*) = R^{(2)}(T^*)$$

□

Method I only uses part of the data to grow the large tree T and part of the data to prune it. It is desirable, especially for small data sets, to use all of the data to grow the large tree T and all of the data to prune it. The following cross validation method provides an alternative.

Method II: CART Cross Validation Method [2]

1) Divide $L^{(0)}$ into $L_1,\ldots,L_V$ such that $L_1,\ldots,L_V$ have approximately equal numbers of samples in each of the classes. Let $L^{(v)} = L - L_v$ for $v = 1,\ldots,V$ (typically $V = 10$).

2) Use $L^{(0)}$ to generate a tree T by splitting until all terminal nodes $t \in \tilde{T}$ have $N_j^{(0)}(t) = N^{(0)}(t)$ for some j, or $N^{(0)}(t) \leq N_{min}$, or $\min[N^{(0)}(t_L), N^{(0)}(t_R)] = 0$ for all possible splits of t, and assign class labels to all nodes.

Let $T(\alpha)$ be the smallest optimally pruned subtree of T with respect to $R_\alpha^{(0)}$.

3) Generate a nested sequence of pruned subtrees $T = T_0 \geq T_1 \geq \ldots \geq T_K = \text{root}(T)$ of T such that $T(\alpha) = T_0$ for $\alpha < \alpha_1$, $T(\alpha) = T_k$ for $\alpha_k \leq \alpha < \alpha_{k+1}$ and $k = 1,\ldots K-1$, and $T(\alpha) = T_K = \text{root}(T)$ for $\alpha \geq \alpha_K$, for some numbers $-\infty < \alpha_1 < \alpha_2 < \ldots < \alpha_K < \infty$ (Use CART pruning algorithm; see Section 4)

4) Use $L^{(v)}$ to generate a tree $T^{(v)}$ by splitting until all terminal nodes $t \in \tilde{T}^{(v)}$ have $N_j^{(v)}(t) = N^{(v)}(t)$ for some j, or $N^{(v)}(t) \leq N_{min}$, or $\min[N^{(v)}(t_L), N^{(v)}(t_R)] = 0$ for all possible splits of t, and assign class labels to all nodes, for $v = 1,\ldots,V$.

Let $T^{(v)}(\alpha)$ be the smallest optimally pruned subtree of $T^{(v)}$ with respect to $R_\alpha^{(v)}$.

5) Generate the parametric family of pruned subtrees $T^{(v)}(\alpha)$, $-\infty < \alpha < \infty$, of $T^{(v)}$, for $v = 1,\ldots,V$ (Use CART pruning algorithm; see Section 4)
Let

$$R^{cv}(T_k) = \frac{1}{V} \sum_{v=1}^{V} R_v(T^{(v)}(\sqrt{\alpha_k \alpha_{k+1}})), \quad k = 1,\ldots,K-1$$

(R_v is the estimate of the misclassification rate based on $\mathcal{L}_v$, and R^{cv} is a cross validation estimate of the misclassification rate)

6) Select the smallest $T^* \in \{T_0,\ldots,T_K\}$ such that

$$R^{cv}(T^*) = \min_k R^{cv}(T_k)$$

7) Estimate the misclassification rate of T^* by

$$\hat{R}(T^*) = R^{cv}(T^*)$$

□

As pointed out in the Introduction there are some significant problems with the CART methods. First, from our point of view there is no real justification that the parametric family of pruned subtrees $\{T(\alpha) : -\infty < \alpha < \infty\}$ is the right set of pruned subtrees to select the final tree T^* from. This problem affects both CART methods. Second, although it is preferable to use Method II over Method I since it uses all the data to grow the large tree T and all the data to select the final tree T^*, Method II can be very expensive computationally as it requires growing the large auxiliary trees $T^{(v)}$ and generating a parametric family of pruned subtrees $\{T^v(\alpha) : -\infty < \alpha < \infty\}$ for each one. Based on these problems with the CART methods, we now propose two new methods for growing, pruning, and estimating the misclassification rate of classification trees.

Method III: Proposed Independent Training and Test Set Method

1) Divide $\mathcal{L}^{(0)}$ into $\mathcal{L}^{(1)}$ and $\mathcal{L}^{(2)}$ such that $\mathcal{L}^{(1)}$ and $\mathcal{L}^{(2)}$ have approximately equal numbers of samples in each class.
2) Use $\mathcal{L}^{(1)}$ to generate a tree T by splitting until all terminal nodes $t \in \tilde{T}$ have $N_j^{(1)}(t) = N^{(1)}(t)$ for some j, or $N^{(1)}(t) \le N_{min}$, or $\min[N^{(1)}(t_L), N^{(1)}(t_R)] = 0$ for all possible splits of t, and assign class labels to all nodes
3) Select the smallest pruned subtree T^* of T such that

$$R^{(2)}(T^*) = \min_{T' \le T} R^{(2)}(T')$$

(We shall give an efficient pruning algorithm for computing T^*; see Section 4)

4) Estimate the misclassification rate of T^* by

$$\hat{R}(T^*) = R^{(2)}(T^*)$$

□

Method IV: Proposed Iterative Method

1) Divide $L^{(0)}$ into $L^{(1)}$ and $L^{(2)}$ such that $L^{(1)}$ and $L^{(2)}$ have approximately equal numbers of samples in each class.
2) Use $L^{(1)}$ to generate a tree T_0 by splitting until all terminal nodes $t \in \tilde{T}$ have $N_j^{(1)}(t) = N^{(1)}(t)$ for some j, or $N^{(1)}(t) \le N_{min}$, or $\min[N^{(1)}(t_L), N^{(1)}(t_R)] = 0$ for all possible splits of t, and assign class labels to all nodes
3) Select the smallest pruned subtree T_0^* of T_0 such that

$$R^{(2)}(T_0^*) = \min_{T' \le T_0} R^{(2)}(T')$$

(Use proposed pruning algorithm in Section 4)
Set k = 1
4) Set i = 1, j = 2 if k is even, and i = 2, j = 1 if k is odd.
5) Use $L^{(i)}$ to generate a tree T_k by splitting the terminal nodes $\tilde{T}_{k-1}^*$ until all terminal nodes $t \in \tilde{T}_k$ have $N_j^{(i)}(t) = N^{(i)}(t)$ for some j, or $N^{(i)}(t) \le N_{min}$, or $\min[N^{(i)}(t_L), N^{(i)}(t_R)] = 0$ for all possible splits of t, and assign class labels to nodes in $T_k - T_{k-1}^*$ (numbers, splits, and class labels of nodes in T_{k-1}^* are unchanged).
6) Select the smallest pruned subtree T_k^* of T_k such that

$$R^{(j)}(T_k^*) = \min_{T' \le T_k} R^{(j)}(T')$$

(Use proposed pruning algorithm in Section 4)
7) If $|T_k^*| = |T_{k-1}^*|$ then set $T^* = T_k^*$; else set k = k + 1 and go to 4)
8) Estimate the misclassification rate of T^* by

$$\hat{R}(T^*) = \sum_{t \in S^{(1)}} R^{(2)}(t) + \sum_{t \in S^{(2)}} R^{(1)}(t)$$

where
$S^{(\ell)} = \{t \in \tilde{T}^*: \text{ t was assigned a class label based on } L^{(\ell)}\}$ □

There is a modification of Step c) in Method IV which can significantly reduce the amount of computation. Specifically, it can be shown that if a node is a terminal node in any two consecutive optimally pruned subtrees, then it is a terminal node in all subsequent optimally pruned subtrees and never has to be split. It can also be thrown that the sequence of pruned subtrees $\{T_k^*\}$ generated by Method IV is nested (i.e. $T_k^* \le T_{k+1}^*$ for all k) and converges (i.e. there exists integer K such that $T_k^* = T_K^*$ for all $k \ge K$). Proofs of those assertions and additional details on the implementation and properties of Method IV can be found in [5].

We now make several remarks about how we perceive Methods III and IV might overcome the problems with Methods I and II discussed above. First, note that in

Method III the pruned subtree T^* of T is selected by minimizing an unbiased independent test set estimate of the misclassification rate over *all* pruned subtrees, whereas in Method I the pruned subtree T^* of T only minimizes this estimate over a parametric family of pruned subtrees, and this parametric family is selected using a biased resubstitution estimate of the misclassification rate. Hence Method III eliminates the problem encountered with Method I as to whether the parametric family of pruned subtrees contains an optimal (or even good) subtree which can be selected. We believe Method IV is superior to Method II by similar reasoning. Second, we believe that Method IV will be much less computationally expensive than Method II in general. By far, the most computationally intensive part of tree construction is the selection of the splits at the nodes of the trees. Furthermore, the amount of computation in selecting a split at a node grows rapidly with the size of the data subset at the node (see [2, pp. 163-167]). The iterative procedure of Method IV generates a nested sequence of optimally pruned subtrees which, as described above, has the property that if a node is terminal node in two consecutive pruned subtrees then it is a terminal node in all subsequent pruned subtrees and never has to be split. Consequently the bulk of the tree is constructed after a few iterations and thereafter relatively few nodes need to be split and these nodes contain relatively few samples. The cross validation procedure of Method II, however, must grow and prune auxiliary large trees from scratch. In Section 5 we compare the various methods on a waveform recognition problem from [2].

4. TREE PRUNING ALGORITHMS

In this section we consider tree pruning algorithms which are needed to implement the methods described in Section 3. We first review an algorithm developed in [2] for generating a parametric family of optimally pruned subtrees. This algorithm is part of the CART methods. We then propose a simple algorithm for generating a particular optimally pruned subtree. This algorithm is used in the proposed methods.

The results in this section actually concern pruning trees which are not necessarily classification trees. Of course, they apply to classification trees as a special case. Let T_0 be a fixed tree. Let $R(t)$, $t \in T_0$, be real numbers, and for each real number α, let $R_\alpha(t) = R(t) + \alpha$ for $t \in T_0$. Given a subtree T of T_0 set

$$R(T) = \sum_{t \in \tilde{T}} R(t)$$

and

$$R_\alpha(T) = \sum_{t \in \tilde{T}} R_\alpha(t) = R(T) + \alpha |\tilde{T}| .$$

Let $T_0(\alpha)$ be the smallest optimally pruned subtree of T_0 with respect to R_α (see Section 2). Of course, $T_0(0)$ is the smallest optimally pruned subtree of T_0 with respect to R. Note that $T_0(\alpha)$, $-\infty < \alpha < \infty$, is required for the CART methods (for appropriate choice of T_0 and R_α), while $T_0(0)$ is required for the proposed methods (for appropriate choice of T_0 and R).

We now give the CART pruning algorithm for generating the family of $T_0(\alpha)$, for $-\infty < \alpha < \infty$, and propose a pruning algorithm for generating a particular $T_0(\alpha)$. Let $T_0 = \{t_1,\ldots,t_L\}$ with $t_1 < \cdots < t_L$. Let $\ell(t) = \text{left}(t)$, $r(t) = \text{right}(t)$, $p(t) = \text{parent}(t)$ for $t \in T_0$. The CART algorithm works with a six-tuple $(\ell(t), r(t), p(t), R(t), S(t), N(t))$ while the proposed algorithm works with a four-tuple $(\ell(t), r(t), R_\alpha(t), S_\alpha(t))$. For the CART algorithm $S(t) = R(T_t)$ and $N(t) = |T_t|$, while for the proposed algorithm $S_\alpha(t) = R_\alpha(T_t)$, where T is the currently considered subtree of T_0 at a particular point in the algorithm.

CART Pruning Algorithm [2]:

```
For $t = t_L,\ldots,t_1$
    {If $t \in \tilde{T}_0$ then
        {$S(t) = R(t)$, $N(t) = 1$}
    If $t \in T_0 - \tilde{T}_0$ then
        {$S(t) = S(\ell(t)) + S(r(t))$, $N(t) = N(\ell(t)) + N(r(t))$}}
$\alpha = \dfrac{R(t_1) - S(t_1)}{N(t_1) - 1}$,  $T = T_0$
$k = 1$
Repeat until $|\tilde{T}| = 1$
    {For $t = t_1,\ldots t_L$
        {If $t \in T - \tilde{T}$ then
            {$\alpha = \min[\alpha, \dfrac{R(t) - S(t)}{N(t) - 1}]$}}
    For $t = t_1,\ldots,t_L$
        {If $t \in T - \tilde{T}$ and $\dfrac{R(t) - S(t)}{N(t) - 1} = \alpha$ then
            {$T = T - (T_{\ell(t)} \cup T_{r(t)})$, $\ell(t) = 0$, $r(t) = 0$
            $S(t) = R(t)$, $N(t) = 1$}
        $s = p(t)$
        Repeat until $s = t_1$
            {$S(s) = S(\ell(s)) + S(r(s))$, $N(s) = N(\ell(s)) + N(r(s))$
            $s = p(s)$}}
    $\alpha_k = \alpha$,  $T_k = T$
    $k = k+1$}
$K = k - 1$
```

$$T_0(\alpha) = \begin{cases} T_0 & \alpha < \alpha_1 \\ T_k & \alpha_k \le \alpha < \alpha_{k+1}, \quad k = 1,\ldots,K-1 \\ T_K & \alpha \ge \alpha_K \end{cases}$$

□

Proposed Pruning Algorithm:

```
$T = T_0$
For $t = t_L, \ldots, t_1$
        {If $t \in T$ then
                {$S_\alpha(t) = R_\alpha(t)$}
        If $t \in T - T$ then
                {$S_\alpha(t) = S_\alpha(\ell(t)) + S_\alpha(r(t))$
                 If $R_\alpha(t) \le S_\alpha(t)$ then
                        {$T = T - (T_{\ell(t)} \cup T_{r(t)})$, $\ell(t) = 0$, $r(t) = 0$
                         $S_\alpha(t) = R_\alpha(t)$}}}
$T_0(\alpha) = T$
```
□

The CART algorithm is essentially a top-down algorithm, in that it starts from the root node and proceeds down the tree, pruning away branches. Each time it prunes away a branch to obtain a terminal node t it suitably modifies the six-tuples $(\ell(s), r(s)p(s), R(s), S(s), N(s))$ corresponding to the ascendents s of t and also t itself. K passes through the tree are required to generate the nested subtrees $T_0 \ge T_1 \ge \cdots \ge T_K$ and hence the family $T_0(\alpha)$ for $-\infty < \alpha < \infty$. In contrast to the CART algorithm, the proposed algorithm is essentially a bottom-up algorithm, in that it starts from the terminal nodes and proceeds up the tree, pruning away branches. Each time it prunes away a branch to obtain a terminal node t it suitably modifies the four-tuple $(\ell(t), r(t), R_\alpha(t), S_\alpha(t))$. Only one pass through the tree is required to generate $T_0(\alpha)$. We remark that the CART algorithm can, of course, be used to generate a particular $T_0(\alpha)$ by generating $T_1, \ldots, T_k$ such that $\alpha_k \le \alpha < \alpha_{k+1}$. This procedure is, however, very inefficient compared to the proposed algorithm.

5. EXPERIMENTAL RESULTS

To illustrate various parts of the methodology in tree structured classification, the waveform recognition problem [2, p. 49] was chosen. It is a three-class problem based on the waveforms $h_1(t)$, $h_2(t)$ and $h_3(t)$ shown in Figure 4.

Each class consists of a random convex combination of two of these waveforms sampled at 21 points with noise added to them. Thus the feature vector is 21 dimensional, $\underline{X} = (X_1, \ldots, X_{21})$. A class was randomly selected with all classes having equal probability. A feature vector for the selected class was then generated by independently generating a uniformly distributed random variable u on the interval [0,1], and 21 normally distributed random variables $\varepsilon_1, \ldots, \varepsilon_{21}$ with zero mean and unit variance, and combining the waveforms as follows.

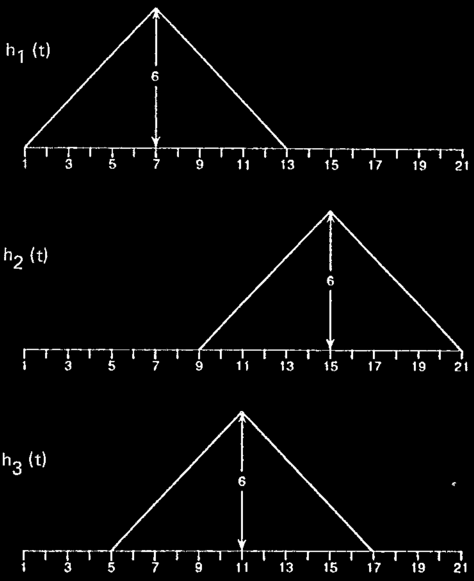


Figure 4 Three waveforms

For class 1,

$X_m = uh_1(m) + (1-u)h_2(m) + \varepsilon_m$; $m = 1,2,\ldots,21$

For class 2,

$X_m = uh_1(m) + (1-u)h_3(m) + \varepsilon_m$; $m = 1,2,\ldots,21$

For class 3,

$X_m = uh_2(m) + (1-u)h_3(m) + \varepsilon_m$; $m = 1,2,\ldots,21$

According to [2], the Bayes misclassification rate for this waveform recognition problem is $R_B \approx .14$.

The data set $L^{(0)}$ consisted of 300 feature vectors and class labels with approximately 100 samples from each class. Trees were grown using splits of the form $s = \{\underline{x}: x_m \leq \tau\}$ for τ ranging over all real numbers and $m = 1,\ldots,21$, and the Gini splitting criterion (see Section 3). Splitting was terminated as soon as a node had pure class membership. Pruning and error rate estimation were performed based on each of the four methods described in Section 3. In Method I (CART Independent Training and Test Set Method), Method III (Proposed Independent Training and Test Set

Method), and Method IV (Proposed Iterative Method), the data set $L^{(0)}$ was divided into the two sets $L^{(1)}$ and $L^{(2)}$ each consisting of 150 total samples with approximately 50 samples from each class. In Method II (CART Cross Validation Method) a 10-fold cross validation was performed where the data set $L^{(0)}$ was divided into 10 sets $L_1, \ldots, L_{10}$ each consisting of 30 total samples with approximately 10 samples from each class. Two independent data sets (each with 300 total samples and approximately 100 samples per class) were generated with different seed values and the results discussed below were obtained by averaging. Method IV required K = 4 growing and pruning iterations to converge on both of the data sets. An additional set of 5000 independent samples was used to obtain highly accurate estimates of the true misclassification rates. The algorithms were run on a dual processor VAX 11/780 running UNIX 4.3 BSD.

In Table 1 we show results for each of the four methods. In each case we show the number of terminal nodes $|\tilde{T}^*|$, the estimated misclassification rate $\hat{R}(T^*)$, the true misclassification rate $R^*(T^*)$ (based on 5000 independent samples), and the required computer time measured in CPU seconds. The results for the CART Methods I and II are consistent with results obtained in [2]. The results show that our proposed Methods III and IV perform better and require less computation than the CART Methods I and II, at least on the waveform recognition problem. It is particularly interesting that the proposed Method III which uses independent training and test sets performs better than the CART Method II which uses cross validation, while requiring only a fraction of the computing time. It seems that selecting the right-sized tree is indeed a critical issue for the waveform recognition problem. We attribute this behavior to the fact that the waveform recognition problem is a difficult problem for tree classifiers which try to approximate the decision regions with hyperplanes orthogonal to the coordinate axes [2].

Table 1
Averaged Results for Waveform Recognition Problem.

Method	$\|\tilde{T}^*\|$	$\hat{R}(T^*)$	$R^*(T^*)$	CPU SECONDS
I	17	.30	.31	90
II	23	.29	.29	700
III	20	.27	.27	70
IV	26	.26	.26	150

6. CONCLUSION

We first described and compared classification trees and feedforward neural network classifiers. We further suggested that it might be possible to combine the methodologies in a useful way; in particular we suggested the idea of using small feedforward nets at the nodes of a classification tree to extract transgenerated features. This idea is developed in [7]. We then focused on the critical issue of obtaining right-sized trees, i.e., trees which neither underfit nor overfit the data. This is an important problem for the design of classification trees in general (whether they use transgenerated features or not). Instead of using stopping rules we followed the philosophy introduced in [2] of growing a large tree with pure (or nearly pure) terminal nodes and selectively pruning it back. New efficient methods were proposed to grow and prune classification trees. The first method divides the data set into two independent subsets, and uses the first subset to grow a large tree and the second subset to select a pruned subtree which minimizes an estimate of the misclassification rate over all possible pruned subtrees. The second method continues this procedure by using the second subset to grow a large tree off of the terminal nodes of the previously selected pruned subtree and the first subset to select a new pruned subtree which minimizes an estimate of the misclassification rate over all possible subtrees, and then iterates this procedure by successively interchanging the roles of the two subsets. The convergence and other properties of the iterative method have been established. Numerical results were given which show that our methods perform better and require less computation than the widely used CART program on a waveform recognition problem.

REFERENCES

[1] L. Atlas, R. Cole, Y. Muthusamy, A. Lippman, J. Conner, D. Park, M. El-Sharkawi and R. J. Marks III, "A performance comparison of trained multi-layer perceptrons and trained classification trees," *Proc. IEEE,* vol. 78, no. 10, pp. 1614-1619, 1990.

[2] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees,* Belmont, CA: Wadsworth Int., 1984.

[3] S. L. Crawford, "Extensions to the CART algorithm," *Int. J. Man-Machine Studies,* vol. 31, pp. 197-217, 1989.

[4] J. H. Friedman, "A recursive partitioning decision rule for nonparametric classification," *IEEE Trans. Comput.,* vol. C-26, pp. 404-408, 1973.

[5] S. B. Gelfand, C. S. Ravishankar, and E. J. Delp, "An iterative growing and pruning algorithm for classification tree design," to appear in *IEEE Trans. Pattern Anal. Machine Intell.*

[6] R. M. Goodman and P. Smyth, "Decision tree design from a communication theory standpoint," *IEEE Trans. Inform. Theory,* vol. 34, pp. 979-994, 1988.

[7] H. Guo and S. B. Gelfand, "Tree classifiers with multilayer perceptron feature extraction," in preparation.

[8] E. G. Henrichon and K. S. Fu, "A nonparametric partitioning procedure for pattern classification," *IEEE Trans Comput.*, vol. C-18, pp. 614-624, 1969.

[9] E. D. Karnin, "A simple procedure for pruning back-propagation trained neural networks," *IEEE Trans. Neural Networks*, vol. 1, no. 2, pp. 239-242, 1990.

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

[11] H. J. Payne and W. S. Meisel, "An algorithm for constructing optimal binary decision trees," *IEEE Trans. Comput.*, vol. C-26, pp. 905-916, 1977.

[12] J. R. Quinlan and R. L. Rivest, "Inferring decision trees using the minimum description length principle," *Information and Computation*, vol. 80, no. 3, pp. 227-248, 1989.

[13] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning internal representations by error propagation," in D. E. Rumelhart and J. L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol 1: Foundations*, MIT Press, 1986.

[14] I. K. Sethi and G. P. R. Sarvarayudu, "Hierarchical classifier design using mutual information," *IEEE Trans. Pattern Anal Machine Intell.*, vol. PAMI-4, pp. 441-445, 1981.

[15] I. K. Sethi, "Entropy nets: from decision trees to neural networks," *Proc. IEEE*, vol. 78, no. 10, pp. 1605-1613.

[16] J. Sklansky, "Locally trained piecewise linear classifiers," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-2, pp. 101-111, 1980.

[17] P. H. Swain and H. Hauska, "The decision tree classifier: design and potential," *IEEE Trans Geosc. Electron*, vol. GE-15, pp. 142-147, 1977.

[18] P. E. Utgoff, "Incremental design of decision trees," *Machine Learning*, vol. 4, no. 2, 1989.

Artificial Neural Networks and Statistical Pattern Recognition
Old and New Connections
I.K. Sethi and A.K. Jain (Editors)
© 1991 Elsevier Science Publishers B.V. All rights reserved

Decision tree performance enhancement using an artificial neural network implementation[1]

Ishwar K. Sethi

Vision and Neural Networks Laboratory, Department of Computer Science, Wayne State University, Detroit, MI 48202, USA

Abstract

Decision tree classifiers represent a popular classification methodology that has been successfully used in many pattern recognition tasks. Noting many similarities between the tree classifiers and the multilayer perceptron classifiers, an artificial neural network based implementation of decision tree classifiers to enhance their performance is described here Several issues related to decision tree classifiers are discussed and it is shown how these can be dealt with following the proposed implementation.

1. INTRODUCTION

Decision tree based classification is a widely used nonparametric method for complex pattern recognition tasks involving several pattern classes and a large number of features. A decision tree classifier utilizes a series of simple decision functions, usually binary in nature, to determine the identity of an unknown pattern. The evaluation of these decision functions is planned in such a way that the outcome of successive decision functions reduces uncertainty about the unknown pattern. Each decision function uses only a subset of features, usually a single feature, as its argument This coupled with the fact that patterns from different classes are identified using different subsets of the decision functions make decision tree based classification computationally very attractive. In some sense then, the decision tree classifier represents an "optimal" way of performing classification by matching the feature subsets with the pattern class subsets at each step of decision making.

A decision tree induces a hierarchical partitioning over the decision space. Starting with the decision function at the root node, successive internal nodes in a decision tree partition their associated decision region into two half spaces with the node decision function defining the dividing boundary. The final decision boundary due to the induced hierarchical partitioning can be very complex depending upon the nature of node decision functions and the tree size. The most common choice for the node decision functions is a threshold comparison on a component of the feature vector which results in a feature space partitioning by hyperplanes parallel or orthogonal to the coordinates axes of the feature space

An important characteristic of the decision tree induced partitioning is that it is autonomously configurable from a collection of labeled pattern vectors for a given classification task independently of any *a priori* information about the functional form of the distribution of pattern vectors in the decision space The procedures to self-configure the hierarchical partitioning of the feature space are generally referred as automatic tree design procedures in

[1]This work was supported in part by NSF grant IRI-9002087

the pattern recognition literature where such tree building procedures have been the focus of research for several years. These procedures are also well-known in the machine learning literature where the data-driven construction of decision trees is synonymous with the acquisition of structured knowledge in the form of concepts and the expert system rules. The successive versions of the *TDIDT* (Top Down Induction of Decision Trees) family of learning systems represent a major machine learning approach to the decision tree construction [31].

Although decision trees are intuitively appealing and have been successfully used in many applications, there are several problems that can hamper their use and performance in many instances. These problems relate to issues such as the process of tree design itself, consequences of hard decision making, missing feature values, and the simultaneous use of all the training vectors. While many of these issues have been addressed in the past by several researchers within the framework of statistical pattern recognition and machine learning, it appears that the performance of decision tree classifiers can be enhanced by exploiting some of the keys features of multilayer perceptron (MLP) networks which are similar to decision tree classifiers in certain respects [39,40]. For example, both types of classifiers do not impose any restriction on the distribution of input observations and are capable of producing arbitrarily complex decision boundaries that can be learned from a set of training vectors. The aim of the present chapter is to show the presence of a link between the tree classifiers and multilayer perceptron (MLP) networks and demonstrate how such a link can be beneficial in enhancing the performance of decision tree classifiers in many ways by implementing them in the form of a three-layer trainable neural structure.

The organization of the chapter is as follows. Section 2 provides an overview of the past research related to decision tree performance issues. After providing a brief introduction to the multilayer perceptron networks in Section 3, a relationship between the tree classifiers and the MLP networks is given in Section 4 which leads to decision tree implementation in the form of a three-layer neural network. Section 5 discusses the training and advantages of such an implementation. Results from some experiments are presented in Section 6 to demonstrate the performance enhancement due to the neural network implementation. Finally, a summary of the chapter is given in Section 7.

2. DECISION TREE CLASSIFIER ISSUES

There are some excellent sources on decision tree classifiers [6,10,23] which should be consulted for more detailed discussions into the various aspects of the decision tree design and performance. In this section, only a brief overview is given.

2.1 Classifier Tree Design

Several automatic tree generation algorithms exist in the pattern recognition literature where the problem of tree design has been treated in two distinct ways. In one approach, the tree design process is broken into two stages. The first stage yields a set of prototypes for each pattern class. These prototypes are obtained from the training set using procedures similar to clustering. Next, these prototypes are viewed as entries in a decision table which is converted into a decision tree using some criterion for optimality. Examples of this type of tree design approaches can be found in [1,17,19,30,36]. The problem of finding prototypes from binary or discrete-valued patterns is considered in [37,41]. The other tree design approach is more universal and can be considered as a generalization of the decision table conversion approach with all the training examples forming entries of the decision table. Since each training pattern is individually used to construct the tree, this approach allows decision boundaries of any arbitrary shape. Examples of the direct top-down tree design approaches can be found in [15,26,33,38,43,45]. While some of these tree generation algorithms can handle only two classes at a time, there are several that are suitable for multifeature, multiclass pattern recognition problems.

The foremost requirement in a top-down tree design procedure is an evaluation criterion to determine the goodness of a particular partitioning of the training vectors. One popular approach for selecting successive subgroupings of the training vectors is based on defining a goodness measure of partitioning in terms of mutual information [15,22,38,43,45]. Consider a two-class problem with only one feature x. Let $x = t$ define the partitioning of the one-dimensional feature space. If we view the feature x taking on values greater or less than threshold t as two outcomes x_1 and x_2 of an event X, then the amount of average mutual information obtained about the pattern classes from the observation of event X can be written as

$$I(C;X) = \sum_{i=1}^{2} \sum_{j=1}^{2} p(c_i,x_j) \log_2 [\, p(c_i / x_j) / p(c_i) \,] \tag{1}$$

where C represents the set of pattern classes and the $p(\,.\,)$'s are the various probabilities. Clearly, for better recognition, the choice of the threshold t should be such that we get as much information as possible from the event X. This means that the value which maximizes (1) should be selected over all possible values of t. Average mutual information thus provides a basis for measuring the goodness of a partitioning.

Another popular criterion for partitioning is the Gini index of diversity used in the CART (Classification and Regression Trees) procedure [5]. In this criterion, the impurity of a set of observations at a partitioning stage s is defined as

$$I(s) = \sum_{i \neq j} p(c_i / s) p(c_j / s) \tag{2}$$

where $p(c_i / s)$ denotes the conditional probability. Data are further split by selecting a partitioning that yields greatest reduction in the average data impurity. The advantage of this criterion is its simpler arithmetic. The other examples of the partitioning evaluation criterion include the use of Kolmogorov-Smirnov distance between estimated cumulative distributions [33] and the permutation statistics [26].

Although for a given collection of training vectors a tree can be grown to yield 100% classification rate on the training vectors, doing so is generally equivalent to "noise fitting". Thus knowing when to stop splitting is very important in a top-down recursive tree design method In the simplest case, the number of training vectors left for further split can be used as a measure for termination. Another possibility is to test the statistical significance of the reduction in uncertainty due to a partitioning. In some cases, the classification error performance, estimated either empirically or theoretically, is used to terminate the tree growing process. For example, the AMIG (Average Mutual Information Gain) algorithm [38] uses the following inequality [11] that determines the lower limit on the average mutual information, $I(C,T)$, to be provided by the tree for the specified error performance P_e

$$I(C,T) \geq H(C) - H(P_e) - P_e \log_2(m-1) \tag{3}$$

where $H(C)$ *and* $H(P_e)$, respectively, represent the pattern class and the error entropy and m indicates the number of pattern classes. Recently, Goodman and Smyth [15] have derived several fundamental bounds for mutual information-based recursive tree design procedures and suggested a new stopping criterion which is claimed to be more robust in the presence of noise

Instead of using a stopping criterion to terminate the recursive partitioning, CART uses a pruning approach In this approach, the recursive partitioning continues till the tree becomes very large This tree is then selectively pruned upwards to find a best subtree having the lowest error estimate. Trees obtained using pruning are typically less biased towards the design samples but this property is achieved at higher computational expense. Recently, Gelfand and Delp [13] have developed an efficient pruning procedure where the training set is divided into two subsets to iteratively grow and prune the tree. In each iteration, the roles of the two subsets are reversed. Their results indicate that the iterative tree growing and pruning requires less computation compared to the CART procedure.

Once a node is designated as a terminal node due to the stopping criterion, a decision rule is set up for that node to assign classification labels to unknown vectors for the future use of the tree as a classifier. The usual practice is to mark a terminal node with a class label that is in majority over its corresponding training vectors Given a partitioning measure, a stopping criterion and a method to set up a decision rule at each terminal node, the problem of tree design then essentially reduces to a search problem for finding the best decision function at each step of the tree development. It is a common practice to simplify the search by enforcing a binary decision based on a single feature at each of the nonterminal nodes. The AMIG algorithm is one such example of the recursive tree design procedure that seeks to maximize the amount of mutual information gain at every stage of tree development by essentially employing a brute force search technique to determine the best feature for that stage along with its best threshold value. Since the orientation of dividing hyperplanes is restricted, i e only one feature is used at any internal node, the search space for maximizing the average mutual information gain is small. The search is made efficient by ordering the labeled patterns along different feature axes to obtain a small set of possible candidate locations along each axis.

In addition to easing computational burden, another advantage of using a single variable decision function at every nonterminal node is that the resulting trees have better interpretability. However, the trees obtained using univariate partitioning are generally larger and occasionally yield poor results, especially in cases where the tree classification is attempted with raw measurements without extracting features Although linear discriminant analysis offers a solution to the problem of recursive partitioning using a linear combination of several variables [12], its application in the past has been limited due to the fact that as the partitioning proceeds, the partitioned training vectors start residing in the subspaces giving rise to almost singular covariance matrices Recently, a modified version of linear discriminant analysis incorporating several statistical techniques including the analysis of variance and the principle component analysis has been suggested by Loh and Vanichsetakul [28] to generate multivariate partitionings. However, the advantages of using multifeature splits are questionable in nonparametric situations. Brieman and Friedman [6] point out that "in most applications where recursive partitioning has higher accuracy than traditional methods, that advantage is achieved through univariate rather than linear combination splitting " In fact, according to Brieman et al. [5], obtaining a tree of proper size is much more crucial to its performance than the choice of the partitioning measure used to develop it.

2.2 Missing Features

In many applications such as medical diagnosis, one or several components of a pattern vector are often found missing. In such cases, classification using decision trees can not be ordinarily done because a missing feature may be involved in a test at an internal node. One solution to this problem, suggested by Bratko and Kononenko [4], is to follow all the decision tree branches from an internal node requiring the missing feature value. Each branch is assigned a probability value which approximates the chance of having taken that branch if the value for the missing feature was known. These probabilities values are compiled at the tree design time and are taken into account at terminal nodes to determine the correct identity of the pattern with missing features. However, the classification performance using this strategy for handling missing information does not appear to be any better than some even simpler schemes such as filling in the missing value with the most common or median value.

CART employs a better strategy for dealing with missing feature values. Each internal node is assigned two tests: a primary test and a surrogate test. The primary test is the one that is normally used The feature variable for the primary test and its threshold value are determined by the usual partitioning criterion. The feature variable for the surrogate test and its corresponding threshold values are determined by the additional criterion of having the best predictive association with the primary split at the node. It has been found that following the surrogate split strategy there is only a slight deterioration in the classification performance with the missing data provided the features have good correlation. An additional advantage of using the surrogate test technique is that it uncovers some important features that may never appear in primary tests but have a role to play by virtue of their strong association with several other features.

2.3 Hard and Soft Decision Making

The same splitting process that is key to the decision tree design is perhaps its biggest drawback when the tree is later used as a classifier As a result of making a hard choice to either move down on the left branch or the right branch, the decision trees give rise to decision models that are too simplistic for many applications. As an example, consider the decision space of Figure 1(a) for some hypothetical machinery where the shaded region represents the

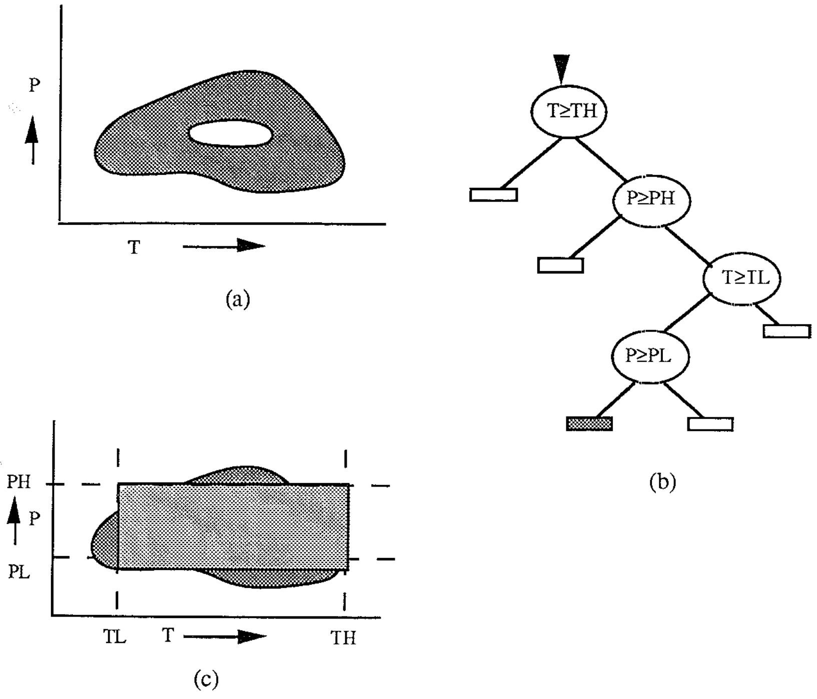


Figure 1. (a) Temperature and pressure space for a hypothetical machinery. The shaded region represents the healthy state of the machinery. (b) A decision tree to identify the status of the machinery. (c) Approximation of the healthy region by the decision tree.

healthy state of the machinery in a measurement space of two continuous features, temperature and pressure. An approximation of the decision region using the decision tree of Figure 1(b) is shown in Figure 1(c) It can be noticed from these figures that either we must increase the tree size or use nonlinear node decision functions to obtain a better approximation of the decision space.

Another consequence of hard splitting at the internal nodes is the sensitivity to the noise. Since the tests at internal nodes typically involve comparing a continuous feature value against a threshold to determine branching, a small change in the feature value due to noise can cause a significant change in the classification. This problem gets exacerbated as the tree structure is moved from the laboratory environment to the field environment for actual use. It, therefore, becomes important to allow for a more flexible or adaptive decision making at each node in a decision tree One source for flexibility is to replace hard decision making at each internal node with a soft decision making scheme where none of the descendent terminal nodes are ruled out as possible class labels; instead each class label is assigned a posteriori probability based on the outcome of the test performed at the internal node [9,32,35] These values are passed on to subsequent descendent nodes. The final decision is made in accordance with the terminal node having maximum a posteriori probability. Such an approach leads to better performance and more accurate modeling of the underlying relationship present in the training examples

One simple approach to soft decision making proposed by Quinlan [32] consists of defining a small interval around each threshold value to assign a probability measure to the outcome of each test of the type "*is fvalue ≥ thrsh* " made in the tree. If the observed feature value lies inside the interval, the probability of the test outcome is made proportional to the difference of the observed feature value from the threshold value as shown in Figure 2; otherwise the probability value assigned to the test outcome is either one or zero depending upon whether the observed feature value is higher or lower than the threshold Quinlan suggests a method for determining the interval size which involves finding the variation in the classification error rate with respect to the variation in the threshold value.

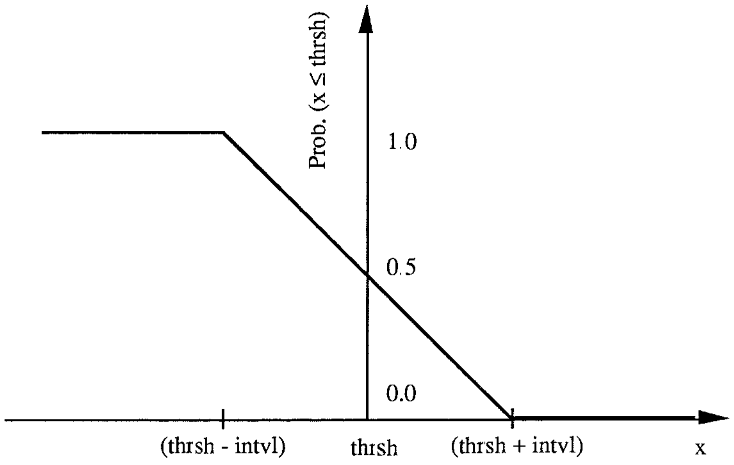


Figure 2. Variation of the test outcome probability with respect to the observed feature value.

2.4 Simultaneous Use of Training Vectors

One difficulty with the top-down recursive tree design methods is that they require all the training vectors to be present in memory at once. For pattern classification problems with high dimensional vectors, it may not be practical to simultaneously store all the training vectors. To overcome this difficulty, the incremental generation of trees has been explored by some researchers. One example of incremental tree generation approach is the *windowing* technique [31]. In this approach, a small subset of the training vectors chosen at random is used as a design set to develop a decision tree. The remaining patterns from the training set are used to evaluate the quality of the tree. If the tree performance is found satisfactory, then the tree building process terminates; otherwise the design set is enlarged by incorporating those training vectors that were misclassified in the first instance and the entire tree building process is repeated. The justification for the windowing approach is based on the assumption of redundancy present in a large training set. Wirth and Catlett [48] have done an extensive study of the decision tree performance with and without windowing for many well known problems in machine learning literature. According to their experience, windowing does not offer any overall advantage in tree design and should not be used unless the limitations of memory space are unavoidable by other means.

Another approach to minimize memory requirements for tree building consists of making incremental modifications in the existing tree structure as more and more training vectors are considered. Similar to the windowing approach, a subset of training vectors is first used to develop a decision tree. Instead of redesigning the entire tree in the presence of poor performance on the remaining training vectors, the incremental approach opts for adapting the existing tree structure either by replacing a subtree with another subtree or by reshaping a subtree that involves replacing the root node of the subtree with one of its descendent nodes following certain criterion. Utgoff [44] has shown that reshaping technique is most cost and performance effective. However, the main difficulty with this approach is the lack of well-defined tree modification rules.

3. MULTILAYER PERCEPTRON NETWORKS

Multilayer perceptron (MLP) networks are feedforward networks having several layers of simple computing elements, called neurons or perceptrons, with signal flow taking place in the forward direction only. The interfacing layer on the input side of the network is called the sensory layer; the one on the output side is referred to as the output layer or the motor control layer All the intermediate layers are called hidden layers. One of the most important attribute of MLP networks is their capability to capture input-output relationship present in a set of training examples.

An example of a MLP network is shown in Figure 3(a). Generally, all neurons in a layer are connected to all the neurons in the adjacent layers through unidirectional links The connection strength between two neurons from adjacent layers is represented in the form of a weight value. The significance of this weight value is that it acts as a signal multiplier on the corresponding connection link. Each neuron in the layered network is typically modeled as shown in Figure 3(b). As indicated in the figure, the input to a neuron is the linear summation of all the incoming signals on the various connection links. This net summation is compared against a threshold value, often called bias. The difference arising due to the comparison drives an output function to produce a signal at the output line of the neuron. The two common choices for the output function are sigmoid and hyperbolic tangent functions.

Each layer in these networks performs a certain transformation on its input signals. Given a sufficiently large number of layers and a capability to manipulate the layer transformations, it is possible to achieve any desired input-output mapping or decision boundary for classification. Although a large number of hidden layers perhaps provides more flexibility in terms of achieving a mapping, it can be easily shown that two hidden layers are sufficient to form piecewise linear decision boundaries of any complexity [7,27] The first of these two hidden layers can be considered as the *partitioning* layer that divides the entire feature space into several regions The second hidden layer functions as an *ANDng* layer that performs *ANDing* of partitioned regions to yield convex decision regions for each class. The output layer can be considered as the *ORing* layer that logically combines the results of the previous layer to produce disjoint decision regions of arbitrary shape with holes and concavities, if needed

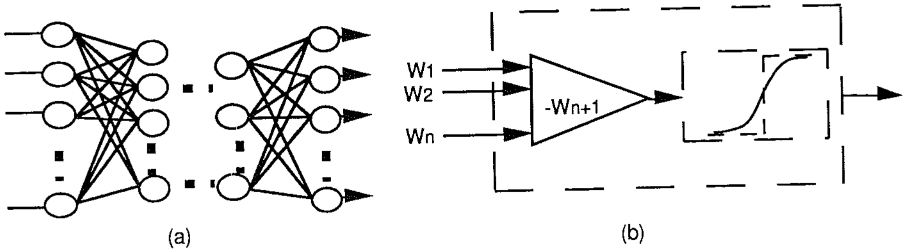


Figure 3. (a) An example of a multilayer perceptron network. (b) A typical perceptron model

The network training is done in an incremental fashion by presenting examples of input-output mapping pairs in a sequence. During the learning process the network continuously modifies its connection strengths or weights to achieve the mapping present in the examples. Since the input-output training examples specify only the desired output for the neurons in the output layer, the expected response of the neurons in the intermediate layers is determined by back propagating the error at the output layer to the intermediate layers through a process of *credit assignment* that takes into account the weights of the various interconnecting links. The resulting backpropagation algorithm [29,34,46] is a gradient descent procedure that minimizes the error at the output layer. Although the convergence of the algorithm has been proved only under the assumption of infinitely small weight changes, practical implementations with larger weight changes appear to yield convergence most of the time. Because of the use of gradient search procedure, the backpropagation algorithm occasionally leads to solutions that represent local minima. Recently, many variations of the backpropagation algorithm [8, 42] have been proposed to speed up the network learning time.

Because of their excellent learning capabilities the MLP networks are finding increasing use as nonparametric pattern classifiers. These networks have been successfully used for classification tasks involving speech, text, and sonar data with performance similar to that of conventional nonparametric classifiers such as the k-nearest neighbor classifier with the benefit of inherent parallelism of the neural net structure. One difficulty often encountered in the successful application of MLP networks is the necessity of matching the network topology, i.e. the number of neurons in each layer, to the given problem at hand. The network topology is an important factor that can significantly affect the learning time as well as the overall input-output mapping performance as indicated by many experimental studies [7,16,20]. Considering the importance of network topology, many researchers [2,3] are looking at the traditional statistical classification techniques to obtain the matching network configuration for a given set of input-output training vectors.

4. AN MLP IMPLEMENTATION OF TREE CLASSIFIERS

One of the earliest references to the implementation of a tree classifier as a layered neural network is the work of Henrichon and Fu [19] who suggested the use of perceptrons with hard nonlinearity to realize the node decision functions Similar schemes have been proposed again in the recent literature in the form of neural decision trees [14,25]. However, the implementation proposed here makes use of a structural transformation [39] that leads to a decision tree implementation in the form of a three-layer feedforward network of perceptrons which can be further trained along the lines of neural network training procedures

In order to see how a decision tree can be transformed into a three layer neural network, let us consider the decision tree classification procedure Classification using decision tree is performed by traversing the tree from root node to one of the leaf nodes using the unknown pattern vector. The response elicited by the unknown pattern is the class or decision label attached to the leaf node that is reached by the unknown vector It is obvious that all the conditions along any particular path from the root to the leaf node of the decision tree must be satisfied in order to reach that particular leaf node. Thus, each path of a decision tree implements an *AND* operation on a set of half-spaces that are defined by the intermediate nodes on that path. If two or more leaf nodes result in the same action or decision then the corresponding paths are in *OR* relationship Since a layered neural network for classification also implements *ANDing* of hyperplanes followed by *ORing* in the output layer, a decision tree can be modeled as a layered network by following some transformation rules These rules can be informally stated as follows:

(i) The number of neurons in the first hidden layer of the layered network equals the number of internal nodes of the decision tree. Each of these neurons implements one of the decision functions of internal nodes This layer is the *partitioning* layer.

(ii) All leaf nodes have a corresponding neuron in the second hidden layer where the *ANDing* is implemented. This layer is the *ANDing* layer.

(iii) The number of neurons in the output layer equals the number of distinct classes or actions. This layer implements the *ORing* of those tree paths that lead to same action.

(iv) The connections between the neurons from the partitioning layer and the neurons from the *ANDing* layer implement the hierarchy of the tree.

An example of restructuring following the above rules is shown in Figure 4. As this example shows, it is fairly straightforward to map a decision tree into a layered network of neurons It should be noted that the mapping rules given above do not attempt to optimize the number of neurons in the partitioning layer. However, a better mapping can be achieved by incorporating checks in the mapping rules for replications of the node decision functions in different parts of the tree to avoid the duplication of the neurons in the partitioning layer. Moreover, it may not be necessary to have the output layer if the number of neurons in the *ANDing* layer is same as the number of classes.

While the mapping rules given above allow an implementation of a decision tree as a three layer network of perceptrons, the full potential of this implementation can not be realized unless the perceptrons are provided with soft nonlinearity and the network is allowed to adapt its connection strengths to overcome the rigidity of the decision tree classifiers There are two possible methods for doing so. One is to simply use the backpropagation learning procedure. However, such a training procedure may take exceedingly long time as the mapped network is partially connected. The other possibility is to exploit the presence of hierarchy in the transformed network to develop a more suitable training method. One such method is described in the next section.

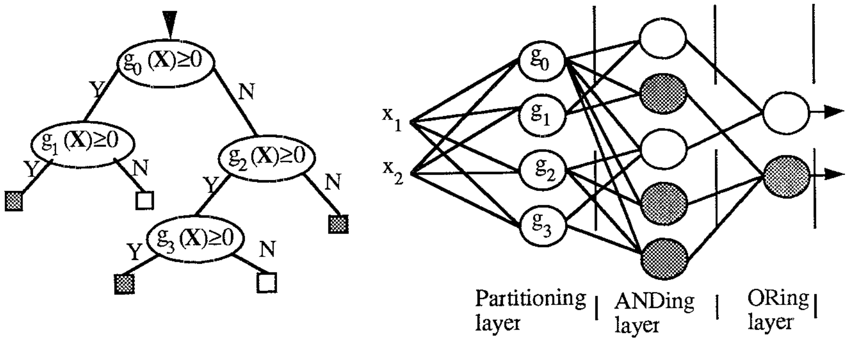


Figure 4. An example of a decision tree mapping

5. TRAINING THE TREE MAPPED NETWORK

To be able to adjust weights in the mapped network, it is essential to know the desired output of the neurons in the partitioning layer and the ANDing layer as well While the desired output of the neurons in the intermediate layers is generally not known, this problem is tractable in the present case. As can be noticed from the mapping rules, there exists a group of neurons for every pattern class in the ANDing layer of the network. The membership in this group is known from the tree-to-network mapping. Thus given an example pattern from class *c*, it is known that only one neuron from the group *c* of the ANDing layer neuron should fire while the remaining neurons from that group as well as those from other groups should remain in an inactive state. Therefore, the solution to weight adjustment for the ANDing layer is very simple: enhance the response of the neuron producing highest output among the neurons from group *c* and suppress the response of the remaining neurons in the ANDing layer for a pattern from class *c*. This is similar to the *winner-take-all* approach followed for the neural net training in the self-organizing mode [24]. The reason that this simple approach works is that the mapped network has a built-in hierarchy of the decision tree which is not present in the other layered networks except for the counter propagation network [18] where also a similar training procedure is used. Once the identity of the firing neuron in the ANDing layer is established for a given example pattern, the desired response from the partitioning layer neurons is also established because of association between a terminal node and internal nodes on its path

Although the presence of tree hierarchy in the mapped network provides a solution to the problem of knowing desired response for weight adjustment for the neurons in the partitioning layer, it is really not necessary to adjust weights on the incoming connections to these neurons due to the use of soft nonlinearity. This is due to the fact that using soft nonlinearities such as sigmoid or hyperbolic tangent functions causes the difference between the observed feature value and the threshold value to be carried across different layers in a coded form This in conjunction with the adaptability provided by the connections between the partitioning layer and the ANDing layer is generally sufficient to obtain desirable performance from the mapped network without a need to adjust the weights on the incoming links to the partitioning layer.

Based on the above discussion, the procedure for adjusting the weights in the mapped network can be described in the following way. Let $x(p)$ with class label $L(x(p))$ be the input pattern to the mapped network at the p-th presentation during the training Let $R_j(x(p))$ denote the response of the j-th neuron from the ANDing/ORing layer. Let $G(j)$ represent the group membership of the j-th neuron, i.e. the class label of the group of neurons whose member is the j-th neuron. Furthermore let w_{ij} be the connection strength between the j-th neuron and the i-th neuron of the previous layer. Then

$$w_{ij}(p+1) = m_{ij}.(w_{ij}(p) + \Delta w_{ij}(p)) \quad \text{if } R_j(x(p)) \geq R_k(x(p)) \text{ for all } k$$
$$\text{such that } G(k) = L(x(p)) = G(j)\text{; and}$$
$$w_{ik}(p+1) = m_{ik}.(w_{ik}(p) - \Delta w_{ik}(p)) \quad \text{for all } k \neq j,$$

where the amount of change in the weights is determined by the Widrow-Hoff procedure [47] or the LMS rule as it is called many times. The term m_{ij} is either *'1'* or *'0'* indicating whether a connection exists to the j-th neuron from the i-th neuron or not. It should be noted that the presence or the absence of the connections is determined at the time of tree-to-network mapping. The suggested training procedure is such that it is possible to train each layer separately or simultaneously.

Examining the proposed decision tree implementation as a trainable three layer network of perceptrons, several comments with respect to the decision tree issues raised earlier can be made. The most important of these is that the MLP implementation of the decision trees is expected to provide a better and more robust classification performance because the final decision making is delayed to the last layer when information from all the previous layers is available. This is in contrast with the usual decision making procedure in a decision tree where a choice has to be made at each node. Moreover, the use of soft nonlinearities in the MLP implementation allows perturbations in the feature values to be tolerated which in the usual decision tree implementation can lead to an entirely different path. Since a soft nonlinearity retains more information about the input, another important consequence of the MLP implementation is that the final decision boundaries are expected to be smoother and of arbitrary orientation leading to better decision models than are possible with the conventional decision tree implementation. This capability in the MLP implementation of the decision trees also reduces the need for finding multivariable splits while developing the tree. The capability to adjust the weights in the proposed implementation also provides a solution to the problem of simultaneous use of all the training vectors while designing a decision tree. With the MLP implementation, it is possible to design the tree using only a subset of the training vectors The remaining training vectors can be used in an incremental fashion to adjust the weights of the mapped network thus avoiding the need to have all training vectors in the memory at the same instant. The size of the tree also becomes less crucial with the MLP implementation Ordinarily when a decision tree is grown beyond the appropriate size, the corresponding feature space partitioning starts getting more and more biased towards noisy training vectors. This later on leads to poor classification performance. However when an overgrown tree is mapped in to its corresponding MLP structure, the effect of inappropriate tree size is expected to be minimal as the ANDing layer will eliminate or weaken, through the use of competitive training procedure, neurons that correspond to those terminal nodes of the tree that are due to inappropriate size [39]. The issue of missing features is also less crucial in the neural implementation because of the parallel nature and graceful degradation property of the neural networks.

6. PERFORMANCE EVALUATION

A series of experiments were performed to compare the performance of the traditional tree classifier implementation with the MLP implementation as described earlier. Three well known data sets were used in these experiments. The first two data sets simulate data for two well-known problems from the classification tree literature [5] The first synthetic data set, called LED data, simulates a faulty LED display. Each displayed digit is represented as a seven component binary vector that forms an input to the classifier. The classifier output indicates the digit represented by the seven component binary input vector. The fault in the display is such that it causes a switching from a correct segment state to an incorrect state with a probability of 0.10 for each segment in an independent fashion. All displayed digits are assumed to have equal probability. Two hundred patterns of such faulty data were generated to be used as training patterns while another 5000 patterns were generated to function as test vectors. Figure 5(b) shows the decision tree for recognizing faulty LED display that was obtained using the training vectors. The number in each circular node of the tree refers to a segment in the display as shown in Figure 5(a). If the segment is found to be turned on, the right branch in the tree is taken; otherwise the left branch is followed. The number in a square box in the tree represents a digit label. This tree as well as the trees for the other data sets were all obtained following the average mutual information gain (AMIG) tree design procedure.

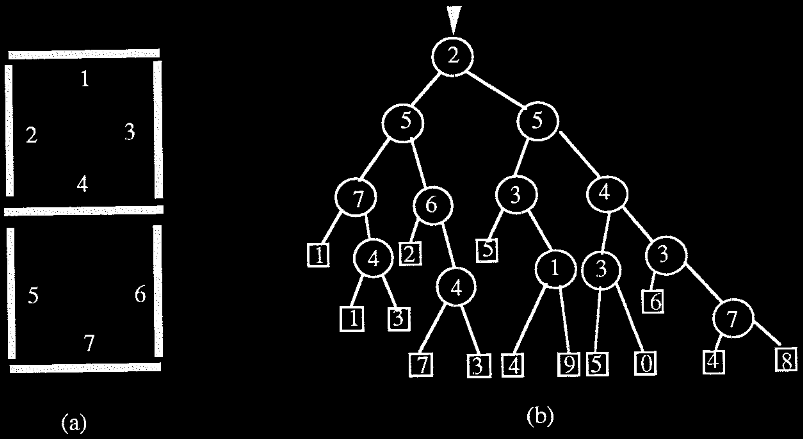


Figure 5 (a) LED display (b) decision tree for the faulty LED data.

The second simulated data set corresponds to the waveform recognition problem. Unlike the LED data that is binary in nature, the WAVE data consists of 21-dimensional continuous valued feature vectors coming from three classes. Data from each class are generated by combining two of the three waveforms of Figure 6(a) at equi-spaced 21 sampled positions. Each component of the 21-dimensional feature vector is corrupted by random noise drawn from a normal distribution of zero mean and variance 1. The training and test set sizes are 300 and 5,000, respectively with equal a-priori probability for all classes. The two entries within each internal node of the tree in this case represent a (feature, threshold) pair with the convention that the left branch is followed if the observed feature value is less than the threshold value; otherwise the right branch is taken up. It is to be noted that only a small number of the features are actually utilized in constructing the decision tree. These are the features that are considered most discriminatory.

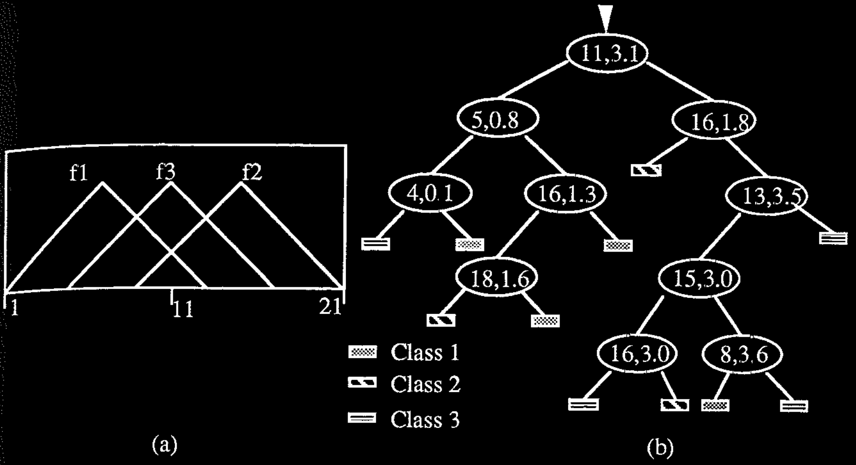


Figure 6. (a) Three waveforms (b) decision tree

The third data set is taken from character recognition domain and uses a subset of Munson handprinted FORTRAN character set [21]. The subset, called IMOX data, consists of four character classes, I, M, O, and X. Each character class consists of 48 patterns. Each pattern is represented as an eight-dimensional feature vector of integer valued components These features represent the length of eight directional lines in terms of pixel counts as shown in Figure 7(a). In the experiment conducted with this data, 36 labeled patterns from each category were used to develop the decision tree of Figure 7(b). The remaining 12 patterns from every category were used to test the relative classification performance of the decision tree and the trained network.

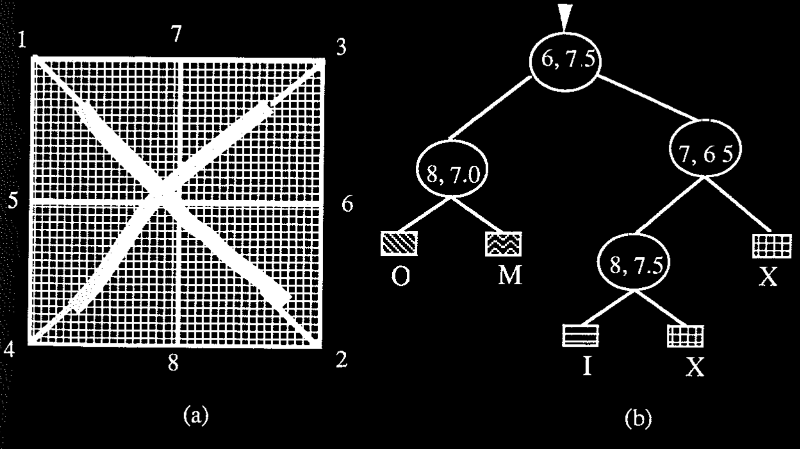


Figure 7. (a) IMOX features (b) decision tree

After mapping each tree into its corresponding MLP network, the training of the mapped network was done using the procedure discussed earlier. The training set consisted of the same vectors that were used to develop decision trees. There are two parameters, ρ and α, involved in the training procedure. The parameter ρ determines the initial learning rate and was set equal to 1.0. In all the experiments, learning rate was decreased in inverse proportion to the square root of the iteration number. The other parameter α determines the shape of the neural output function which was realized through a hyperbolic tangent function. Figure 8 shows the output function for three different α values of 0.25, 0.50, and 1.0. Since a large α value brings the output function closer to the step function, we call α a generalization constant that determines the degree of flexibility of the mapped network. After experimenting with few values, 0.25 was found to be the most suitable α value.

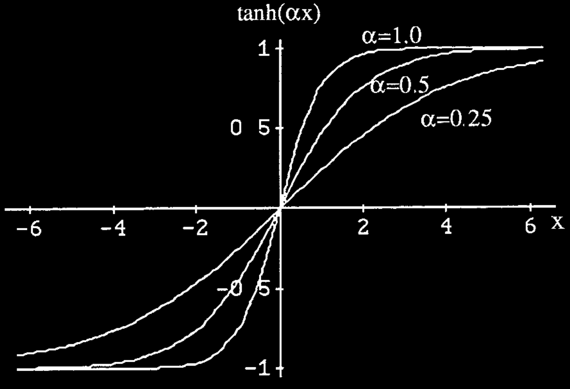


Figure 8 Plots of hyperbolic tangent function *tanh(αx)* for three α values

In order to monitor the learning progress, it was decided to perform classification on the test data with the mapped net after every ten iterations of weight adjustment with the training data. The initial choice for the weights was made randomly. The training procedure was repeated many times with different initial weight values. No significant differences, either in terms of the training time or the classification performance, were observed due to initial weight selection. Figure 9 shows an example of the learning progress of the mapped networks for the LED and WAVE data sets that was observed in the experiments. It is seen that a stable classification performance is provided by these networks after going through few tens of iterations. The number of iterations in the case of WAVE data is far less than the LED data and additionally the learning progress curve is much smoother. This is possibly due to the analog nature of the WAVE data which provides more flexibility as opposed to the binary nature of the LED data. The learning progress rate for the IMOX data was found to be similar to the WAVE data

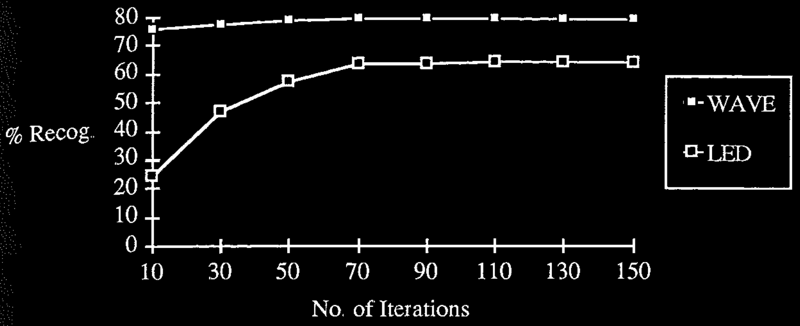


Figure 9. The learning progress curves for the mapped network. $\alpha = 0.25$ and $\rho = 1.0$

The relative classification performance of the decision trees and the corresponding MLP implementations is shown in Figure 10. These performance results were obtained using the test data in each case. It is seen that the MLP implementation in each case provides an improvement over the corresponding decision tree performance. The reason for improvement is essentially due to the use of soft nonlinearity and the combination of the output of different internal nodes of the decision tree that becomes possible through an MLP implementation. It is interesting to note that the improvement for the LED data is very small compared to the WAVE data or the IMOX data. The reason for smaller improvement in the case of LED data is because the binary data does not provide as much flexibility as the continuous data

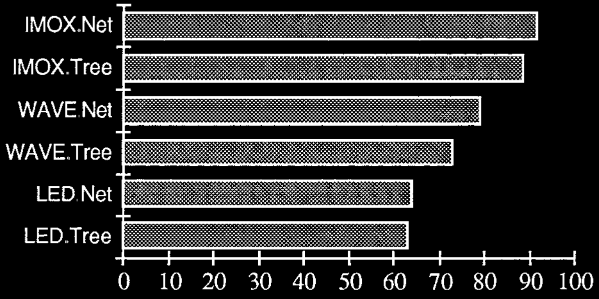


Figure 10. Relative recognition performance.

7. CONCLUSIONS

A neural network implementation of the decision trees has been described. The proposed implementation is based on a set of mapping rules that transform a decision tree into a three layer structure of partially connected neurons. It has been shown that the transformed structure can be trained following the winner-take-all strategy. The incorporation of the soft nonlinearities in the neurons has been shown to overcome many of the decision tree deficiencies. The experimental results that compare the classification performance of the neural implementation with the traditional tree classifier implementation show that the neural implementation is able to provide improved performance.

REFERENCES

[1] P. Argentiero, R. Chin, and P. Beaudet, "An automated approach to the design of decision tree classifiers," *IEEE Trans. PAMI*, Vol.4, pp.51-57, 1982.

[2] E.B. Baum and D. Haussler, "What size net gives valid generalization?," in D.S. Touretzky (ed.), *Advances in Neural Information Processing Systems I*, Morgan Kaufmann Pub., 1989.

[3] M Bichsel and P. Seitz, "Minimum class entropy: A maximum information approach to layered networks," *Neural Networks*, Vol 2, pp. 133-142, 1989.

[4] I. Bratko and I. Kononenko, "Learning diagnostic rules from incomplete and noisy data," in B. Phelps (Ed.), *Interactions in AI and Statistical Methods*, Technical Press, Hants, England, 1987.

[5] L. Breiman, J. Friedman, R Olshen, and C.J. Stone, *Classification and Regression Trees*, Wadsworth Int'l Group, Belmont, Calif., 1984.

[6] L. Breiman and J. Friedman, "Comment on "Tree-structured classification via generalized discriminant analysis"," *J Amer. Statistical Association*, Vol 83, No 403, pp.725-727, 1988.

[7] D.J. Burr, "Experiments on neural net recognition of spoken and written text," *IEEE Trans. ASSP*, Vol. 36, pp.1162-1168, July 1988.

[8] E.D. Dahl, "Accelerated learning using the generalized delta rule," *Proc. IEEE First Int'l Conf Neural Networks*, San Diego, California, pp. II-523-530, 1989.

[9] G.R. Dattatreya and L.N. Kanal, "Adaptive improvement of pattern recognition trees," *Proc IEEE Int'l Conf. Systems, Man, and Cyber.*, Bombay, India, pp. 393-397, 1983.

[10] G.R. Dattatreya and L.N. Kanal, "Decision trees in pattern recognition," in L.N Kanal and A. Rosenfeld (Eds.), *Progress in Pattern Recognition 2*, Elsevier Science Publishers B.V. (North-Holland), 1985.

[11] R.M. Fano, *Transmission of Information*, John Wiley & Sons, New York, 1963.

[12] J.H. Friedman, "A recursive partitioning decision rule for nonparametric classification," *IEEE Trans Computers*, Vol.26, pp. 404-408, 1977.

[13] S.B Gelfand, C.S. Ravishanker, and E.J. Delp, "An iterative growing and pruning algorithm for classification tree design," *IEEE Trans. Pattern Anal Machine Intell*, Vol. 13, pp. 163-174, 1991.

[14] M. Golea and M. Marchand, "A growth algorithm for neural network decision trees," *Europhys Lett.*, Vol. 12, pp.205-210, 1990.

[15] R M. Goodman and P. Smyth, "Decision tree design from a communication theory standpoint," *IEEE Trans. Inform Theory*, Vol. IT-34, pp.979-994, Sept. 1988.

[16] R.P. Gorman and T.J. Sejnowski, "Analysis of hidden units in a layered network trained to classify sonar targets," *Neural Networks*, Vol.1, pp.75-89, 1988

[17] C R.P. Hartmann, P.K. Varshney, K G. Mehrotra, and C L. Gerberich, "Application of information theory to the construction of efficient decision trees," *IEEE Trans. Inform. Theory*, Vol.IT-28, pp. 565-577, July 1982.

[18] R. Hecht-Nielsen, "Counterpropagation networks," *Applied Optics*, Vol 26, pp. 4979-4984, December 1987.

[19] E.G Henrichon and K S. Fu, "A nonparametric partitioning procedure for pattern classification," *IEEE Trans. Comput*, Vol. C-18, pp. 614-624, July 1969.

[20] W.Y. Huang, and R.P. Lippmann, "Comparison between neural net and conventional classifiers, *Proc. IEEE First Int'l Conf Neural Networks*, San Diego,California, pp.

485-493, June 1987.

[21] A.K. Jain and M.D. Ramaswami, "Classifier design with parzen windows," in E.S. Gelsema and L.N Kanal (Eds.), *Pattern Recognition and Artificial Intelligence*, Elsevier Science Publishers B.V (North-Holland), 1988.

[22] J.H. Kagel and R.L. Saurer, "Autonomous target acquistion algorithms," *Proc. Sixth Annual KRC Symp on Ground Vehicle Signatures*, Houghton, Michigan, August 1984.

[23] L.N. Kanal, "Problem-solving models and search strategies for pattern recognition," *IEEE Trans. Pattern Anal. Machine Intell,*, Vol. PAMI-1, pp 194-201, Apr. 1979.

[24] T. Kohonen, *Self-organization and Associative Memory*, Springer-Verlag, Berlin, 1984.

[25] C. Koutsougeras and C A. Papachristou, "Training of a neural network for pattern classification based on an entropy measure, *Proc. IEEE Int'l Conf. Neural Networks*, Vol.I, San Diego, California, pp. 247-254, 1989.

[26] X. Li and R.C. Dubes, "Tree classifier design with a permutation statistic," *Pattern Recognition*, Vol. 19, pp.229-235, 1986.

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

[28] W.-Y. Loh and N. Vanichsetakul, "Tree-structured classification via generalized discriminant analysis," *J. Amer. Statistical Association*, Vol 83, No.403, pp.715-725, 1988.

[29] D.B. Parker, *Learning Logic*, Tech. Rep. TR-47, Center for Computational Research in Economics and Management Science, MIT, 1985.

[30] H.J. Payne and W S. Meisel, "An algorithm for constructing optimal binary decision trees," *IEEE Trans. Comput.*, Vol.C-25, pp.905-916, Sept. 1977.

[31] J.R. Quinlan, "Induction of decision trees," *Machine Learning*, Vol.1, pp 81-106, 1986.

[32] J.R. Quinlan, "Decision trees as probabilistic classifiers," *Proc. Fourth Int'l Workshop Machine Learning*, Irvine, California, pp 31-37, 1987

[33] E.M. Rounds, "A combined nonparametric approach to feature selection and binary decision tree design," *Pattern Recognition*, Vol.12, pp.313-317, 1980.

[34] D E. Rumelhart, G E.Hinton, and R.J.Williams, "Learning internal representation by error propagation," in D E. Rumelhart & J.L McClelland (Eds), *Parallel Distributed Processing : Explorations in the Microstructure of Cognition Vol 1. Foundations.*, MIT Press, 1986.

[35] J. Schuermann and W. Doster, "A decision theoretic approach to hierarchical classifier design," *Pattern Recognition*, Vol. 17, pp.359-369, 1984.

[36] I.K. Sethi and B. Chaterjee, "Efficient decision tree design for discrete variable pattern recognition problems," *Pattern Recognition*, Vol.9, pp.197-206, 1978.

[37] I.K. Sethi and B. Chaterjee, "A learning classifier scheme for discrete variable pattern recognition problems," *IEEE Trans. Systems, Man, and Cyber*, Vol SMC-8, pp. 49-52, Jan. 1978.

[38] I.K. Sethi and G.P.R. Sarvarayudu, "Hierarchical classifier design using mutual information," *IEEE Trans. PAMI*, Vol. PAMI-4, July 1982, pp.441-445.

[39] I.K. Sethi, "Entropy nets : from decision trees to neural nets," *Proceedings of the IEEE*, Vol.78, pp. 1605-1613, 1990.

[40] I.K. Sethi and M. Otten, "Comparison between entropy net and decision tree classifiers," *Proc Int'l Joint Conf. Neural Networks*, Vol.III, San Diego, California, pp.63-68, 1990.

[41] J.C. Stoffel, "A classifier design technique for discrete variable pattern recognition problems," *IEEE Trans. Comput*, Vol. C-23, pp.428-441, 1974

[42] W.S Stornetta and B A. Huberman, "An improved three layer backpropagation algorithm," *Proc. IEEE First Int'l Conf. Neural Networks*, pp. II-523-530, San Diego, Calif., 1987.

[43] J.L. Talmon, "A multiclass nonparametric partitioning algorithm," in E S Gelsema & L.N. Kanal (Eds.), *Pattern Recognition in Practice II*, Elsevier Science Pub B.V. (North

Holland), 1986.

[44] P E. Utgoff, "ID5: An incremental ID3," *Proc Fifth Int'l Conf Machine Learning*, Ann Arbor, Michigan, pp.107-120, 1988

[45] Q.R Wang and C.Y Suen, "Analysis and design of a decision tree based on entropy reduction and its application to large character set recognition," *IEEE Trans. PAMI*, Vol. 6, pp.406-417, July 1984.

[46] Werbos, P. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Ph.D. thesis, Harvard, 1974.

[47] B Widrow and M.E. Hoff, "Adaptive switching circuits," *1960 IRE WESCON Conv. Record*, Part 4, pp.96-104, 1960.

[48] J. Wirth and J. Catlett, "Experiments on the costs and benefits of windowing in ID3," *Proc Fifth Int'l Conf Machine Learning*, Ann Arbor, Michigan, pp 87-99, 1988.

Artificial Neural Networks and Statistical Pattern Recognition
Old and New Connections
I.K. Sethi and A.K. Jain (Editors)
© 1991 Elsevier Science Publishers B.V. All rights reserved

Bayesian and neural network pattern recognition : a theoretical connection and empirical results with handwritten characters

Dar-Shyang Lee[a], Sargur N. Srihari[a] and Roger Gaborski[b]

[a]Department of Computer Science, State University of New York at Buffalo, New York 14260

[b]Eastman Kodak Company, Rochester, New York 14650

Abstract

Statistical Pattern Recognition and Artificial Neural Networks provide alternative methodologies to the classification of patterns represented as feature vectors. This paper provides a theoretical relationship and an empirical comparison between the Bayes decision rule and the backpropagation model. It is shown that backpropagation performs least mean square approximation to the Bayes discriminant function. While a three-layer backpropagation network (one hidden layer) with a sufficient number of hidden units is known to possess universal mapping ability, gradient-descent based backpropagation learning does not guarantee finding the minimum probability of error solution. Experimental results with handwritten character recognition (digits and letters extracted from handwritten addresses) are presented. The experiments are with two different representations of characters : binary pixel arrays and structural features represented as binary vectors. With pixel arrays, the backpropagation model performs better than the first-order Bayes discriminant that assumes statistical independence between pixels. With structural features, the first-order Bayes and backpropagation have similar performance. However, training of a backpropagation network is much more involved. Inherent difficulties with both classifiers are discussed.

1 Introduction

Pattern recognition has been an active field of research for over thirty years inspiring many theoretical and experimental results. Well-developed statistical approaches to pattern recognition exist [Fuk72, DH73, TG74]. Recently much attention has been diverted to pattern recognition using the artificial neural network approach, with many successful experiments reported [Fuk88, Bur87, RMS89, DJG+89]. Weiss and Kapouleas [WK89] have done a thorough empirical comparison of pattern recognition, neural network, and machine learning classification methods.

This paper focuses on the statistically optimal Bayes classifier and a popular neural network architecture, the backpropagation network. In particular, our interest is in experimental evaluation for the case of binary feature vectors arising from the problem of handwritten character recognition. In analyzing these two classifiers from both theoretical and empirical aspects, we hope to gain some insight into their relationships.

The organization of the paper is as follows. Section 2 introduces the Bayes classifier and its approximation for the case of binary feature vectors. Section 3 introduces artificial neural networks and the backpropagation concept. Section 4 provides a theoretical analysis of the generalized delta rule used in a backpropagation network and relates it to the error rate achieved by the Bayes classifier. It is shown that backpropagation performs least mean square error approximation to the Bayes discriminant function. Section 5 describes experiments with both classifiers trained and tested on a database of more than 20,000 handwritten characters. Comparisons of these classifiers and practical problems in training them are discussed in section 6.

2 Bayes Classifier

The Bayes decision rule is a well-studied statistical classification method which is defined to be optimal with known *a priori* and *class-conditional* probabilities [DH73]. It minimizes conditional risks for any loss function.

Let the input x be a vector of d random variables, and let $C_1, C_2, \ldots, C_n$ be the n classes to which all possible input vectors may belong. Bayes rule states that

$$P(C_j|\mathrm{x}) = \frac{P(\mathrm{x}|C_j)P(C_j)}{P(\mathrm{x})} \tag{1}$$

where

$$P(\mathrm{x}) = \sum_{i=1}^{n} P(\mathrm{x}|C_j)P(C_j). \tag{2}$$

By assigning x to the class with maximum *a posteriori* probability, $P(C_j|\mathrm{x})$, the probability of error is minimized. Defining the discriminant function $g_j(\mathrm{x})$ for each class C_j as the *a posteriori* probability for class j

$$g_j(\mathrm{x}) = P(C_j|\mathrm{x}) \tag{3}$$

the mininum probability of error can be achieved by selecting the class whose discriminant function has the highest value, or

$$g_j(\mathbf{x}) = \frac{P(\mathbf{x}|C_j)P(C_j)}{P(\mathbf{x})}. \tag{4}$$

Since $P(\mathbf{x})$ is independent of the classes, it can be eliminated without affecting the decisions. We can further reduce this equation by applying a monotonically increasing function, such as log, on all discriminant functions. The result is

$$g_j(\mathbf{x}) = \log P(\mathbf{x}|C_j) + \log P(C_j). \tag{5}$$

In order to use this discriminant function, we need to know the probability density function of $\mathbf{x}$ with respect to each class C_j, and the *prior* probability of each class C_j. In practical pattern recognition problems, complete knowledge of either is rare. More often we are provided with a finite number of samples, and are forced to estimate the *a priori* and *class-conditional* probabilities based on the training set.

2.1 Binary Case

We will now consider the case where the components of $\mathbf{x}$ are binary. In general, the *a priori* probability, $P(C_j)$ can be estimated quite accurately if the training set is large However, approximation of class-conditional probabilities is a challenging task Since there are 2^d possible d-dimensional binary vectors, 2^d probabilities need to be calculated for each class. The large dimensionality often encountered in pattern recognition prohibits us from making such approximations.

2.1.1 First-Order Approximation

If the components of $\mathbf{x}$ are all statistically independent of each other, the class-conditional probability can be expressed as the product of conditional probabilities of each component, as

$$P(\mathbf{x}|C_j) = \prod_{i=1}^{d} P(x_i|C_j), \tag{6}$$

where x_i is the ith component of $\mathbf{x}$ Therefore, we need to estimate d probabilities for each class.

If we let

$$P_{ij} = P(x_i = 1|C_j) \tag{7}$$

and

$$1 - P_{ij} = P(x_i = 0|C_j) \tag{8}$$

then

$$P(\mathbf{x}|C_j) = \prod_{i=1}^{d} P_{ij}^{x_i}(1-P_{ij})^{1-x_i} \tag{9}$$

Substituting Equation 9 into Equation 5, it is simple to derive the following.

$$g_j(\mathbf{x}) = \sum_{i=1}^{d} x_i \log \frac{P_{ij}}{1-P_{ij}} + \sum_{i=1}^{d} \log(1-P_{ij}) + \log P(C_j). \tag{10}$$

Since this equation is linear in x_i, we can rewrite Equation 10 as

$$g_j(\mathbf{x}) = \sum_{i=1}^{d} x_i w_{ij} + b_j \tag{11}$$

where

$$w_{ij} = \log \frac{P_{ij}}{1-P_{ij}} \tag{12}$$

and

$$b_j = \sum_{i=1}^{d} \log(1-P_{ij}) + \log P(C_j) \tag{13}$$

Therefore, the discriminant function for each class can be efficiently characterized by a bias factor and a d-dimensional weight vector.

2.1.2 Estimating Parameters

Since the input vector is binary, and its components are independent under our assumption, the *class-conditional* probabilities can be estimated by building a nxd histogram. Then P_{ij} can be approximated by the sample mean k_{ij}/s_j, where s_j is the number of samples for class j, and k_{ij} is the number of occurrences of 1's in the ith component among all s_j samples. One common problem encountered using this approach is to determine a suitable value for P_{ij} when $k_{ij} = 0$, $s_j = 0$ or $k_{ij} = s_j$. Any one of those situations leads to representing either infinity or negative infinity. A useful technique that avoids this problem is to estimate P_{ij} by $(k_{ij}+1)/(s_j+2)$. In fact, it has been shown this is the best estimation in case of a mean square error loss function. Good results have also been observed by assigning P_{ij} the value of $1/3s_j$ when P_{ij} is 0

With a given set of feature vectors, with complete knowledge of *a priori* and *conditional* probabilities, the Bayes decision rule will provide the minimum error rate. The discriminant function in Equation 10 would be appropriate if the components in the feature vectors are indeed statistically independent and all conditional probabilities are known. In some pattern recognition problems, neither of those assumption can be justified. Consequently, working under the above assumptions, we run the risk of obtaining suboptimal performance.

2.1.3 Higher-Order Approximation

There are techniques which allow us to compromise between working under the assumption of strict statistical independence among components and having to estimate 2^d probabilities. One technique is to use a generalized decision function which has the form

$$\sum_{i=0}^{m} c_i \phi_i(\mathrm{x}), \tag{14}$$

where $\phi_i(\mathrm{x})$ are orthogonal functions and c_i are their coefficients

An example of a generalized decision function is one based on Bahadur-Lazarsfeld polynomials [DH73]. The *class-conditional* probability can be calculated by

$$P(\mathrm{x}|C_j) = P_I(\mathrm{x}|C_j) \sum_{i=0}^{2^d-1} a_{ji}\psi_{ji}(\mathrm{x}), \tag{15}$$

where

$$a_{ji} = \mathcal{E}[\psi_{ji}(\mathrm{x})] \tag{16}$$

is the expected value of $\psi_{ji}(\mathrm{x})$, and

$$\psi_{ji}(\mathrm{x}) = \begin{cases} 1 & i = 0 \\ y_1 & i = 1 \\ \vdots & \\ y_d & i = d \\ y_1 y_2 & i = d+1 \\ \vdots & \\ y_{d-1}y_d & i = d+1+d(d-1)/2 \\ y_1y_2y_3 & i = d+2+d(d-1)/2 \\ \vdots & \\ y_1y_2 \cdots y_d & i = 2^d - 1 \end{cases} \tag{17}$$

and y_i is the normalized variable,

$$y_i = \frac{x_i - P_{ij}}{\sqrt{P_{ij}(1 - P_{ij})}}$$

The weighting function $P_I(\mathbf{x})$ is precisely the conditional probability under the independence assumption,

$$P_I(\mathbf{x}) = \prod_{i=1}^{d} p_i^{x_i}(1-p_i)^{1-x_i}. \tag{18}$$

From the definition of ψ, it is clear that $a_{j0} = 1$ and $a_{j1} = a_{j2} \cdots = a_{jd} = 0$. Therefore, the discriminant function in Equation 10 is simply the first order approximation by Bahadur-Lazarsfeld expansion. The probability density function can be estimated more accurately by going into higher order terms. For our database, empirical results discourage us from such expansion as little is gained at a cost of many more coefficients. Nevertheless, in general this trade off between accuracy and efficiency will depend on the nature of the problem at hand

3 Artificial Neural Networks and Back Propagation

Artificial neural networks have been used in pattern recognition problems as early as 1957 when Rosenblatt introduced the perceptron [Ros57]. It is well-known that they are incapable of discriminating between non-linearly separable classes [MP69].

One major contribution in the revival of neural networks is the generalized delta rule of Rumelhart et.al. [MRG86], which was discovered independently by Werbos [Wer74] and Parker [Par85]. This rule provides a learning scheme for a multi-layer perceptron-like network with non-linear activation functions, thus allowing generation of any type of decision surface. Of course, other neural network architectures have been attempted for pattern recognition and other applications. However, we will not give a comprehensive overview of all neural network models here. A good introduction to various models can be found in [MRG86, Lip87].

Although the generalized delta rule can be applied to any network, we will concentrate on a particularly popular *layered feedforward* network, the backward error propagation (BP) model. The network consists of three types of layers, each of which is composed of various number of units. Units in adjacent layers are connected through links whose associated weights determine the contribution of units on one end to the overall activation of units on the other end. There are generally three types of layers. Units in the *input* layer bear much resemblance to the sensory units in a classical perceptron. Each of them is connected to a component in the input vector. The *output* units, analogous to the response units in a perceptron, represent different classes of patterns. Arbitrarily many *hidden* layers may be used depending on the desired complexity. Each unit in the hidden layer is connected to every unit in the layer immediately above and below.

Data flow in a BP can be either in forward or backward direction. In the forward direction, input data is transmitted from input to output layer, layer by layer, using the propagation rule.

$$o_j = f(\sum_i w_{ij} o_i + \theta_j) \tag{19}$$

where the output of unit j, o_j, is determined by the weighted sum of outputs of all units in the preceding layer, $\sum_i w_{ij} o_i$, and a bias term, θ_j, applied on an activation function f. An example of f suggested in [MRG86] is the sigmoid function,

$$f = \frac{1}{1 + e^{-x}}. \tag{20}$$

However, any choice of continuous, non-decreasing, differential function will be appropriate. The decision of the system is measured at the output layer. For the purpose of pattern recognition, it is common to assign an input pattern to the class whose corresponding output unit has the highest value among all output units.

The decision surface for a BP is formed during training phase by a series of weight adjustments. These weight adjustments are determined by the error signals transmitted in the backward direction. As each labeled pattern is fed in from the input layer and propagated to the output layer, values of the output units are compared to the desired output responses. The amount of error attributed by each unit, δ_j, is calculated, layer by layer, from output to input layer. For output units,

$$\delta_j = (t_j - o_j) o_j (1 - o_j), \tag{21}$$

where t_j is the ideal output response. For hidden units,

$$\delta_j = o_j (1 - o_j) \sum_k w_{jk} \delta_k, \tag{22}$$

where δ_k is the error contributed by each unit in the layer immediately above. After δ is calculated for each unit, every weight is adjusted by

$$\Delta w_{ij(t)} = \eta \delta_j o_j + \alpha \Delta w_{ij(t-1)}, \tag{23}$$

where $\Delta w_{ij(t)}$ is the weight change at iteration t; η, the learning rate and, α, the momentum, are scaling factors. The choice of these two parameters has great effect on the convergence rate of the system. In experiments described in Section 5, we used a learning rate of 0.75 and a momentum of 0.5. It has also been observed that BP performance is influenced by the number of hidden units available [KH88, GWG89]. For digit recognition we chose 15 hidden units, and for alphabet recognition we used 70 hidden units.

4 Relationship

We will first derive a theoretical relationship between the Bayes decision rule and backward error propagation.

The Bayes decision rule performance is optimal for a given set of features in the sense that it minimizes the probability of error and the conditional risk. As stated, this requires complete knowledge of the underlying probability density function for each class. However, in practice, finite training samples and high dimensionality compound this simple decision theory. The difficulty in actuating the class-conditional probability often makes the performance less than optimal.

Unlike the empirical approach in the 60's, the resurgence of neural networks has been led off by a series of theoretical analysis on their capabilities. Many important works have demonstrated the universal mapping ability of a backpropagation model under various constraints [HN89, Ara89, IM88, SW89, Shv90]

In order to understand the theory behind backpropagation, we need to study the derivation of generalized delta rule, which originates from minimizing the squared error sum between network and desired output responses over all patterns.

$$E = \sum_{\mathbf{x}} E_{\mathbf{x}} = \sum_{\mathbf{x}} \frac{1}{2} \sum_{j} (t_{\mathbf{x}j} - o_{\mathbf{x}j})^2 \tag{24}$$

where $t_{\mathbf{x}j}$ and $o_{\mathbf{x}j}$ are, respectively, the desired and actual response of jth output unit to pattern $\mathbf{x}$ To minimize E with respect to each weight w_{ij}, it is necessary to find the root for its partial derivative,

$$\frac{\partial E}{\partial w_{ij}} = \sum_{\mathbf{x}} \frac{\partial E_{\mathbf{x}}}{\partial w_{ij}}. \tag{25}$$

Thus it is sufficient to minimize $E_{\mathbf{x}}$. Gradient descent is the standard technique for solving such problems An approximation of gradient descent is achieved by making weight changes proportional to $E_{\mathbf{x}}$ after each presentation, i.e.,

$$\Delta_{\mathbf{x}} w_{ij} \propto -\frac{\partial E_{\mathbf{x}}}{\partial w_{ij}}.$$

The detailed derivation can be found in [MRG86]. For our purpose, it suffices to state the result:

$$\Delta_{\mathbf{x}} w_{ji} = \eta \delta_{\mathbf{x}j} o_{\mathbf{x}i}, \tag{26}$$

where

$$\delta_{\mathbf{x}j} = (t_{\mathbf{x}j} - o_{\mathbf{x}j}) f'(\sum_{i} w_{ij} o_{\mathbf{x}i} + \theta_j) \tag{27}$$

for output units and

$$\delta_{xj} = f'(\sum_i w_{ij} o_{xi} + \theta_j) \sum_k \delta_{xk} w_{kj} \tag{28}$$

for hidden units. In Equation 28 and 29, f' denotes the first derivative of f. When f is the sigmoid function in Equation 20, it can be shown that

$$f'(x) = f(x)(1 - f(x)) \tag{29}$$

and we obtain the formula in Equation 21 and 22.

We should point out that backpropagation performs gradient descent in E. However, by adjusting the weights after each pattern is presented, we deviate from true gradient descent. Nevertheless, Rumelhart noted that by selecting sufficiently small learning rate, η, a good approximation of gradient descent can be obtained through sequences of small movements[MRG86]

The statistical significance of minimizing squared error can be shown in the following analysis. First, we rewrite the Bayes decision rule for notational convenience

$$P(C_j|\mathrm{x}) = \frac{P(\mathrm{x}|C_j)P(C_j)}{\sum_{i=1}^{s} P(\mathrm{x}|C_{i\neq j})P(C_{i\neq j})} = \frac{P(\mathrm{x}, C_j)}{P(\mathrm{x})}$$

Consider a single output unit $f_j(\mathrm{x}, \mathrm{w})$ and random classification variable t_j, which is 1 if $\mathrm{x} \in C_j$ and 0 otherwise.

$$t_j(\mathrm{x}) = \begin{cases} 1 & \text{if } \mathrm{x} \in C_j \\ 0 & \text{if } \mathrm{x} \notin C_j \end{cases} \tag{30}$$

Then the criterion function J_s is defined.

$$\begin{aligned} J_s(\mathrm{w}) &= \sum_{\mathrm{x}} [f_j(\mathrm{x}, \mathrm{w}) - t_j]^2 \\ &= \sum_{\mathrm{x} \in C_j} [f_j(\mathrm{x}, \mathrm{w}) - 1]^2 + \sum_{\mathrm{x} \notin C_j} [f_j(\mathrm{x}, \mathrm{w}) - 0]^2 \\ &= n[\frac{n_j}{n} \frac{1}{n_j} \sum_{\mathrm{x} \in C_j} [f_j(\mathrm{x}, \mathrm{w}) - 1]^2 + \frac{n - n_j}{n} \frac{1}{n - n_j} \sum_{\mathrm{x} \notin C_j} f_j(\mathrm{x}, \mathrm{w})^2] \end{aligned}$$

where n is the total number of training samples, and n_j is the number of samples for class C_j.

Applying the law of large numbers and Bayes rule, we obtain the following result

$$\begin{aligned} \lim_{n \to \infty} \frac{1}{n} J_s(\mathrm{w}) &= J(\mathrm{x}, \mathrm{w}) \\ &= P(C_j) \int [f_j(\mathrm{x}, \mathrm{w}) - 1]^2 P(\mathrm{x}|C_j) d\mathrm{x} + P(C_{i\neq j}) \int f_j(\mathrm{x}, \mathrm{w})^2 P(\mathrm{x}|C_{i\neq j}) d\mathrm{x} \\ &= \int [f_j(\mathrm{x}, \mathrm{w}) - 1]^2 P(\mathrm{x}, C_j) d\mathrm{x} + \int f_j(\mathrm{x}, \mathrm{w})^2 P(\mathrm{x}, C_{i\neq j}) d\mathrm{x} \end{aligned}$$

$$= \int f_j(\mathrm{x},\mathrm{w})^2 P(\mathrm{x})d\mathrm{x} - 2\int f_j(\mathrm{x},\mathrm{w})P(\mathrm{x},C_j)d\mathrm{x} + \int P(\mathrm{x},C_j)d\mathrm{x}$$
$$= \int f_j(\mathrm{x},\mathrm{w})^2 P(\mathrm{x})d\mathrm{x} - 2\int f_j(\mathrm{x},\mathrm{w})P(C_j|\mathrm{x})P(\mathrm{x})d\mathrm{x} + \int P(C_j|\mathrm{x})P(\mathrm{x})d\mathrm{x}$$
$$= \int [f_j(\mathrm{x},\mathrm{w}) - P(C_j|\mathrm{x})]^2 P(\mathrm{x})d\mathrm{x} + \int P(C_j|\mathrm{x})P(C_{i\neq j}|\mathrm{x})P(\mathrm{x})d\mathrm{x}$$

In the last equation, since the second term on the right hand side is independent of w, whatever minimizes J_s/n also minimizes the first term. Therefore, backpropagation learning seeks least mean square approximation of the posterior density function in weight space. This is an extension of the analysis of two class linear classifiers given in [DH73].

Knowing that the output of a backpropagation network approximates the posterior density function, it is reasonable to ask whether a minimum probability of error solution exists. In [HH90] it has been shown that when the probability density function is Gaussian, a perceptron with a sigmoid transfer function approximates the a posteriori probabilities. We will extend this result to multiclass problems without restricting the form of underlying density functions

Using the criterion function in our previous analysis, we can minimize the total error by finding the root to the first partial derivative of $J(\mathrm{w},\mathrm{x})$ with respect to w Therefore,

$$\frac{\partial J}{\partial \mathrm{w}} = 2P(C_j)\int_{\mathrm{x}}(o_j-1)\frac{\partial f_j(\mathrm{x},\mathrm{w})}{\partial \mathrm{w}}P(\mathrm{x}|C_j)d\mathrm{x} + 2(1-P(C_j))\int_{\mathrm{x}} o_j\frac{\partial f_j(\mathrm{x},\mathrm{w})}{\partial \mathrm{w}}P(\mathrm{x}|C_{i\neq j})d\mathrm{x} = 0 \quad (31)$$

All possible w which achieves local minimum must satisfy this condition. However, any solution which satisfy this requirement may not be a minimum in E space. In fact, it may well be a local maximum or saddle point. To ensure the local minimum property, we need to exam the sign of the second derivative. The solution is a minimum if and only if

$$\frac{\partial^2 J}{\partial \mathrm{w}^2} > 0$$

Nevertheless, we will proceed to analyze its most obvious solution. This condition can be satisfied when

$$P(C_j)(1 - f_j(\mathrm{x},\mathrm{w}))\frac{\partial f_j(\mathrm{x},\mathrm{w})}{\partial \mathrm{w}}P(\mathrm{x}|C_j) = (1 - P(C_j))f_j(\mathrm{x},\mathrm{w})\frac{\partial f_j(\mathrm{x},\mathrm{w})}{\partial \mathrm{w}}P(\mathrm{x}|C_{i\neq j})$$

for all x. Under the assumption that $\partial f_j(\mathrm{x},\mathrm{w})/\partial \mathrm{w}$ is integrable and nonzero everywhere, the above equation can be simplified to

$$(1 - f_j(\mathrm{x},\mathrm{w}))\frac{P(C_j)P(\mathrm{x}|C_j)}{P(\mathrm{x})} = f_j(\mathrm{x},\mathrm{w})\frac{P(C_{i\neq j})P(\mathrm{x}|C_{i\neq j})}{P(\mathrm{x})}$$

The multiplicative constant $P(\mathbf{x})$, of course, has to be nonzero. By applying Bayes rule (Equation 1), we obtain

$$(1 - f_j(\mathbf{x}, \mathbf{w}))P(C_j|\mathbf{x}) = f_j(\mathbf{x}, \mathbf{w})P(C_{i\neq j}|\mathbf{x})$$

Thus,

$$P(C_j|\mathbf{x}) = f_j(\mathbf{x}, \mathbf{w})P(C_{i\neq j}|\mathbf{x}) + f_j(\mathbf{x}, \mathbf{w})P(C_j|\mathbf{x})$$

Since

$$P(C_j|\mathbf{x}) + P(C_{i\neq j}|\mathbf{x}) = 1$$

the following conclusion is reached :

$$f_j(\mathbf{x}, \mathbf{w}) = P(C_j|\mathbf{x}) \tag{32}$$

Not surprisingly, one solution is the posterior density function. The implicit assumption being made in the above analysis is that f is functionally capable of representing $P(C_j|\mathbf{x})$. A three layer BP model with infinite number of hidden units is necessary and sufficient to approximate any posterior density function to any degree of accuracy [HN89, Ara89, SW89]. The diagnosis of a more realistic feedforward network with bounded fan in can be found in [Shv90]. Without any prior knowledge of forms of the density functions, using a two layer network (no hidden layers) or a linear activation function will severely limit the representation power of our classifier.

From the above analysis, we make the following conclusion. Backpropagation performs least mean square approximation to the Bayes decision rule. A minimum probability of error solution indeed exists if the network is computationally capable of representing the a posteriori probability function exactly. A three layer backpropagation model (1 hidden layer) with sufficient number of hidden units has been shown to possess universal mapping ability and, therefore, can approximate any (a posteriori density) function to any degree of accuracy [HN89, SW89, Ara89].

However, the gradient descent technique used in backpropagation learning rule does not guarantee convergence to this solution. Networks can get trapped in a local minimum. Without an exact representation of the posterior function, this mean-square-error approximation may not give us the optimal classifier.

In this section we have shown that a multilayer feed forward network is functionally equivalent to approximating the a posteriori probability using gradient descent.

5 Experimental Results

Several experiments were conducted on handwritten digits and alphabets using the first order Bayes approximation and backward error propagation. Original images were di-

Table I:
95% confidence interval for digit recognition correct rate with 212 PPI images (manual segmentation)

	low	estimation	high
First-order Bayes (pixel)	91.4	92.2	92.9
Backprop (pixel)	94.9	95.5	96.0
First-order Bayes (feature)	97.5	97.9	98.3
Backprop (feature)	98.1	98.5	98.8

rectly obtained from handwritten addresses on mail samples. The specification of data and size of training and testing sets will be described in each section.

Input data for those pattern recognition algorithms primarily consisted of two formats. The first type of input, which we will refer to as *pixel* input, was binarized, size normalized images. Binarization was achieved by global thresholding, and moment normalization [Cas70] was used to scale all images to size 16x16.

The second type, which we will call *feature* input, was uniform length feature vectors of 636 binary components obtained by various feature extractors described in [LLS90]. Three types of feature extractors were used in constructing the feature vector : structural, contour analysis and feature templates. The structural approach was based on the work of [Pav86]. Several parameters and thresholds were adjusted to adapt the variations in character styles. The contour analysis based method used chaincodes to extract features such as holes and concavities. The third extractor implemented feature templates [BGJH88]. Detected features are represented in form of a fixed lengthed binary vector.

5.1 Digit Recognition

Two different experiments were done on handwritten digit recognition. Data from the first experiment consists of binary images that were scanned by OCR machines at 212 ppi (pixel per inch). These digits were acquired by manually locating the zipcodes on an envelope, and then segmenting them by a program. Binarization and moment normalization were applied on the images to obtain 16x16 bitmaps. The data has been cleaned up to eliminate any badly segmented images. The training set used to estimate parameters (section 2.1.2) contained 10,000 images, and the test set was a disjoint set of 5,000 images. Table I shows the 95% confidence interval for the correct rate [Hig62, CP34].

The images used for the second experiment were scanned at 300 ppi with 256 different gray scale value, and then thresholded to binary and moment normalized. Segmentation of digits were performed by a program. The training set, which was composed of 18,650 images, was cleaned up. Whereas the 2711 images in the novel testing set contains an estimated 3% of improperly segmented images. The performance is shown in Table II. Not surprisingly, the performance is not as good as that in the first experiment. Most of the errors were caused by poor segmentations.

5.2 Alphabet Recognition

The alphabet recognition experiment involves classification of all handprinted upper and lower case alphabets. However, since each individual character is presented without the baseline information, the following classes are combined : C-c, K-k, O-o, P-p, S-s, U-u, V-v, W-w, X-x, Z-z. After preliminary experiments, it was observed that much of the substitution error was caused by badly formed characters in class I-i-l, Y-y. Therefore, those classes are also combined. Thus we have a total of 41 classes.

All images in the training and testing set were scanned at 300ppi with 256 gray scale values and then thresholded and moment normalized to 16x16 binary images. Segmentation was performed on-line manually The training set consists of 8000 images, unevenly distributed among classes. The test set contained 2865 previously unseen images. Experimental results are summarized in Table III.

Samples of images used in both the digit and alphabet recognition problems are given in the appendix. These images were extracted from addresses on envelopes.

6 Discussion

From the analysis in section 4, we have seen for a specific target function, outputs of a backpropagation network approximate the a posteriori probabilities. Therefore, by assigning the input pattern to the class whose corresponding output unit has the highest value, we effectively maximize the (estimated) posterior probability, and minimize the probability of error

In practice, however, the comparison between Bayesian classifier and backpropagation is complicated by various factors. In most real-world problems, high-dimensionality and finite training samples inhibit us from making accurate calculation of probability density functions. Working reluctantly under an assumption of statistical independence or with low order approximations, we expectedly obtain suboptimal performance.

While backpropagation provides an automated solution to such problems, it also has its pitfalls. Using a gradient descent technique in a most likely non-parabolic error space, it is susceptible to local minima In fact, Hecht-Nielsen [HN89] has shown the existence of

Table II:
95% confidence interval for digit recognition correct rate with 300 PPI images (automatic segmentation)

	low	estimation	high
First-order Bayes (pixel)	84.9	86.2	87.4
Backprop (pixel)	89.1	90.3	91.4
First-order Bayes (feature)	92.7	93.7	94.6
Backprop (feature)	91.9	92.9	93.8

Table III:
95% confidence interval for alphabetic character recognition correct rate with 300 PPI images (manual segmentation)

	low	estimation	high
First-order Bayes (pixel)	68.9	70.6	72.2
Backprop (pixel)	72.0	73.7	75.3
First-order Bayes (feature)	89.0	90.1	91.1
Backprop (feature)	87.4	88.6	89.7

local minima in backpropagation error surface. Although optimization techniques have been proposed to find global minimum [Was88, Bab90], their effectiveness in complex problems remains to be seen. Furthermore, the classical backpropagation is vulnerable to other problems such as slow convergence and generalization. Many researchers have proposed methods to improve the speed of convergence [Fah88, IP90, ST90, Jac88] However, there is still some dispute in the definition of *convergence* itself. It has been observed that an *absolute convergence* may not provide the best generalization result. The effect of additional hidden units on generalization has not been thoroughly explored either. Under such conditions, its performance may also be degraded.

From the tables in section 5, we can make the following observations. The accuracies of Bayesian classifier and BP are comparable in most cases. BP perform better than the first order Bayes approximation with *pixel* input. This is not surprising considering the second classifier, being a first order approximation, is capable of achieving only linear decision surfaces. Contrarily, the activation function in a backpropagation network allows it to capture higher order relations

In the case of *feature vector* input, their relative performance is not clear. The results in digit and character recognition experiments showed similar accuracy rates for both classifiers. Although the first-order Bayesian approximation has out performed backpropagation network in two experiments, their confidence intervals overlap considerably. If the feature components are indeed statistically independent, the first-order Bayesian classifier provides optimal decision, and backpropagation network is albe to find good approximation to such decision. However, it is more likely that the features are not independent, and better local minima would have been found had the parameter space been explored more thoroughly.

We should also take into consideration that while the implementation of first order Bayes discriminant classifier is straight forward, the optimal configuration for a BP network can only be found with more research. Therefore, backpropagation has much more potential for improvement.

7 Conclusion

We have shown for a specific case that the outputs of a backpropagation network are direct estimations of a posteriori probabilities. Therefore, this neural network model has the same computational power as the Bayes decision rule. Empirical results obtained in real world handprinted characters recognition experiments showed that these two classifiers have comparable performances. A finer comparison would require a detailed analysis of the nature of input data. It is expected that if the input components are statistically independent, the Bayes classifier will be optimal. However, if any dependence exists among input components, backpropagation is likely to out-perform first order Bayesian classifier. Normally the limitation in computing time and space hinders higher order approximation of probability density functions. On the other hand, many techniques have been developed to mitigate the difficulties encounted in training a backpropagation. Under the circumstance, the neural network approach is a very promising alternative to Bayesian classification.

8 Acknowledgements

This work was supported in part by the United States Postal Service and by the Eastman Kodak Company.

References

[Ara89] M. Arai. Mapping abilities of three-layer neural networks. *Proc. of International Joint Conference on Neural Networks*, 1:419–423, 1989.

[Bab90] N. Baba. A hybrid algorithm for finding the global minimum of error function of neural networks. *Proc. of International Joint Conference on Neural Networks*, 1:585–588, 1990.

[BGJH88] H Baird, H P. Graf, L D. Jackel, and W. E. Hubbard. A vlsi architecture for binary image classification. In *Proceedings of Cost 13 Workship : From the Pixels to the Features*, Bonas, France, 1988.

[Bur87] D. J. Burr Experiments with a connectionist text reader. *Proc. of International Conference on Neural Networks*, 4:717–724, 1987.

[Cas70] R G. Casey. Moment normalization of handprinted characters. *IBM Journal of Research and Development*, Sept 1970.

[CP34] C.S. Clopper and E. S Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26, 1934

[DH73] R. O Duda and P E Hart *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973.

[DJG$^+$89] J. Denker, L. Jackel, W. R. Gardner, H. P. Graf, D Henderson, R. E Howard, W. Hubbard, H S. Baird, and I. Guyon. Neural network recognizer for handwritten zip code digits. In *Advances in Neural Information Processing Systems*, volume 1. Morgan Kaufmann, 1989.

[Fah88] S. E. Fahlman. An empirical study of learning speed in back-propagation networks Technical Report CMU-CS-88-162, CMU, 1988.

[Fuk72] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, New York, 1972.

[Fuk88] K Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition *Neural Networks*, 1:119–130, 1988.

[GWG89] M Gutierrez, J. Wang, and R. Grondin. Estimating hidden unit number for two-layer perceptrons *Proc. of International Joint Conference on Neural Networks*, 1:677–681, 1989.

[HH90] B Horne and D Hush. On the optimality of the sigmoid perceptron. *Proc. of International Joint Conference on Neural Networks*, 1:269–272, 1990.

[Hig62] W. H. Highleyman. The design and analysis of pattern recognition experiments. *Bell System Technical Journal*, 41:723–744, March 1962.

[HN89] R. Hecht-Nielsen Theory of the backpropagation neural network *Proc. of International Joint Conference on Neural Networks*, 1:593–605, 1989.

[IM88] B. Irie and S Miyake. Capabilities of three-layered perceptrons *Proc. of International Conference on Neural Networks*, 1:641–648, 1988.

[IP90] Y. Izui and A Pentland. Speeding up back propagation. *Proc. of International Joint Conference on Neural Networks*, 1:639–642, 1990.

[Jac88] R. A. Jacobs. Increased rates of convergence through learning rate adaption. *Neural Networks*, 1:295–307, 1988.

[KH88] S. Y. Kung and J. N. Hwang. An algebraic projection analysis for optimal hidden units size and learning rates in backpropagation learning. *Proc. of International Joint Conference on Neural Networks*, 1:363–370, 1988.

[Lip87] R. P. Lippman. An introduction to computing with neural nets. *IEEE ASSP*, 1987.

[LLS90] D S. Lee, S. W Lam, and S. N. Srihari. A structural approach to recognize handprinted and degraded machine-printed characters. In *Proceedings of IAPR Syntactic and Structural Pattern Recognition Workshop*, Murray Hill, New Jersey, 1990

[MP69] M Minsky and S. Papert *Perceptrons*. MIT Press, 1969.

[MRG86] J L. McClelland, D E. Rumelhart, and PDP Research Group. *Parallel Distributed Processing*. MIT Press, 1986.

[Par85] D. Parker Learning-logic. Technical Report TR-47, MIT, 1985.

[Pav86] T Pavlidis. A vectorizer and feature extractor for document recognition. *Computer Vision, Graphics, and Image Processing*, 35:111–127, 1986.

[RMS89] A. Rajavelu, M T. Musavi, and M. V. Shirvaikar. A neural network approach to character recognition. *Neural Networks*, 2:387–393, 1989.

[Ros57] F. Rosenblatt. The perceptron : A perceiving and recognizing automaton. *Cornell Aeronaut. Lab Report*, 85-4601-1, 1957.

[Shv90] H Shvaytser. On the learning power of networks with a bounded fan-in layer. *Proc of International Joint Conference on Neural Networks*, 1:313–316, 1990.

[ST90] R. S. Scalero and N. Tepedelenlioglu. A fast training algorithm for neural networks *Proc of International Joint Conference on Neural Networks*, 1:715–718, 1990

[SW81] J. Sklansky and G N. Wassel. *Pattern Classifiers and Trainable Machines*. Springer-Verlag, New York, 1981.

[SW89] M. Stinchcombe and H. White Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions. *Proc of International Joint Conference on Neural Networks*, 1:613–617, 1989.

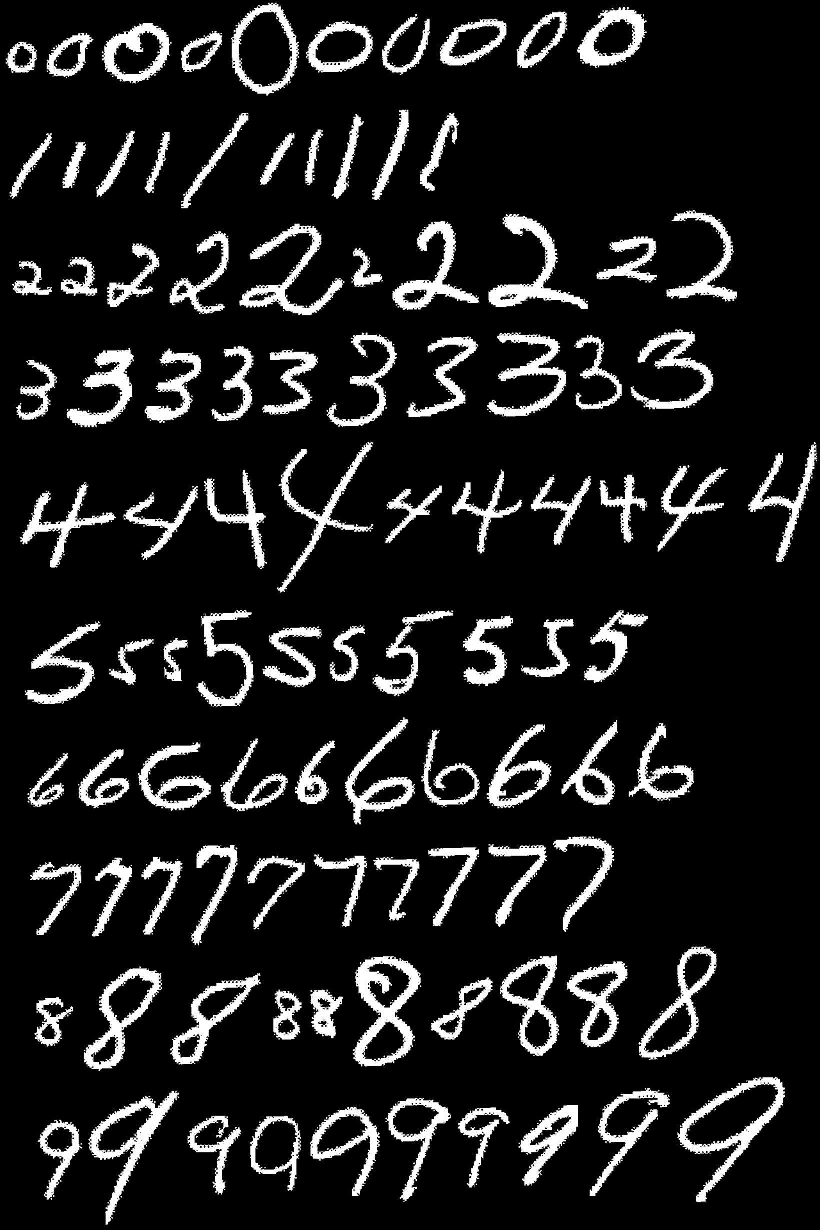
[TG74] J. T. Tou and R C. Gonzalez. *Pattern Recognition Principles*. Addison-Wesley, 1974

[Was88] P. D. Wasserman Combined backpropagation/cauchy machine. In *Proceedings of International Neural Network Society*, New York, 1988. Pergamon Press.

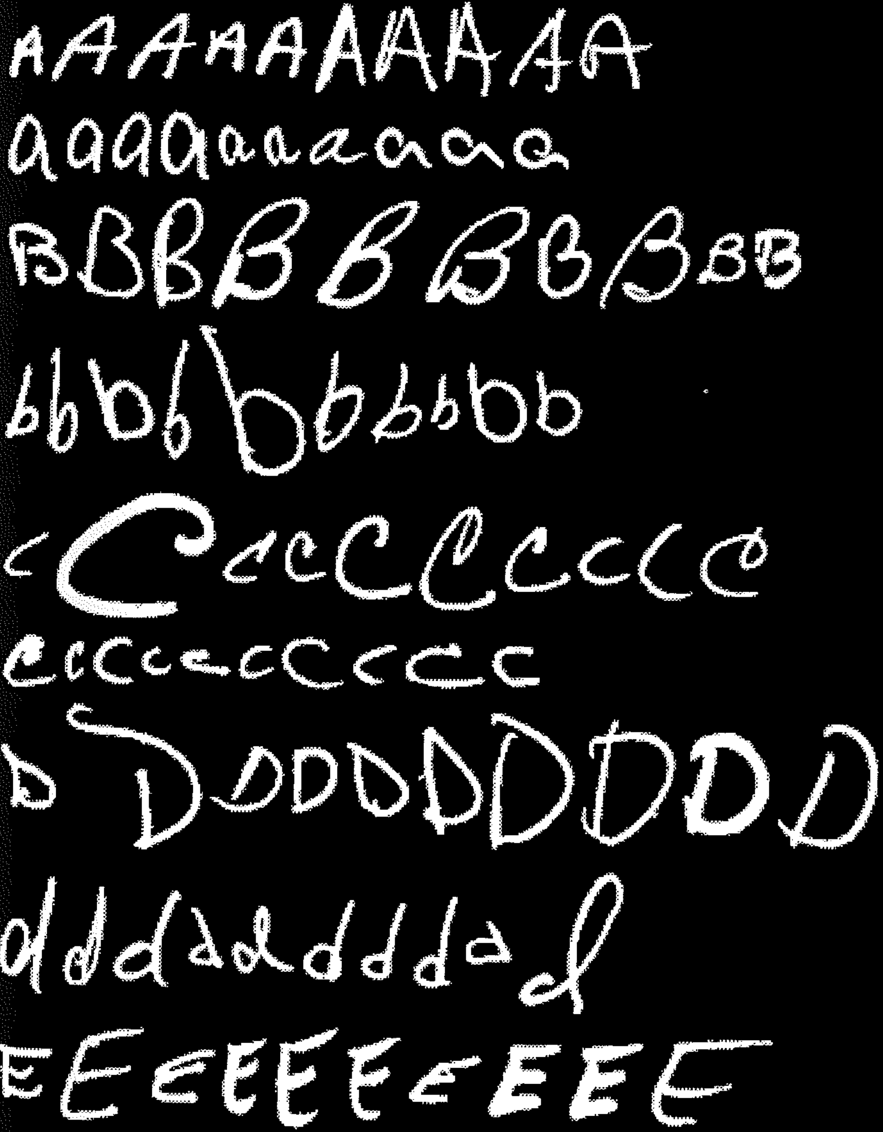
[Wer74] P Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

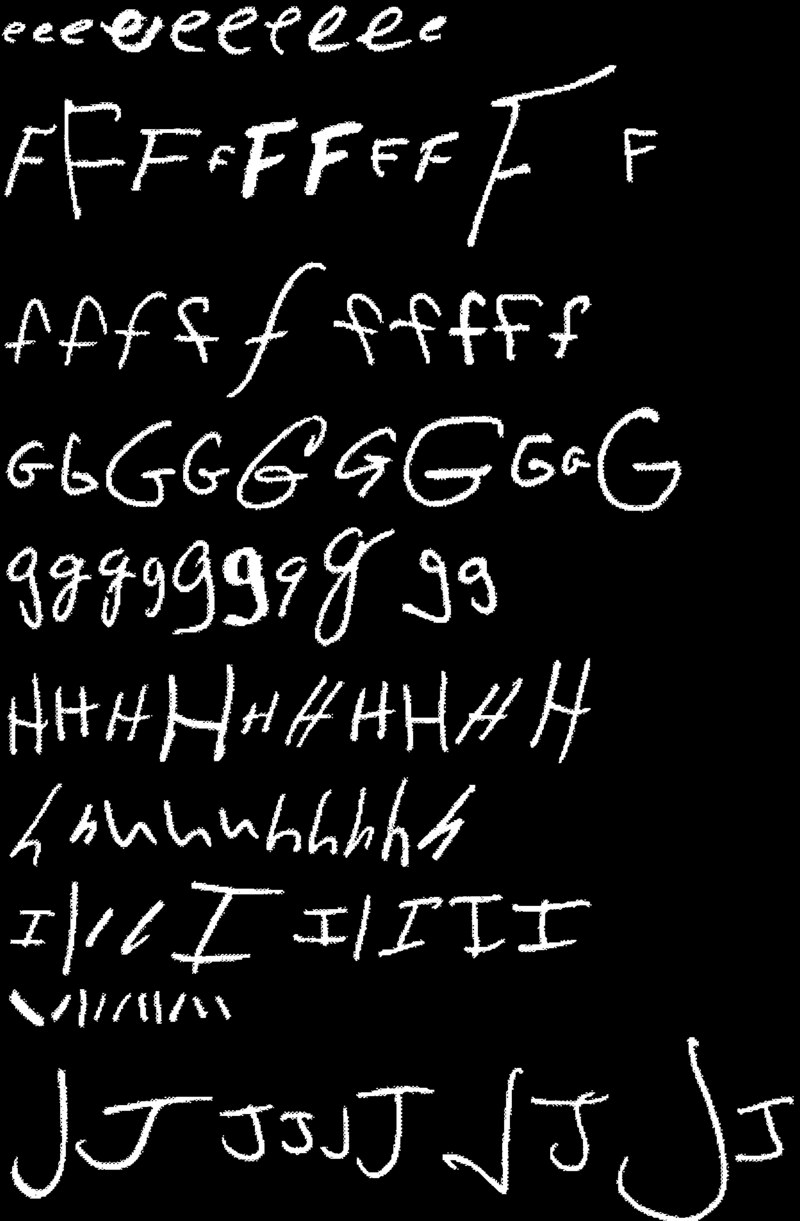
[WK89] S. M. Weiss and I. Kapouleas. An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. *Proc. of Eleventh International Joint Conference on Artificial Intelligence*, 1:781–787, 1989.

Appendix : Sample images used in experimentation.
Handwritten Digits (10 classes)



Alphabets (52 classes)




eeeeeeeeee
FFFFFFFFFFF
ffffffffff
GGGGGGGGGGG
gggggggggg
HHHHHHHHHHH
hhhhhhhhhh
IIIIIIIIIII
iiiiiiiiii
JJJJJJJJJJ

Artificial Neural Networks and Statistical Pattern Recognition
Old and New Connections
I.K. Sethi and A K Jain (Editors)
© 1991 Elsevier Science Publishers B.V All rights reserved

Shape and Texture Recognition by a Neural Network

Alireza Khotanzad [a] and Jiin-Her Lu [b]

[a]Image Processing and Analysis Laboratory, Electrical Engineering Department, Southern Methodist University, Dallas, Texas 75275, U.S.A.

[b]Image Recognition Equipment Corp., 1198 Commerce Dr., Richardson, TX 75081, U.S.A.

1. INTRODUCTION

Pattern recognition is an essential part of any high level computer vision system. Such systems are now in use in many diverse fields, among them robotics, military reconnaissance, remote sensing, document processing, and industrial automation. Recent developments in the field of artificial neural networks (ANN) have provided potential alternatives to the traditional techniques of pattern recognition. An ANN is composed of many simple nonlinear computational elements operating in parallel and arranged in patterns reminiscent of biological nervous systems. These nonlinear elements which are the building blocks of the network play the same role as the neurons in the brain and thus are usually called "neurons" or "nodes". The nodes are interconnected via weights that can be adapted and changed according to a given situation, analogous to synaptic connections to neurons in the brain. It should be noted that while ANNs are inspired from studies of biological systems, they are far from mimicking functions of the human brain. The view taken in this study is that an ANN is a self-consistent model and does not require the correctness of neural theory to validate its operation. It is only inspired by the tremendous potential, highly parallel operation, and fault tolerant nature of the brain, and is not constrained by the exact details.

ANNs are capable of many functions, among them optimization, clustering, mapping, and classification. In this study, the net is utilized in the context of a supervised classifier. This is a decision making process which requires the net to identify the class or category which best represents an input pattern. It is assumed that the net has already adapted the classes it is expected to recognize through a learning process using labeled training prototypes from each category. There are many traditional techniques for solution of this problem. One of the contributions of this chapter is to show the advantages of an ANN classifier over some of the conventional recognition algorithms.

In this chapter, two specific image recognition problems namely, classification of two-dimensional shapes and texture recognition are addressed. The focus of the study is on images containing a single shape or a single kind of texture. Images containing several objects or those composed of several regions with different texture can first be divided into several single shape/texture images. Description of several image and texture segmentation techniques could be found in [6], [12], and [13].

The key step in any classification problem is to represent an image with a set of numerical features (a feature vector) whose dimension is much lower than the original image data. This removes redundancy from the data and drastically cuts the computational cost of the classification stage. In a recognition application, the most important criterion for the extracted features is that they must retain much of the useful discriminability information present in the original data. Selection of "good" features is a crucial step in the process since the next stage only sees these features and acts upon them. "Good" features are those satisfying two requirements. The first one is small intra-class invariance - slightly different shapes with similar general characteristics should have numerically close values. The second one is inter-class separation - features from different classes should be quite different numerically. Additionally, a flexible recognition system must be insensitive to parameters such as orientation of the object/texture, and scale and location of the object in the field of view. If the features that are extracted from the image are invariant to such parameters, the classifier is relieved from the difficult task of handling these variations. This is the approach taken in this study. In the case of shape recognition, rotation, scale, and translation invariant features recently developed by the authors [14],[15], [17] are utilized. They are based on a set of complex moments of image termed as "Zernike Moments". For the texture classification problem, random-field model based rotation invariant features also developed by one of the authors [10] are used to characterize the image.

The neural network classifier used in this study is a multi-layer feed-forward ANN which is typically called a "Multi-Layer Perceptron" (MLP). The input to this net consists of the features extracted from the image. It produces the class label of the input at its output. Many aspects of the performance of this ANN are experimentally studied. They include: (1) comparison with performances of two other conventional classifiers namely the minimum-mean-distance, and the nearest-neighbor, (2) effect of changes in the net parameters, (3) noise tolerance, and (4) fault tolerance. These studies are carried out using two binary shape databases consisting of a 26-class data set of English characters and a 10-class data set of handwritten digits. Texture recognition is studied using a 12-class gray level database of natural textures.

The organization of this chapter is as follows. Section 2 discusses the Zernike moment-based shape features and their invariance properties. Random field-based texture features are considered in Section 3. In Section 4 the MLP classifier and its learning rule are described. The two conventional classifiers are discussed in Section 5. Section 6 reports the experimental results on the shape databases. Texture classification results are presented in Section 7. Some discussions and concluding remarks are presented in Section 8.

2. ZERNIKE MOMENT FEATURES FOR SHAPE RECOGNITION

In [22], Zernike introduced a set of complex polynomials which form a complete orthogonal set over the interior of the unit circle, i.e. $x^2+y^2=1$. Let the set of these polynomials be denoted by $\{ V_{nm}(x,y) \}$. The form of these polynomials is :

$$V_{nm}(x,y) = V_{nm}(\rho,\theta) = R_{nm}(\rho)\exp(jm\theta) \qquad (1)$$

where
n: positive integer or zero
m: positive and negative integers subject to constraints $n - |m|$ even, $|m| \leq n$
ρ: length of vector from origin to (x,y) pixel
θ: angle between vector ρ and x axis in counterclockwise direction
$R_{nm}(\rho)$: radial polynomial defined as

$$R_{nm}(\rho) = \sum_{s=0}^{\frac{n-|m|}{2}} \frac{(-1)^s [(n-s)!]\, \rho^{n-2s}}{s!\,(\frac{n+|m|}{2}-s)!\,(\frac{n-|m|}{2}-s)!}.$$

Note that $R_{n,-m}(\rho) = R_{nm}(\rho)$.
These polynomials are orthogonal and satisfy

$$\iint_{x^2+y^2\leq 1} [V^*_{nm}(x,y)]\, V_{pq}(x,y)\, dxdy = \frac{\pi}{n+1}\delta_{np}\delta_{mq} \qquad (2)$$

with

$$\delta_{ab} = \begin{cases} 1 & a=b \\ 0 & \text{otherwise} \end{cases}$$

Zernike moments are the projection of the image function onto these orthogonal basis functions. The Zernike moment of order n with repetition m for a digital image, f(x,y), that vanishes outside the unit circle is

$$A_{nm} = \frac{n+1}{\pi}\sum_x\sum_y f(x,y)V^*_{nm}(\rho,\theta), \qquad x^2+y^2\leq 1 \qquad (3)$$

To compute the Zernike moments of a given image, the center of the image is taken as the origin and pixel coordinates are mapped to the range of unit circle, i.e. $x^2+y^2\leq 1$. Those pixels falling outside the unit circle are not used in the computation. Also note that $A^*_{nm} = A_{n,-m}$.

The features defined on Zernike moments are derived by using rotational properties of these moments. Consider a rotation of the image through angle θ. The relationship between A'_{nm} and A_{nm}, the Zernike moment of the rotated image and the unrotated one is [20]

$$A'_{nm} = A_{nm}\exp(-jm\theta) \tag{4}$$

This relation shows that Zernike moments have simple rotational transformation properties; each Zernike moment merely acquires a phase shift on rotation. This simple property leads to the conclusion that the magnitudes of the Zernike moments of a rotated image function remain identical to those before rotation. Thus $|A_{nm}|$, the magnitude of the Zernike moment, can be taken as a rotation invariant feature of the underlying image function. Note that since $A_{n,-m} = A^*_{nm}$, then $|A_{nm}| = |A_{n,-m}|$, thus one can concentrate on $|A_{nm}|$ with $m \geq 0$ as far as the defined Zernike features are concerned. Table 1 lists the rotation invariant Zernike features and their corresponding numbers from order zero to order twelve.

Table 1
Order zero to order twelve Zernike moments whose magnitudes can be used as features.

Order	Moments	No. of Moments
0	A_{00}	1
1	A_{11}	1
2	A_{20}, A_{02}	2
3	A_{31}, A_{33}	2
4	A_{40}, A_{42}, A_{44}	3
5	A_{51}, A_{53}, A_{55}	3
6	A_{60}, A_{62}, A_{64}, A_{66}	4
7	A_{71}, A_{73}, A_{75}, A_{77}	4
8	A_{80}, A_{82}, A_{84}, A_{86}, A_{88}	5
9	A_{91}, A_{93}, A_{95}, A_{97}, A_{99}	5
10	$A_{10,0}$, $A_{10,2}$, $A_{10,4}$, $A_{10,6}$, $A_{10,8}$, $A_{10,10}$	6
11	$A_{11,1}$, $A_{11,3}$, $A_{11,5}$, $A_{11,7}$, $A_{11,9}$, $A_{11,11}$	6
12	$A_{12,0}$, $A_{12,2}$, $A_{12,4}$, $A_{12,6}$, $A_{12,8}$, $A_{12,10}$, $A_{12,12}$	7

This rotation invariancy property is illustrated by an experiment. Fig. 1 shows a 64 × 64 binary image of character "A" and five rotated versions of it, with rotation angles of 30°, 60°, 150°, 180°, and 300°, respectively. Table 2 is the list of the magnitudes of their Zernike moments for orders two, and three, their respective sample mean, μ, sample standard deviation, σ, and σ/μ %, which indicates the percentage of spread of the $|A_{nm}|$ values from their corresponding means. It is observed that rotation invariancy is very well achieved since σ/μ % values are very small. The reason for not obtaining exact invariances (i.e. $\sigma/\mu = 0$ %) is that image function is digital rather than continuous.

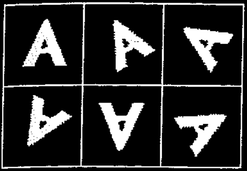


Figure 1. The images of character "A" and five rotated versions of it. From top left to right rotation angles are: 0°, 30°, 60°, 150°, 180°, and 300°.

Table 2
Magnitudes of some of the Zernike moments for rotated images shown in Fig. 1 and their corresponding statistics

Angle	$\lvert A_{20} \rvert$	$\lvert A_{22} \rvert$	$\lvert A_{31} \rvert$	$\lvert A_{33} \rvert$
0°	439 62	41 79	57 97	172 57
30°	436 70	40 20	63 82	171 96
60°	440 63	40 08	66 28	169 41
150°	438 53	41 55	65 47	170 83
180°	439 01	46 85	62 39	168 47
300°	438 43	39 19	65 77	170 84
μ	438 82	41 61	63 62	170 68
σ	1 32	2 74	3 12	1 53
σ/μ %	0 30	6 57	4 90	0 90

The proposed Zernike features are only rotation invariant. To achieve scale and translation invariancy, the image is first normalized with respect to these parameters. The Zernike features are then extracted from the scale and translation normalized images. The scale and translation normalization is carried out using the regular moments of the image. The (p+q)th regular moment of a unit circle mapped image is defined as:

$$m_{pq} = \sum_{x=-1}^{+1} \sum_{y=-1}^{+1} x^p \, y^q \, f(x,y) \tag{5}$$

Translation invariancy is achieved by moving the origin to the centroid of the shape. This means that the image is transformed into a new one whose first order moments m_{01} and m_{10} are both equal to zero. This is done by transforming original f(x,y) image into $f(x+\bar{x}, y+\bar{y})$ image where $\bar{x}$ and $\bar{y}$ are the centroid location of the original image computed as

$$\bar{x} = \frac{m_{10}}{m_{00}} \qquad \bar{y} = \frac{m_{01}}{m_{00}} \, . \tag{6}$$

Scale invariancy is accomplished by enlarging or reducing each shape such that its zeroth order moment, m_{00}, is set equal to a predetermined value, β. This task is achieved by transforming the original image function, f(x,y), into a new function

$$f(\frac{x}{a}, \frac{y}{a}),$$

with $a = (\frac{\beta}{m_{00}})^{\frac{1}{2}}$ [17].

In summary, an image function, f(x,y), can be normalized with respect to scale and translation by transforming it into g(x,y), where

$$g(x,y) = f\left(\bar{x}+\frac{x}{a}, \bar{y}+\frac{y}{a}\right), \qquad (7)$$

Fig. 2 shows six 64×64 scaled and translated images of character "A" along with their scaled and translation normalized versions using $\beta = 800$.

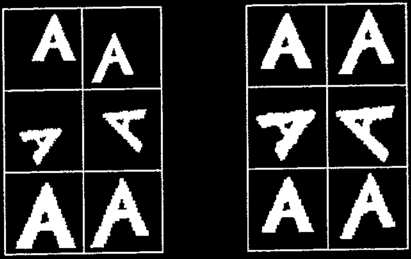


Figure 2. Six scaled and translated images of character "A" along with their scaled and translation normalized versions using β=800.

The scale and translation normalization process affects two of the Zernike features, $|A_{00}|$ and $|A_{11}|$. $|A_{00}| = \frac{\beta}{\pi}$ and $|A_{11}| = 0$ for all the normalized images [14]. Therefore, these two moments are not included in the set of features utilized in the classification experiments and only those of second order and higher are considered.

3. RANDOM FIELD FEATURES FOR TEXTURE RECOGNITION

The textural features introduced in this section are invariant under rotation or gray scale changes. Two types of noncausal random field models are used to characterize the spatial interactions among neighboring pixels which give rise to the notion of perceived texture. The first kind of models is called Circularly-Symmetric-Autoregressive (CSAR) [10]. Let $\{g(x,y);\ x, y = 0, 1, ..., M\text{-}1\}$ be gray-level values of a discrete M×M image with the top left of the image at (x=0, y=0) and the bottom right at (x=M-1, y=M-1) and with x and y axes representing vertical and horizontal directions, respectively. It is assumed that the sample mean of this image is zero. If g(x,y) obeys a CSAR model defined over a M×M toroidal lattice then;

$$g(x,y) = \phi \sum_{(i,j)\epsilon N_c} g(x \oplus i, y \oplus j) + v(x,y) \qquad (8)$$

where

$N_c =$ $\{(0,1), (0,-1), (1,0), (-1,0), (\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}), (-\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}),$ $(\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}), (-\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})\}$, i.e. a circular neighbor set containing eight symmetric coordinates as listed; N_c is shown in Fig. 3.

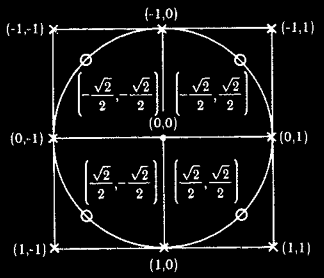

(-1,-1)
(-1,0)
(-1,1)
$\left[-\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}\right]$
$\left[-\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right]$
(0,0)
(0,-1)
(0,1)
$\left[\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}\right]$
$\left[\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right]$
(1,-1)
(1,0)
(1,1)

Figure 3. Relative spatial locations of the eight members of the circular neighbor set N_c.

$\phi =$ Coefficient of the model characterizing the dependence of g(x,y) to those in its N_c neighborhood.

$\oplus =$ addition modulo M. Acts like an ordinary addition except at those (x,y) points located on or near the edges of image for which a complete neighbor set cannot be found. In this situation, $\oplus$ operation causes a wrap around (torus lattice) effect and creates a full neighbor set for each pixel.

v(x,y) = A correlated sequence of identically distributed random variables with zero mean and variance of ν characterizing fluctuations and noise in the model.

Thus, each pixel intensity is modeled as a linear combination of eight other quantities plus a noise term. Four out of the eight are actual gray levels of the 4-connected neighbors of (x,y) pixel. The four other ones correspond to locations which do not fall on the grid corners of a digitized image. Each of these four quantities are computed as a linear combination of the gray level of its four nearest grid corner surrounding it. For instance, the value associated with $(x \oplus \frac{-\sqrt{2}}{2}, y \oplus \frac{\sqrt{2}}{2})$ coordinate is computed from g(x,y), g(x,y+1), g(x-1,y+1) and g(x-1,y). The interpolation scheme works as follows. Let (t_1, t_2) be one such coordinate. Then

$$g(t_1, t_2) = \left[\frac{1}{\sum_{i=1}^{4} d_i}\right]\left[\sum_{i=1}^{4} d_i\, g_i\right] \quad (9)$$

where g_i is the gray level of one of the four grid corners surrounding (t_1, t_2) and d_i is the inverse Euclidean distance between (t_1, t_2) and that corner.

Characterization of a texture by CSAR model requires that its parameters (ϕ, ν) be estimated. A least-squares estimation scheme developed in [10] is used to obtain these estimates denoted by $(\hat{\phi}, \hat{\nu})$. One can interpret $\hat{\phi}$ as the degree of isotropy of the texture and $\hat{\nu}$ as the degree of its roughness. A useful property associated with the CSAR parameters is their rotation invariance. Since they are obtained using a circularly symmetric neighborhood, they are insensitive to the orientation of the underlying texture. Fig. 4 shows seven differently oriented images of raffia texture. In the first two columns of Table 3 the estimated $(\hat{\phi}, \hat{\nu})$ parameters and their corresponding statistics are shown. These entries confirm the rotation invariance property.

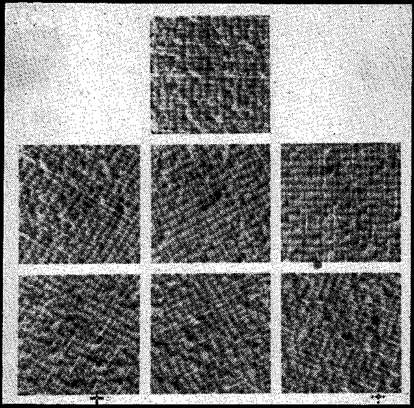


Figure 4. A 64×64 digitized sample of each of the seven orientations of raffia texture. From top and from left to right: 0°, 30°, 60°, 90°, 120°, 150°, and 200°.

Table 3
Statistics of texture features for differently oriented images of Fig. 4.

Angle	$\hat{\phi}$	$\hat{\nu}$	ξ
0°	.1344	.4757	.0751
30°	.1326	.4834	.0620
60°	.1326	.4733	.0737
90°	.1328	.4802	.0822
120°	.1329	.4796	.0744
150°	.1329	.4785	.0660
200°	.1339	.4755	.0758
μ	.1332	.4780	.0727
σ	.0007	.0084	.0023
$\sigma/\mu\%$	.5	1.8	3.16

In addition to these two features, a third one is developed. Neither $\hat{\phi}$ nor $\hat{\nu}$ could capture information about strong directionality that is present in some textures. This is the task of the third feature termed ξ. It is obtained by fitting a different random field model to the image. This model is called "Simultaneous Autoregressive (SAR)" [7], [8] and is expressed as

$$g(x,y) = \sum_{(i,j)\epsilon N} \theta_{(i,j)}\, g(x \oplus i, y \oplus j) + \omega(x,y) \tag{10}$$

where N is a neighbor set which includes only integer coordinates (i.e. grid corners) excluding (0,0), $\theta_{(i,j)} = \theta_{(-i,-j)}$, and $\{\omega(\cdot)\}$ is a sequence of i.i.d. Gaussian random variables with zero mean and variance of ρ_N. Thus, the SAR model relates the gray level of a pixel to those in its neighborhood, N, through θ parameters. The noise term added to this linear combination accounts for fluctuations throughout the image. The model parameters are $\{\theta_{(i,j)}\}$, and ρ_N. The maximum likelihood estimates of these parameters are computed by a method described in [8] and [16].

ξ is computed as a function of the estimated θ parameters of two different SAR models. The first one is a model which characterizes g(x,y) as a function of the gray-levels of its 4-connected neighbors. In other words, it uses $N_a = \{(0,1), (0,-1), (1,0), (-1,0)\}$. Let $\theta^*_{(0,1)}$ and $\theta^*_{(1,0)}$ be the ML estimates of the corresponding parameters of this model. The second model relates g(x,y) to those of its diagonal and off-diagonal neighbors. This is done using $N_b = \{(1,1), (1,-1), (-1,1), (-1,-1)\}$. Let $\theta^*_{(1,1)}$, and $\theta^*_{(1,-1)}{}^*$ be the ML estimates of this model. Then ξ is defined as:

$$\xi = \max[\, |\theta^*_{(1,0)} - \theta^*_{(0,1)}|,\ |\theta^*_{(1,1)} - \theta^*_{(1,-1)}|\,] \tag{11}$$

ξ can be interpreted as the maximum extent of variation of the underlying texture in the [horizontal, vertical] (captured by the first term) and [diagonal, off-diagonal] (captured by the second term) directions. Thus it is a measure of directionality. For highly directional textures ξ is large and vice versa.

ξ is also rotation invariant. The third column of Table 3 shows the computed values of ξ corresponding to seven differently oriented raffia images. These entries show little variation among ξ values.

To summarize, a textured image is represented by three features $\hat{\phi}$, $\hat{\nu}$, and ξ. The utilized classifiers process these features to carry out their decision making task.

4. MULTI-LAYER PERCEPTRON CLASSIFIER

In this study a neural network topology known as "Multi-Layer Perceptron" or MLP is used. A MLP is a feed-forward net with one or more layers of nodes between the input and output nodes. These in-between layers are called hidden layers. A MLP with one hidden layer is shown in Fig. 5. Connections within a layer or from higher to lower layers are not permitted. Each node in a layer is connected to all the nodes in the layer above it. Training is equivalent to finding proper weights for all the connections such that a desired output is generated for a corresponding input. Using MLP in the context of a classifier requires all output nodes to be set to zero except for the node that is marked to correspond to the class the input is from. That desired output is one. In this study, the inputs are either the Zernike moment

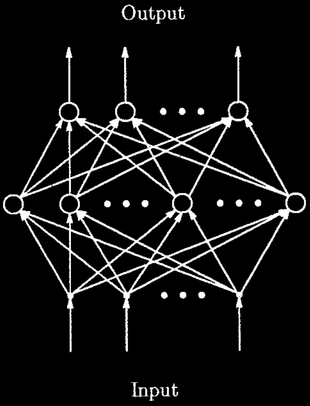


Figure 5. A Multi-Layer Perceptron (MLP) with one hidden layer.

features or the random field features extracted from the image to be classified.

MLPs were not used in the past because of the lack of effective training algorithms for them. This has recently changed due to development of an iterative gradient procedure known as "Back-Propagation Algorithm" [19]. According to this algorithm which is outlined next, for each pattern in the training set, learning proper weights is conducted by (1) computing the discrepancy between the desired and actual outputs, and (2) feeding back this error signal level-by-level to the inputs, changing the connection weights in such a way as to modify them in proportion to their responsibility for the output error. The major steps of the algorithm are as follows:

Step 1 Initialize all w_{ij}s to small random values. w_{ij} is the value of the connection weight between unit j and unit i in the layer below.

Step 2 Present an input from class m and specify the desired output. The desired output is 0 for all the output nodes except the mth node which is 1.

Step 3 Calculate actual outputs of all the nodes using the present value of w_{ij}s. The output of node j, denoted by y_j is a non-linear function of its total input:

$$y_j = \frac{1}{1 + \exp(-\sum_i y_i w_{ij})}. \tag{12}$$

This particular non-linear function is called a "Sigmoid" function.

Step 4 Find an error term, δ_j, for all the nodes. If d_j and y_j stand for desired and actual values of a node respectively, for an output node

$$\delta_j = (d_j - y_j)\, y_j\, (1 - y_j) \tag{13}$$

and for a hidden layer node

$$\delta_j = y_j(1 - y_j) \sum_k \delta_k w_{jk} \tag{14}$$

where k is over all nodes in the layer above node j.

Step 5 Adjust weights by

$$w_{ij}(n+1) = w_{ij}(n) + \alpha\, \delta_j\, y_i + \gamma\, (w_{ij}(n) - w_{ij}(n-1)) \tag{15}$$

where (n+1), (n), and (n-1) index next, present, and previous respectively. α is a learning rate similar to step size in gradient search algorithms. γ is a constant between 0 and 1 which determines the effect of past weight changes on the current direction of movement in weight space. This provides a kind of momentum that effectively filters out high-frequency variations of the error surface.

Step 6 Present another input and go back to step 2. All the training inputs are presented cyclically until weights stabilize (converge).

It has been shown that a MLP with at most two hidden layers can form any arbitrarily complex decision region in a feature space [18]. However, no specific rule for selection of the number of nodes in the hidden layers has yet been developed.

5. CONVENTIONAL STATISTICAL CLASSIFIERS

To be an acceptable alternative to traditional classifiers, the ANN must either outperform or at least do as well as them. In this study, two popular non-parametric statistical classifier were selected for comparison of their performances to that of the suggested ANN. These two classifiers are briefly discussed here.

The first one is the nearest neighbor rule. When an unknown sample $\mathbf{X} = [x_1, x_2, \cdots, x_n]$, x_i being one of the n utilized features, is to be classified, the nearest neighbor of $\mathbf{X}$ is found among the pool of all the M available training samples from the C considered classes and its label is assigned to $\mathbf{X}$. When number of training samples are large, the probability of error for the nearest neighbor rule has an upper bound of twice the Bayes error [3].

The distance between $\mathbf{X}$ and a training sample is measured using city block distance. This is a mapping from the n-dimensional feature space to a one-dimensional distance space. However, since the feature vector components have different dynamic ranges, it is possible for one or a subgroup of them to dominate the distance measure. To prevent this from happening and in order to equally weight distances between each component of feature vectors, the distances need to be normalized. The normalization consists of dividing by summation of the standard deviations of the corresponding component for all the C classes. Let $\mathbf{t}_k=[t_{k_1},t_{k_2},\cdots,t_{k_n}]$ denote the k th training feature vector, k=1, ... , M. The unknown test sample **X** is classified to class i^* , where

$$i^* = \{\text{class of } \mathbf{t}_j \mid d(\mathbf{X},\mathbf{t}_j) \leq d(\mathbf{X},\mathbf{t}_k),\ k=1,\cdots,M\} \tag{16}$$

and

$$d(\mathbf{X},\mathbf{t}_k) = \sum_{m=1}^{n} \left\{ \frac{|x_m - t_{k_m}|}{\sum_{i=1}^{C} \sigma_{t_m}^{(i)}} \right\} \tag{17}$$

with $\sigma_{t_m}^{(i)}$ representing the sample standard deviation of the m th element of the n-dimensional training feature vectors of class i.

The second considered classifier is a weighted minimum mean-distance rule. It characterizes each category by means and standard deviations of the components of its training feature vectors. The weighted distance between an unknown sample **X** and the mean of the features of class i, d(**X**, i), is then measured. The weighting factor is again the summation of the standard deviations of the respective component. The unknown sample is then classified to class i^* for which such distance is minimum, i.e.

$$i^* = \text{Min}_i\, d(\mathbf{X}, i) \qquad i= 1, 2, \cdots, C \tag{18}$$

and

$$d(\mathbf{X}, i) = \sum_{m=1}^{n} \left\{ \frac{|x_m - \overline{t_m}^{(i)}|}{\sum_{i=1}^{C} \sigma_{t_m}^{(i)}} \right\} \tag{19}$$

with $\overline{t}_m^{(i)}$ representing the sample mean of the m th element of the n-dimensional training feature vectors of class i. Again, weighting by the sum of the standard deviations is to balance the effect of all m feature vector components on distance d.

6. EXPERIMENTAL STUDY ON SHAPE CLASSIFICATION

In this section, the results of application of the MLP neural network classifier to two shape databases are reported. For one of the shape databases the performance on noisy images with varying SNRs are also examined. In addition, the two described traditional classifiers are applied to each of the considered problems and their performances are compared to that of the ANN.

6.1. Description of the Shape Databases

The first shape database consists of 64 × 64 binary images of all twenty-six upper case English characters from "A" to "Z". Twenty-four different images from four slightly different silhouettes of each character are generated (for a total of 624). The set of twenty-four images per character consists of six images with arbitrary varying scales, orientations, and translations from each of the four considered silhouettes per character. Fig. 6 shows the twenty-four images of character "A". In Fig. 7 the four silhouettes of each of the other characters are shown.

In addition to the above noiseless image set, five other sets of noisy images with respective SNR of 50, 25, 12, 8, and 5 dB are also constructed from the normalized images of the noiseless set. This is done by randomly selecting some of the 4096 pixels of a noiseless binary image and reversing their values from 0 to 1 or vice verse. The random pixel selection is done according to a uniform probability distribution between 1 and 4096. The SNR is computed using $20 \log[\frac{4096-L}{L}]$, where L is the number of pixels which are different between a noisy image and clean version. Fig. 8 shows one image of character "A" with different SNRs.

The second shape data set is an extensive handwritten numeral data set obtained from Recognition Equipment Incorporation. These data are gathered from 3000 forms from a United States government agency. It has approximately 86000 characters with size 32 × 24 written by approximately 3000 people selected at random from general public. Fig. 9 shows four characters per class of this data set.

6.2. Data Partitioning

An important parameter in any pattern classification problem is the estimation of the classification error. To compute it, the available samples are divided into two sets, one for training, and one for testing. For English character experiments, three cases are considered. In the first case, the available samples from each character are divided into halves such that each half contains images of each silhouette. The first half is then used for training and the second half for testing. Therefore, there are 12 training images and 12 testing images per character.

In the second case, the training is limited to four unrotated images per character. These four are the four different silhouettes of each character (i.e. images shown in Fig. 7). The remaining twenty images per class are used for testing. This way, the classifier does not see the rotated, translated, and scaled versions during learning but has to deal with them during testing.

In the third case, the 12 images of only two silhouettes per character are used for training and the remaining 12 per character which are from the other two silhouettes are tested. This way, the classifier does not see all silhouettes of each character during training but has to deal with them during testing.

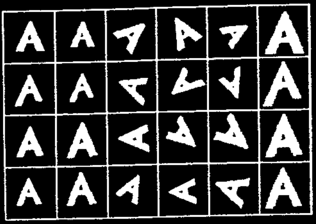


Figure 6. The twenty-four scaled, translated, and rotated images of character "A" in the data set. Note the slight variations in shapes of the images shown in the first column.

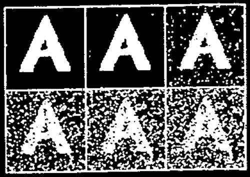


Figure 8. One sample of image of character "A" with different levels of noise. From top left to right are: noiseless, 50, 25, 12, 8, and 5 dB.

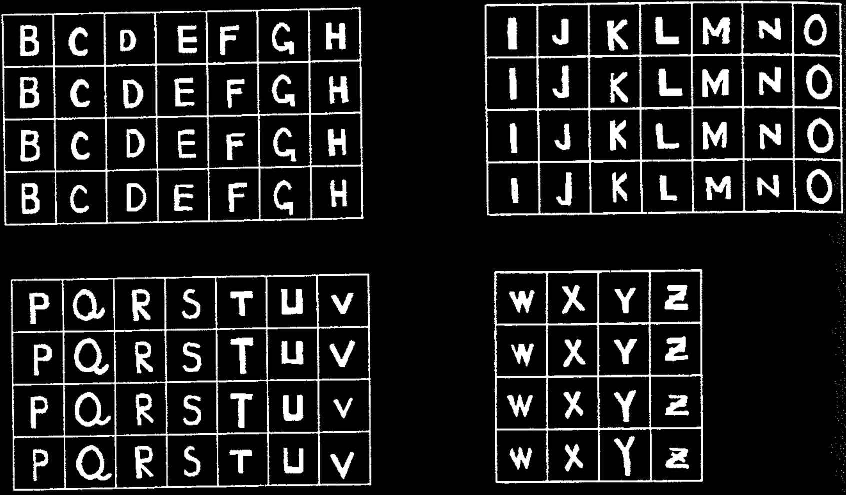


Figure 7. Four out of the twenty-four images of letters "B" to "Z" in the data set. The remaining twenty images per character are rotated, scaled, and translated versions.

In experiments with noisy images, the classifier is trained with noiseless images and tested with the noisy ones. Therefore, no noisy images are used for training.

For the handwritten numeral character experiments, a training set of 13250

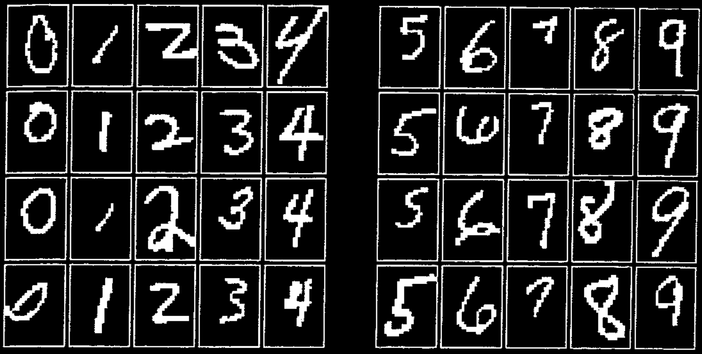


Figure 9. Four samples of each of the 10 handwritten numeral characters in the database.

images was generated by selecting 1325 characters of each numeral at random from database. A test set of 3300 characters was created by randomly selecting 330 characters of each numeral from the remaining characters in the data base. Thus, the training set is separated from the test set.

The estimation of the error rate is done by finding the ratio of the number of misclassified testing samples to the total number of tested samples.

6.3. Results

Utilizing Zernike moment features requires the selection of the maximum order, i.e., the highest n. Two different synthesis based methods for doing so are presented in [14],[15]. Using those procedures, it was decided that up to 12th order Zernike moments are required for the considered data sets. That means that 47 Zernike features are utilized (see Table 1)

The selected value of parameter β for English character database and handwritten numeral characters is 800 and 150 respectively.

The selected parameters for the MLP classifier are as follows: initial weight assignment randomly selected from [-0.5, 0.5] interval, step size $\alpha = 0.2$, learning rate $\gamma = 0.7$. The number of input nodes is 47, the number of output nodes is 26 for the case of English characters and 10 for the case of handwritten numeral characters, and finally, the number of hidden layer nodes are varied from 5 to 200 in unequal increments for some cases. Instead of testing for convergence of the weights, 500 passes over the training set is used for all cases.

The MLP and the back-propagation rule were simulated on a SEQUENT SYMMETRY S81 MIMD parallel computer utilizing 6 processors each performing 3 MIPS. The learning for the case of 47 inputs, 50 hidden layer nodes and 312 training samples took around 65 minutes.

The training features are normalized to have zero mean and unit variance before being input to the MLP. This is necessary in order to ensure that a subgroup of the features do not dominate the weight adjustment process during training. The m th feature is normalized by

$$\tilde{t}_m = \frac{t_m - \bar{t}_m}{\sigma_{t_m}} \tag{20}$$

where $\bar{t}_m$ and σ_{t_m} are the sample mean and standard deviation of the m th training features of all samples. During the testing phase each test sample is also normalized in a similar manner.

Fig.s 10 to 12 show the performances of the MLP as well as the other two classifiers with varying number of hidden layer nodes on the noiseless and noisy data sets for the three different data partitioning schemes considered. The MLP performance is shown as a function of the number of utilized hidden layer nodes. The obtained classification accuracy rates using the other two classifiers are shown in the form of legends with NM and MM standing for the nearest-neighbor and the minimum-mean-distance classifier respectively.

These results indicate that in all three cases, the performance of the nearest-neighbor classifier and the MLP are very close to each other for images that are not very noisy. However, for SNR of below 12 db, the MLP performs better. In all the cases, the minimum-mean-distance classifier yields the lowest accuracy rate. One other point to be noted is that in nearly all the examined cases, a number in the range of [20,50] for hidden nodes gave the best (or very close to best) classification accuracy. Utilizing larger than 50 hidden nodes did not alter the results significantly, especially for high SNR images.

For the experiments dealing with the handwritten numeral characters, 50 hidden layer nodes are used for the MLP classifier. The obtained accuracy rates are 83.8%, 83.45%, and 58.31% for the MLP, nearest-neighbor, and min.-mean-distance respectively. Considering the large amount of variations and distortions within samples of each class and noting that the Zernike features are global statistical ones, the obtained results are actually quite good. Combining Zernike features with other topological features which capture local structure of the image will certainly improve the recognition rate.

6.4. Fault Tolerance

One of the advantages of the neural networks is that the processing is distributed among many nodes. This provides a good degree of fault-tolerance and graceful degradation to the system. Even if some of the nodes fail to function properly, the effect on the overall performance of the system will not be much. This assertion was examined by turning off m randomly selected hidden layer nodes and observing the resulting effect on the system performance.

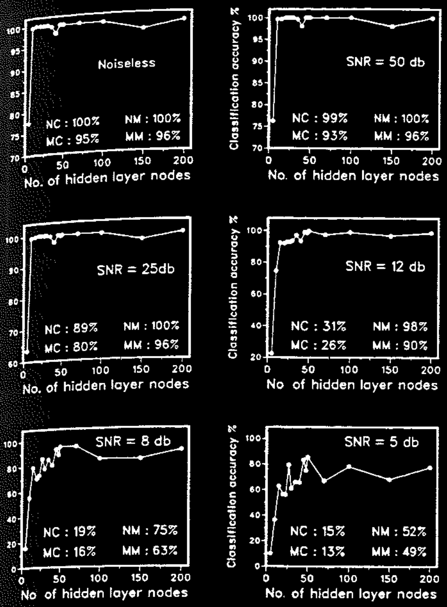


Fig. 10. Classification results of the English characters database using 47 Zernike features. Twelve images per class (three from each of the four silhouettes per class) are used for training and the remaining twelve for testing. NM and MM stand for nearest-neighbor and minimum-mean-distance classifiers, respectively.

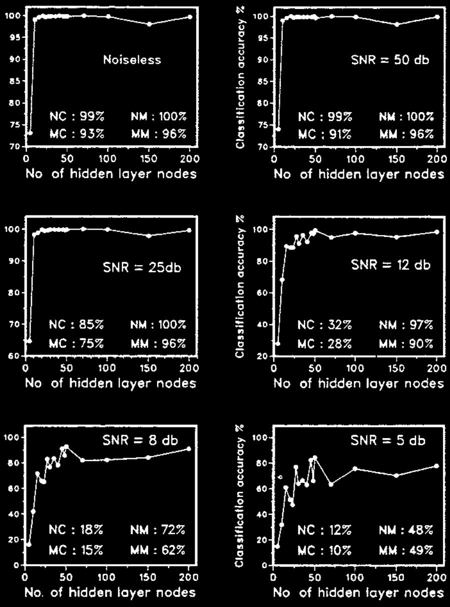
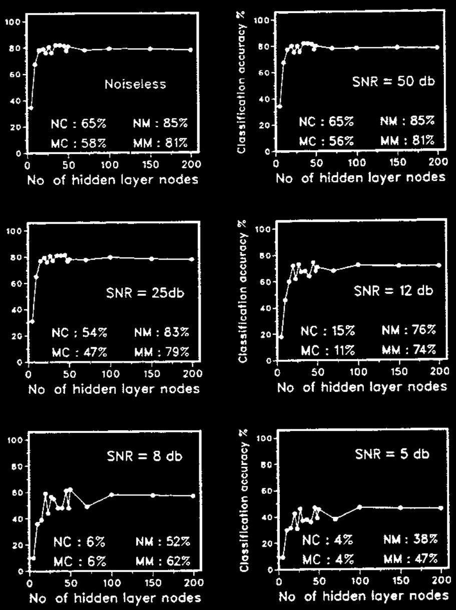


Fig. 11. Classification results of the English characters database using 47 Zernike features. Four images per class (one from each of the four silhouettes per class) are used for training and the remaining 20 for testing. NM and MM stand for nearest-neighbor and minimum-mean-distance classifiers, respectively.

The MLP with 30 hidden layer nodes using the first data partitioning method was considered for this purpose. After training the net with all 30 nodes functioning, the classification of test images was carried out with m of the hidden nodes not functioning (i.e. their output were set to zero); m was varied from 1 to 20 in increments of one and for each m, fifteen different combinations of m out of 30 nodes were considered. The result is plotted in Fig. 13 which shows a very graceful degradation of the performance.



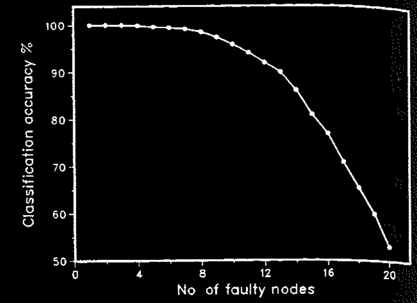


Figure 13. Degradation in system performance as a function of faulty hidden layer nodes.

Fig. 12. Classification results of the English characters database using 47 Zernike features. Twelve images of two silhouettes per class are used for training and the remaining twelve from the other two silhouettes for testing. NM and MM stand for nearest-neighbor and minimum-mean-distance classifiers, respectively.

7. EXPERIMENTAL STUDY ON TEXTURE CLASSIFICATION

A similar approach to the one described in the previous section is taken toward the study of texture recognition. Each gray-level textured image is represented by three rotation invariant real-valued features. The MLP and the other two classifiers only see these features and act upon them. In the following sections, the utilized database and the obtained results are reported.

7.1. Description of the Texture Database

A database consisting of twelve different natural textures, all taken from Brodatz [1] (a photo album which has become a standard data source for texture researchers) is constructed. Seven differently oriented 128×128 images with relative angles of 0°, 30°, 60°, 90°, 120°, 150° and 200° are taken from each texture. Each image has a (0-255) gray level range. Each 128×128 image is then segmented into four 64×64 images resulting in 28 samples for each class. A 64×64 0° oriented sample of each texture is shown in Fig. 14.

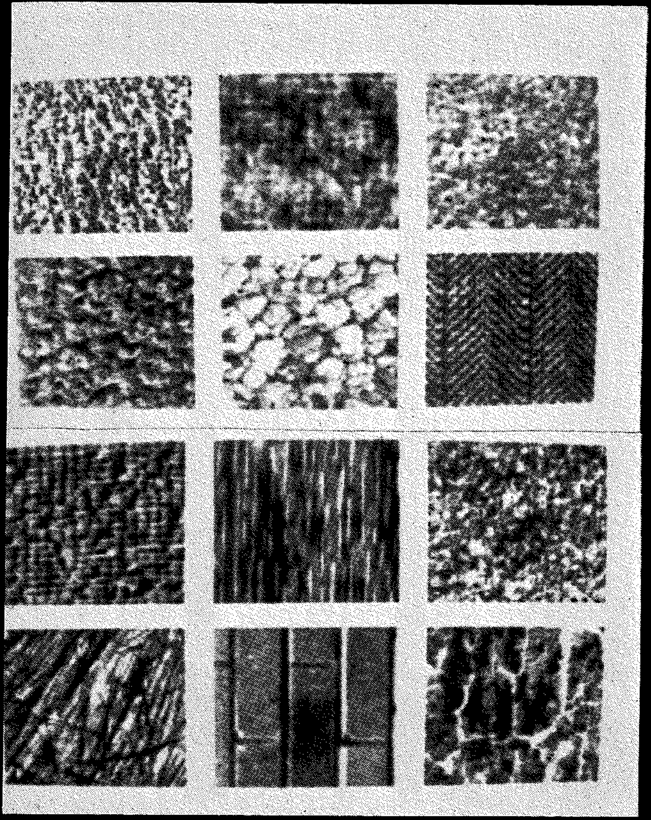


Figure 14. A 64×64,0° digitized sample of each texture of database. From left to right: first row: calf leather, wool, sand; second row: pigskin, plastic bubbles, herringbone weave; third row: raffia, wood, grass; fourth row: straw, brick wall, bark of tree.

To remove the variability caused by differences in illumination, each 64x64 image is, first histogram equalized. Next, the histogram equalized image is normalized to have zero empirical mean and unit empirical variance. In other words, each pixel value is replaced by

$$g_N(i,j) = \frac{g(i,j) - \mu_g}{\sigma_g} \qquad (21)$$

where μ_g and σ_g are the sample mean and sample standard deviation of the histogram equalized image and g_N is the pixel value of the normalized image. Note that g_N is no longer an integer and is not in the (0-255) range.

7.2. Data Partitioning

Several data partitioning schemes are considered. In the first case, the available samples are divided into two halves, one for training and one for testing. The division is in such a way that each half contains 14 samples from each class. These 14 samples consist of two samples from each of the seven orientations. Thus, the classifier "sees" instances of each texture at each orientation. In this scheme there are 168 training and 168 test samples. In the rest of the considered cases, the classifier is trained on samples from three out of the seven orientations and tested on the samples from the remaining four orientations which it has not seen during training. Thus, there are 144 training and 192 test samples for each of these cases.

7.3. Results

The only difference between the topology of the utilized MLP in this case and the one discussed for shape recognition is in the number of input and output nodes. For this application, only three input nodes corresponding to $\hat{\phi}$, $\hat{\nu}$, and ξ are needed. The number of output nodes is also reduced to 12. The number of hidden nodes is fixed at 20. The parameters of the learning rule are kept the same.

The obtained results for the MLP and the other two classifiers are shown in Table 4. Again, note that on the average the ANN does better than minimum-mean-distance classifier and about as well as the nearest-neighbor.

8. DISCUSSIONS AND CONCLUSIONS

In this chapter, the effectiveness of a feed-forward fully connected multilayer ANN for supervised classification of two-dimensional shape and texture was studied. The ANN was viewed as a new classification tool operating on features extracted from the images. For shapes, a set of scale, rotation, and translation invariant features based on the Zernike moments of the image was introduced. For textures, three random field model-based features which are rotation invariant were discussed.

Table 4
Texture classification results using all three classifiers and different data partitioning schemes.

Orientation of Training Samples	Orientation of Test Samples	Recog. Rate MLP	Recog. Rate Nearest-Neig	Recog. Rate Min.-Mean
All seven	All seven	93	93	91
0,30,60	90,120,150,200	91	94	87
30,60,90	0,120,150,200	92	92	88
60,90,120	0,30,150,200	88	91	88
90,120,150	0,30,60,200	93	93	89
120,150,200	0,30,60,90	88	92	86
0,60,120	30,90,150,200	92	93	91
30,90,150	0,60,120,200	91	92	90
0,90,200	30,60,120,150	87	85	90
0,150,200	30,60,90,120	93	90	91
30,150,200	0,60,90,120	89	90	90
	Average	**90.4**	**91.2**	**89**

The ANN classifier and two other conventional classifiers namely the minimum-mean-distance and the nearest-neighbor rule were trained and tested with similar data and their recognition accuracy rates were compared. The conclusion that is reached is that the MLP does better than the minimum-mean-distance and performs very similar to the nearest-neighbor for clean images. However, as noise is introduced into the images, the ANN becomes a much better alternative. This is due to the generalization ability of the ANN.

A point that needs to be stressed when comparing the ANN and the nearest-neighbor rule is the computational complexity. The main computational requirement of an ANN is during its training phase which can usually be performed off-line. The difference in the computational demand of a trained ANN and the nearest-neighbor is striking. Take the example of recognition of a handwritten digit represented by 47 features as discussed. A trained MLP with 50 hidden nodes makes a decision by performing 2,910 ($48\times50 + 51\times10$) multiplications, 2,910 additions, 60 sigmoid transformations and 10 comparisons. On the other hand, the nearest-neighbor classifier must search among all the 13250 training images. Each image comparison roughly requires 47 subtractions, 47 additions and 47 divisions or 141 operations. This translates into a total of 1.9×10^6 operations for the entire database. Then, 13250 comparisons are needed to reach a decision. This represents several hundred times more computation compared to the ANN. Thus, it can be concluded that regardless of the noise level, the ANN classifier is a better choice over the other two classifiers.

Of course, the main difficulty with the MLP is the absence of a systematic method for selection of the appropriate number of hidden nodes. However, as the empirical results indicate, there seems to be a threshold for such a number beyond which the performance is not altered significantly. The other problem is the slow rate of convergence of the back-propagation learning rule and the fact that it does not guarantee a global minimum when searching for the best weights.

In summary, the presented material in this chapter shows that the neural network can serve as a good alternative to the considered conventional classifiers. If the data is noisy or the number of training samples are large, it becomes the best choice.

Acknowledgement

This work was supported in part by DARPA under Grant MDA 903-86-C-0182.

9. REFERENCES

[1] P. Brodatz, *Texture A Photographic Album for Artists and Designers*, Dover, New York, N.Y., 1956.

[2] D. J. Burr, "Experiments on Neural Net Recognition of Spoken and Written Text," *IEEE Trans on ASSP*, Vol. 36, No. 7, pp. 1162-1169, July, 1988.

[3] R. O. Duda, and P. E. Hart, *Pattern Classification and Scene Analysis*, John Wiley & Sons, New York, 1973.

[4] R. C. Gonzalez, and P. Wintz, *Digital Image Processing*, Addison-Wesley, Reading, MA, pp. 119-127, 1977.

[5] R. M. Haralick, "Statistical and Structural Approaches to Texture," *Proceedings of IEEE*, Vol. 67, pp. 786-804, 1979.

[6] R. M. Haralick and L. G. Shapiro, "SURVEY-Image Segmentation Techniques", *Computer Vision, Graphics, and Image Processing*, 29, pp. 100-132, 1985.

[7] R. L. Kashyap, "Analysis and Synthesis of Image Patterns by Spatial Interaction Models," Progress in Pattern Recognition, Edited by L. N. Kanal, and A. Rosenfeld, Vol. 1, North-Holland, New York, pp. 149-186, 1981.

[8] R. L. Kashyap, and R. Chellappa, "Estimation and Choice of Neighbors in Spatial Interaction Models of Images," *IEEE Trans on Information Theory*, Vol. IT-29, pp. 60-72, 1983.

[9] R. L. Kashyap, R. Chellappa, and A. Khotanzad, "Texture Classification Using Features Derived from Random Field Models," *Pattern Recognition Letters*, Vol. 1, pp. 43-50, 1982.

[10] R. L. Kashyap, and A. Khotanzad, "A Model-Based Method for Rotation Invariant Texture Classification," *IEEE Trans on Pattern Analysis and Machine Intelligence*, Vol. PAMI-8, pp. 472-481, 1986.

[11] A. Khotanzad, "Stochastic Model-Based Techniques for Classification and Segmentation of Textures," Ph. D. dissertation, School of Electrical Engr., Purdue Univ., W. Lafayette, IN., 1983.

[12] A. Khotanzad, and A. M. Bouarfa, "Image Segmentation by a Parallel, Non-Parametric Histogram Based Clustering Algorithm", To appear in *Pattern Recognition.*

[13] A. Khotanzad, and J.-Y. Chen, "Unsupervised Segmentation of Textured Images by Edge Detection in Multidimensional Features", *IEEE Trans on Pattern Analysis and Machine Intelligence*, Vol. 11, No. 4, pp. 414-421, April, 1989.

[14] A. Khotanzad, and Y. H. Hong, "Invariant Image Recognition by Zernike Moments,"*IEEE Trans on Pattern Analysis and Machine Intelligence*, Vol. 12, No. 5, pp. 489-498, May, 1990.

[15] ___, "Rotation Invariant Image Recognition Using Features Selected via a systematic Method,"To appear in *Pattern Recognition.*

[16] A. Khotanzad, and R. L. Kashyap, "Feature Selection for Texture Recognition Based on Image Synthesis,"*IEEE Trans on Sys, Man, Cyber*, Vol. SMC-17, No. 6, pp. 1087-1095, Nov./Dec., 1987.

[17] A. Khotanzad, and J. H. Lu, "Classification of Invariant Image Representations Using a Neural Network,"*IEEE trans on ASSP*, Vol. 38, No. 6, pp. 1028-1039, June, 1990.

[18] R. P. Lippmann, "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, Vol. 4, No. 2, pp. 4-22, April 1987.

[19] D. E. Rumelhart, and J. L. McClelland, *Parallel Distributed Processing Explorations in the Microstructure of Cognition*, Vol. 1: Foundations. MIT press, 1986.

[20] M. Teague, "Image analysis via The General Theory of Moments," *J Opt Soc Am*, Vol. 70, No. 8, pp. 920-930, August 1980.

[21] C. H. Teh, and R. T. Chin, "On image Analysis by the Methods of Moments," *IEEE Trans on PAMI*, Vol. 10, No. 4, July 1988.

[22] F. Zernike, *Physica*, Vol. 1, pp. 689, 1934.

Artificial Neural N
Old and New Con
I.K. Sethi and A.K
© 1991 Elsevier S

Neural Net

J. Ghosh and

Department of E
Texas 78712-10

Abstract

We review
images, and hig
which provide
materials and co
image analysis.
study of textur
segmentation of
This approach
Cooperative neu
next, and relate
texture-based s
processing textu

1. INTRODU

In order to v
one needs to not
of the imaging p
surfaces are smo
assumptions und
always we are u
computation ne
effortless On t
machine vision,
with variations i

The tremend
three decades ha
psychological o
signal processin
Some researche
develop useful
serving as a com

Artificial Neural Networks and Statistical Pattern Recognition
Old and New Connections
I.K. Sethi and A.K. Jain (Editors)
© 1991 Elsevier Science Publishers B.V. All rights reserved

Neural Networks for Textured Image Processing

J. Ghosh and A. C. Bovik

Department of Electrical and Computer Engineering, The University of Texas, Austin, Texas 78712-1084, U.S.A

Abstract

We review key conventional and neural network techniques for processing of textured images, and highlight the relationships among different methodologies and schemes. Texture, which provides useful information for segmentation of scenes, classification of surface materials and computation of shape, is exploited by sophisticated biological vision systems for image analysis. A brief overview of biological visual processing provides the setting for this study of textured image processing. We explain the use of multiple Gabor filters for segmentation of textured images based on a locally quasimonochromatic image texture model. This approach is compared to the functioning of localized neuronal receptive fields. Cooperative neural processes for perceptual grouping and emergent segmentation are reviewed next, and related to relaxation labelling. The recently developed SAWTA neural network for texture-based segmentation is then presented. Finally, techniques for describing and processing texture as a constrained optimization problem are outlined.

1. INTRODUCTION

In order to visually reconstruct the world that projects itself as an image on a 2-D surface, one needs to not only analyze the image but also have a model of the nature of that world and of the imaging process. A plausible model might assume, for example, that objects are rigid, surfaces are smooth and boundaries are continuous. Optical illusions are perceived when the assumptions underlying our internal models about the imaged world fail. However, almost always we are unaware not only of these internal models but also of the tremendous amount of computation needed for preprocessing and recognition of images, so that vision seems effortless. On the other hand, even the basic step of detecting edges poses a problem for machine vision, since any kind of textured image gives a multitude of noisy edge segments with variations in reflectance and illumination [43].

The tremendous amount of research in image processing and computer vision over the past three decades has been influenced not only by physiological or psychophysical discoveries and psychological observations about perception among living beings, but also by advances in signal processing, computational mathematics, pattern recognition and artificial intelligence. Some researchers in the recently rejuvenated field of neural networks are also attempting to develop useful models of biological and machine vision. With the human visual system serving as a common source of inspiration, it is not surprising that neural network approaches to image processing/ understanding often have commonalities with more traditional techniques. However, they also bring fundamentally different elements of adaptation and learning, and promise breakthroughs through massively parallel and distributed implementations in VLSI[18]

In this chapter, we present both conventional and artificial neural network (ANN) techniques for the processing of textured images. The goal is not only to compare different methodologies, but also to highlight the relationships among them so that more comprehensive techniques incorporating the best of the diverse approaches can be developed. For our purposes, an ANN is a collection of computing cells (artificial neurons) interconnected through weight links (synapses with varying strengths). The cells compute asynchronously and in parallel using information available locally or from topologically adjacent cells through the weighted links. The knowledge of the system is embodied in the pattern of interconnects and their strengths which vary as the system learns or adapts itself. This learning can be supervised by an external "teacher" or through some local rule such as Hebb's rule [25].

Section 2 reviews the best accepted computational schemes for detecting edges, and then introduces self-organizing neural networks giving rise to cells that are sensitive to particular orientation and position of edges These neural networks use simple local learning rules to yield cell behavior similar to those found in the striate cortex of mammals In Section 3, we examine different models of textured images and the corresponding paradigms for their analysis. A localized filtering approach to texture processing that is inspired both by its computational properties and biological plausibility, is examined in some detail. Section 4 presents several neural network models that are pertinent to processing of textured images, and compares them to those presented in earlier sections.

The interested reader can find a good coverage of the more conventional techniques in image understanding and machine vision in the texts by Ballard and Brown [4] and Horn [29], and in the collection of papers by Rosenfeld [51] and Fischler & Firschein [16]. A comprehensive list of papers published annually on computer vision and image processing is provided by Rosenfeld [52], and indicates the enormous amount of research in these areas. More interdisciplinary works include Arbib & Hanson [2] which provides a broad overview of problems in vision from neurophysiological, psychophysical and computer science perspective. Theories of visual cognition are expounded in several books such as Pinker [50]. A good collection of seminal papers on neural networks can be found in Anderson & Rosenfeld [1].

1.1 Biological Visual Processing.

We first briefly outline key features of the mammalian visual system in order to define some terms and characteristics that are alluded to in the following sections Details can be obtained from Kandel & Schwartz [35] or Van Essen [54]

As indicated in Fig. 1, visual patterns impinging on the retina are encoded and conveyed to the visual cortex via the lateral geniculate nucleus (LGN) Processing takes place concurrently in all these three segments The retina contains over 100 million photoreceptors, namely the rods and cones that are sensitive to light intensity and wavelength (color) respectively Bipolar cells establish conduction lines from the rods and cones to the ganglion cells, while the horizontal and amacrine cells establish inhibitory crosslinks among these conduction lines. This results in each ganglion cell collecting the output of a localized group of photoreceptors that forms its *receptive field*. Ganglion cell receptive fields tend to have center-surround profiles such that a uniform illumination on the receptive field elicits no response from a

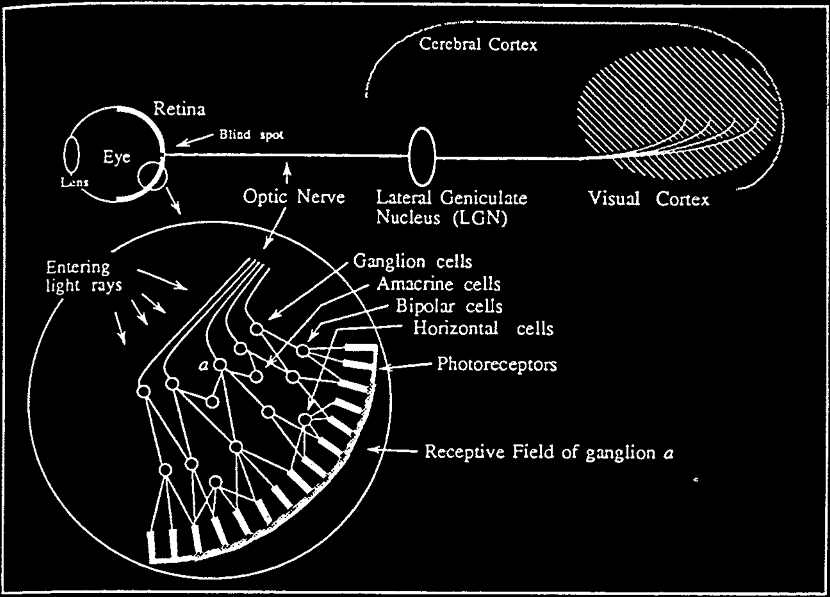


Figure 1: The mammalian visual system [46]

ganglion, but some form of contrast does. The long axons from all the ganglion cells are bundled together to form the optic nerve, which follows two separate pathways to the brain. One leads to the superior colliculus in the midbrain, presumably for controlling eye movements, while the other leads to the visual cortex via the LGN.

The visual cortex has been subdivided into several areas according to function and/or neural structure. Each area is essentially a 2-D layered sheet of neurons. Interconnections among the areas are invariably bi-directional. Area V1, also known as area 17 or the striate cortex, contains a more or less topographic map of the visual field, just like the LGN It contains simple cells, complex cells and hypercomplex cells A simple cell fires at maximum frequency in response to a small edge with a specific angular orientation and position in the visual field. Its response falls dramatically for small shifts from this position, or for a change of more than 20 degrees in orientation Complex cells, while remaining orientation sensitive, are more tolerant to position changes. Hypercomplex cells have been observed to respond to line terminations and corners. These three types of cells are believed to be organized in a roughly hierarchical fashion. A simple cell receives input from a group of LGN cells responding to the same edge; the receptive field of a complex cell is a group of neighboring simple cells with the same orientation preference, and a hypercomplex cell receives excitatory input from some complex cells and inhibitory inputs from others.

Structurally, the cells of V1 are organized in a columnar fashion according to orientation preference or eye preference. Cells lying in a column perpendicular to the surface of the cortex are found to respond preferentially to the same angular orientation and same eye. Moving from one column to the next across the cortex, one finds that the preferred orientation changes continuously, about 10 degrees every 50mm, and the eye preference alternates between the left and right eyes in an orthogonal direction

In the middle temporal area and the medial superior temporal area, neurons are found to be highly selective to speed and direction of motion. However, they exhibit little or no sensitivity to shape or color. A columnar organization similar to the V1 is found except that the cells are now grouped according to the direction of motion The cells of the inferotemporal cortex (IT) have very large receptive fields, and seem to be sensitive to extremely complex shapes. For example, cells that respond selectively to the silhouette of a monkey's hand or to particular faces, have been reported

2. DETECTION OF EDGES IN COMPUTER AND HUMAN VISION

An image is a distribution of light intensity values. An edge in an image is characterized by a rapid, sustained change in intensity as one traverses in a direction orthogonal to it. Thus an edge can be described by its location, its direction (orthogonal to the local intensity gradient), and its magnitude which is a measure of the intensity change Edges are useful in defining boundaries, characterizing texture, and for detecting shapes The detection of edges is central to image preprocessing in biological systems, wherein the retina, LGN and visual cortex seem to be actively involved in some forms of contrast enhancement or edge detection. In this section, we review the best accepted computational schemes for detecting edges, and then introduce self-organizing neural network models that lead to development of cells that are sensitive to particular orientation and position of edges These neural networks use simple local learning rules to yield cell behavior similar to those found in the striate cortex of mammals.

2.1 Edge Detection: Computational Techniques.

Most computational schemes for edge detection use operators for detecting gradient maxima of the intensity surface [44] or for estimating the parameters of an assumed edge model [24]. Since edges in actual images deviate substantially from ideal step discontinuities, some form of postprocessing is required to eliminate the edges caused by noise and other effects. In natural images, edges can occur over a wide range of scales or resolutions Marr and Hildreth showed that filtering an image with a Gaussian filter restricts the range of resolutions over which intensity changes can occur while maintaining the spatial location of detected discontinuities in a near-optimal fashion. After an image has been Gaussian filtered at several resolutions by adjusting the variance of the Gaussian function, the Laplacian operator is applied to detect intensity change maxima, indicated by the zero crossings in the output. The effect of the Gaussian function and Laplacian operator can be combined to yield a single Laplacian of Gaussian or LOG filter. An edge is indicated by a segment of zero-crossings that occur at the same positions for more than one resolution.

In a quantitative model of human spatial vision proposed by Wilson & Bergen [59], the authors relied on neuroanatomical studies to suggest that the receptive fields in the fovea have four different sizes for any point in the visual field, and that each receptive field has a profile similar to a "Difference of Gaussian" or DOG filter This filter is remarkably similar to the LOG filter. In an attempt to relate his model to the human visual system, Marr postulated that the LGN computes the zero-crossings of a LOG filter while simple cells detect lines of zero crossings that indicate an edge.

The directional edge-detector of Canny [6] is also based on detecting the zero-crossings of a 2-D Gaussian but is more computationally efficient than the Marr-Hildreth detector as it is able to combine the differentiation and smoothing steps, and is less sensitive to noise. It is interesting that the Gabor filters that are described in detail in Section 3, also have attractive properties for detecting certain types of localized contrasts in an image.

2.2 Development of Orientation-Selective Cells in Self-Organizing Neural Networks.

Through their studies of the visual striate cortex (area 17) of the cat and monkey, Hubel and Weisel [31,32] found neurons that are selectively sensitive to light bars and edges of a certain orientation. They also detected "functional columns" in which neurons within a column are sensitive to the same orientation, and neighboring columns tend to respond to stimuli of similar orientation Malsburg [40] succeeded in developing a neural network model that could evolve to exhibit behavior similar to those found experimentally by Hubel and Weisel. This model consisted of a 2-D "cortical plane", containing excitatory (E) and inhibitory (I) cells of uniform and equal density. The E cells had excitatory connections to nearby E and I cells, with connection strengths decreasing with distance, while the I cells would inhibit E cells over a *wider* area. Bell-shaped functions were used to describe the decreasing intra-cortical strengths with cell distance for both cell types. The input image was projected onto the "retina", a plane of A cells that were initially connected to the cells in the cortical plane through synapses of random strengths. Only the connections afferent to the E cells were modified using Hebb's rule [25] and renormalized so that the net strength of afferent connections to a cell remained constant. In Malsburg's model, $V_k(t)$, the excitatory state of cell *k*, evolves according to:

$$\frac{d}{dt}V_k(t) = -\alpha_k V_k(t) + \sum_l p_{lk} V_l^*(t) + \sum_i s_{ik} A_i^*(t), \tag{1}$$

where s_{ik}, p_{lk} are connection strengths, $A^*_i(t)$ is the signal of afferent fiber *i* The output signal V_k^* (t) of cell *k* is given by:

$$\begin{aligned} V_k^*(t) &= V_k(t) - \theta_k, \quad \text{if } V_k(t) > \theta_k; \\ &= \quad \text{zero} \quad \text{otherwise.} \end{aligned} \tag{2}$$

Note that Eq. 2 brings about a non-linearity that is essential for suppression of noise.

In one experiment, a retina of 19 cells was stimulated with lines at 9 different angular orientations The cortex was a hexagonal arrangement of 338 E and 338 I cells For each input line, the steady state responses of the cells were noted The cells were then labeled with the line orientation to which they responded most strongly. Figure 2 shows the results after 100 learning steps. The bars indicate the response orientation of each E cell. The dots marks cells which never reacted to the standard set of stimuli. Malsburg's results indicate that cortical organization is not genetically predetermined These results are also commensurate with the findings of dramatic experiments such as those conducted by Hirsh & Spinelli [26], who raised kittens wearing goggles so that they could only see horizontal line segments in one eye and vertical line segments with the other A rewiring of the visual cortex took place so that cells connected to the first eye only developed horizontal receptive fields, cells connected to the second only had vertical orientation, and the number of binocular cells that responded to both eyes decreased dramatically

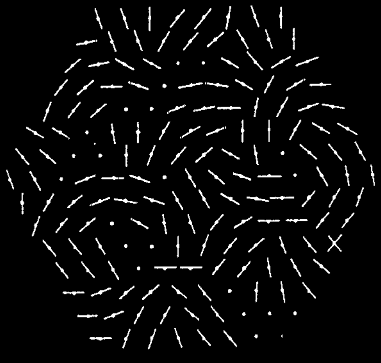


Figure 2: View of Malsburg's "cortex" after 100 learning trials. The bars indicate the orientation selectivity of the E-cells in the cortex [40]

A completely different and enlightening model proposed by Linsker [39] demonstrates that experience is not necessary to develop the edge-detection function of either LGN-type cells or simple cells. He uses a multilayer feedforward network with linear units. Each unit within a layer receives inputs from a local area of units in the previous layer, and performs a linear summation of the weighted inputs. The weights of the network are modified using a variant of Hebb's rule, which is shown to maximize the variance in a layer's response to the input patterns from the previous layer. Note that in his model, the adaptive afferent connection weights to a layer of cells are allowed to settle down before the output of those cells are used to modify the connections to the next layer. If random noise is used as the input to Linsker's network, then center-surround cells that respond maximally to a bright spot centered on the cell's receptive field against a dark background (or to a dark spot on a bright background) emerge in the third layer. Orientation-selective cells develop in the seventh layer. Moreover, if lateral interactions are added to this layer, then a columnar-type organization similar to that in the striate cortex is found to develop. Other self-organizing approaches to feature detection have been discussed by Grossberg [23] and Kohonen [37]

3. TEXTURE ANALYSIS USING MULTIPLE CHANNEL FILTERS

In this section, we first describe various approaches to the modeling and analysis of textured images. Subsequently, we focus on a computational framework for analyzing image textures using the class of two-dimensional image processing filters known collectively as *Gabor functions*. These functions have played an important role both in the development of models of biological visual information processing and in the development of computer vision algorithms for low-level image processing and analysis.

3.1 Modeling of Textured Images.

Texture can be used in the analysis of images in several ways: in the segmentation of scenes into objects, in the classification of surface materials, and in the computation of shape.

However, an exact definition of texture as either a surface property or as an image property has never been adequately formulated While the concept of a surface texture as a pattern of variations in macroscopic surface topology is easy to accept, real-world surface textures are difficult to model. Attributes giving rise to a sense of perceptual homogeneity have been construed to include such spatio/temporal surface properties as color, relative depth, motion, and flicker rate [34]. Even for static, monocular, monochromatic images, modeling texture is quite complex, since an accurate model must incorporate descriptions of both the optical properties of the surface materials and of the geometries of the lighting sources and imaging system However, much progress has been made towards developing texture analysis techniques recently, by examining the problem in the post-image formation setting. Here, an image texture is considered as a local arrangement of image irradiances projected from a surface patch of perceptually homogeneous radiances. A technique that use such a characterization are highlighted in Section 3.3.

Early efforts at texture analysis were largely motivated by the need for classifying aerial images, such as those obtained from remote sensing, into categories based on their textural properties [24]. Texture was described by (1) the local tonal primitives out of which the image texture is composed, and (2) the number and types of these primitives and their spatial organization. A tonal primitive is a maximally connected set of pixels with a common tonal property, and is characterized by its area, shape, average tone, etc. Spatial organization was typically characterized by the structural, probabilistic or functional dependence of one primitive on its neighbors This leads to statistical approaches using autocorrelation functions, optical transforms, digital transforms, gray tone co-occurrence, run lengths and autoregressive methods [57]. Some structural approaches based on more complex primitives than gray tone were also attempted, assuming that a texture is made up of primitives which appear in near-regular, repetitive spatial arrangements. Then, a textured image could be described in terms of the primitives and the placement rules that encapsulated the dependency of a particular primitive being placed at a given location on the positioning of other primitives The structural techniques were found to be of limited applicability.

A fundamentally different type of model for preattentive texture perception has been proposed by Julesz and others [34]. They contend that texture processing involves the computation of the densities of fundamental texture features called "textons " Textons that have been considered so far include elongated blobs (line segments, ellipses, etc.) of various orientations, sizes and aspect ratios, line terminations, and crossing of line segments. Segregation of textures proceeds by comparison of relative texton densities [55]. Texton detection is consistent with the feature detection model of early visual processing that involves the responses of the bar- and edge-sensitive receptive fields There is substantial evidence that the visual system regards texton-type features as important, and that visual discrimination of textures coincides with differences in texton density. However, there is no apparent mechanism that suggests that textons are resolved first and their densities are computed later.

The use of direct Fourier analysis for analyzing texture dates from Bajcsy [3], who suggested dividing images into smaller subimages within which textural information could be derived from local Fourier spectra. Several other authors extended this concept by using the outputs of multiple frequency-selective filters applied to textured imagery as *features attributes* in classical pattern recognition paradigms configured to allow texture discrimination/ classification [38]. The fact that granularity (or texture coarseness) and orientation are meaningful low-level texture descriptors led researchers to consider density- and orientation-selective mechanisms for analyzing texture Segmentation/discrimination algorithms were reported for detecting periodic and oriented image regions by partitioning the image spectrum into bins or by locating spectral peaks. A multi-channel segmentation algorithm using both orientation-sensitive and moderately narrow channel bandwidths was introduced in [9]. Again, segmentation is augmented by clustering the channel outputs.

The use of space-frequency localized filters was further explored by several authors [5,53] all of whom suggest a class of functions known as *Gabor filters* for analyzing, discriminating, and segmenting images based on texture. The general framework for computational texture analysis using Gabor functions which developed from this research, and more recent extensions, is described in Sections 3.3. First, however, we will discuss the important role that Gabor functions have played in recent models of biological visual processing

3.2 Gabor Functions and Neuronal Receptive Fields.

Computational models of low-level biological vision often derive from physiological and psychophysical investigations of the structure and function of neuronal receptive fields in the post-retinal ganglion cells, the LGN, and the striate cortex. Until recently, two schools of thought dominated theories of early biological vision or neuronal image processing: the *feature detection paradigm* as exemplified by the early work of Hubel and Weisel [30,31] and the *Fourier decomposition paradigm* as exemplified by the somewhat later work of Pantle and Sekular [47]. Briefly, the first school of thought held the view that early visual information processing entailed the detection of specific features processed by higher abstract mechanisms; the second held that early visual image analysis could be likened to local spectrum analysis techniques.

More recent work suggests that, at the cortical level, both interpretations may be correct. Recent models of neural receptive fields in the striate cortex, in conjunction with the functional version of the uncertainty principle, suggest that biological neurons have evolved to be sensitive to elementary features occurring over a range of scales (frequency ranges), or equivalently, that images are decomposed by the cortical neurons into narrowband frequency channels that are highly spatially localized (sensitive to local features).

Marcelja [42] observed the highly-oriented simple cell receptive fields in the cortex can be accurately modeled by 1-D Gabor functions, which are Gaussian modulated sine wave functions. The Gabor functions play an important role in functional analysis and in physics, since they are the unique functions that satisfy the *uncertainty principle,* which is a measure of the function's simultaneous localization in space and in frequency [17]

Daugman [11] successfully extended Marcelja's neuronal model to 2-D, also extending Gabor's result by showing that the 2-D Gabor functions are the unique minimum-uncertainty 2-D functions. The implication of this for texture analysis purposes, and perhaps for neuronal processing of textured images, is that highly accurate measurements of textured image spectra can be made on a highly localized spatial basis This simultaneous localization is important, since then it is possible to accurately identify sudden spatial transitions between texture types, which is important for segmenting images based on texture, and for detecting gradual variations within a textured region, which is important for computing deformations arising from surface defects, or from effects arising from the projection of 3-D textured surfaces into 2-D images. Other researchers have independently confirmed the validity of the Gabor receptive field model.

3.3. Multiple Channel Texture Segmentation.

We now describe a texture analysis model motivated in large part by the Gabor receptive field model. It is important to observe that the model is advocated only as a convenient computational formalism, not as a probable model of cortical processing. Nevertheless, it is not unreasonable to suggest that the simple cells play an important role in the early stages of visual textural processing.

The convolution version of the complex 2-D Gabor functions take the general form (where j is the square root of -1)

$$h(x, y) = g(x', y') \cdot \exp[2\pi j(U_0 x + V_0 y)] \tag{3}$$

where $(x', y') = (x \cos a + y \sin a, -x \sin a + y \cos a)$ are rotated coordinates, and

$$g(x, y) = (1 / 2\pi l s^2) \cdot \exp\left\{-[(x/l)^2 + y^2] / 2s^2\right\} \tag{4}$$

Thus, h(x, y) is a complex sine grating modulated by a 2-D Gaussian function with aspect ratio l, scale parameter s, and major axis oriented at an angle a from the x-axis. If l = 1, a need not be specified since g(x, y) is circularly symmetric The spatial frequency response of the Gabor function (3) is then

$$H(u, v) = \exp\left\{-2\pi^2 s^2[(u - U_0)^2 + (v - V_0)^2]\right\}, \tag{5}$$

a bandpass Gaussian with radial center frequency $w_0 = \sqrt{U_0^2 + V_0^2}$ (cycles/image) and orientation $q_0 = \tan^{-1}(V_0/U_0)$ (degrees or radians measured from the u-axis)

Assume that t(x,y) is a real-valued, continuous intensity image that *locally* contains only narrow ranges of spatial frequencies:

$$t(x, y) = I(x, y)\left\{1 + 2\, C(x, y) \cos\left[2\pi\, W(x, y)\right]\right\}. \tag{6}$$

Here the 2-D function $I(x, y) \geq 0$ represents a slowly-varying ambient image intensity function, $2 \cdot C(x, y)$ Œ [0, 1] is an amplitude or *texture contrast function,* and W(x, y) is a 2-D *texture phase function* For simplicity, assume C(x, y) = C and I(x, y) = I to be so smooth that they may be regarded locally constant If W(x, y) is assumed to vary smoothly nearly everywhere, then (6) can be used to model an image intensity function that is highly coherent or spatially correlated. More specifically, the image t(x, y) may be regarded as narrowband or *quasimonochromatic* on a local basis: at each (x, y), the local frequency content of the image away from the frequency origin will be concentrated about $—W(x, y) = \left[U(x, y), V(x, y)\right]$, where the gradient components $U(x, y) = \frac{\partial}{\partial x} W(x, y)$ and $V(x, y) = \frac{\partial}{\partial y} W(x, y)$ are *instantaneous texture frequencies.*

The model (6) admits a computational model for textured image analysis by isolating local image frequencies using tuned narrowband channel filters. By using the logarithmic image and removing the low-frequency terms by filtering, the model can be simplified to:

$$t(x, y) = 2\,C \cos\left[2\pi\, W(x, y)\right] \tag{7}$$

The concept of *emergent texture frequencies*, given by the responses of Gabor channel filters [5], can be used for estimating the locally dominant or instantaneous image frequencies in t(x, y). We now describe a simple method for segmenting image textures by simple comparison of the responses of these Gabor filters. The success of the technique is quite impressive, given that there no use is made of any sophisticated pattern classification superimposed on the basic segmentation structure However, it is important that a large set of channel filters be used to sample the frequency plane densely, to ensure that a filter exists that will respond strongly to any dominant texture frequency component

In the examples given here, an implementation utilizing forty unity aspect ratio Gabor filters is used, with filter orientations ranging over 8 evenly spaced angles in the right-half frequency plane, 5 filters along each orientation The radial center frequencies form a geometric progression with common ratio 1 8, ranging from 9.6 cycles/image to 100 cycles/image. The filters are spaced approximately 1 octave apart with responses intersecting well above half-peak A single additional Gabor filter centered at the frequency origin (a Gaussian filter) is also applied, yielding a total of 41 responses: $m_n(x, y) = | t(x, y) * h_n(x, y) |$; $n = 1, \ldots, N$, where $N = 41$, and where $h_n(x, y)$ are Gabor functions indexed by center frequencies (U_n, V_n) and space constants s_n. The set of filters used is depicted by Fig. 3, which shows the spatial frequency responses of all 41 filters

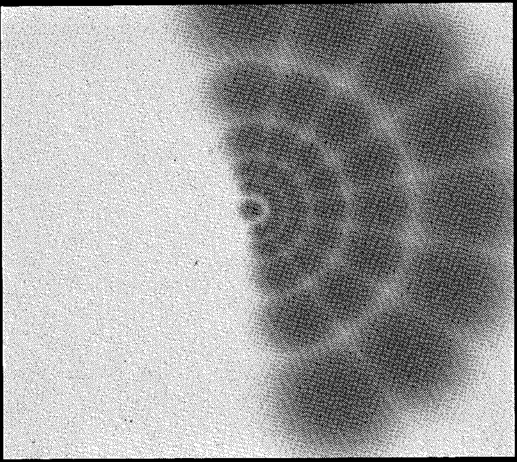


Figure 3 Plot of spatial frequency responses of Gabor functions used to analyze texture.

By choosing the filter amplitude response $m_{n'}(x, y)$ that is maximum at (x, y), a rough estimate $(U_{n'}, V_{n'})$ is obtained, i e., we choose:

$$n' = \arg\left[\max_{1 \le n \le N} \{m_n(x, y)\}\right] \tag{8}$$

A simple and useful segmentation of the image is then obtained by the region assignment

(x, y) Œ $¬_n$, if n = n' (9)

Usually, direct application of (9) leads to segmentation errors arising from modeling errors, noise, and local irregularities in the textures. These can often be effectively ameliorated by smoothing each of the channel amplitude responses with a Gaussian smoothing filter having the same shape as the corresponding channel filters but a greater spatial extent, prior to making region assignments. Thus, the alternate definition

$$n' = \arg \left[\max_{1 \leq n \leq N} \{ g_n(x/g, y/g) * m_n(x, y) \} \right] \qquad (10)$$

will yield a smoother segmentation when used with the segmentation paradigm (9). We have found the value g = 2/5 to be effective in most instances.

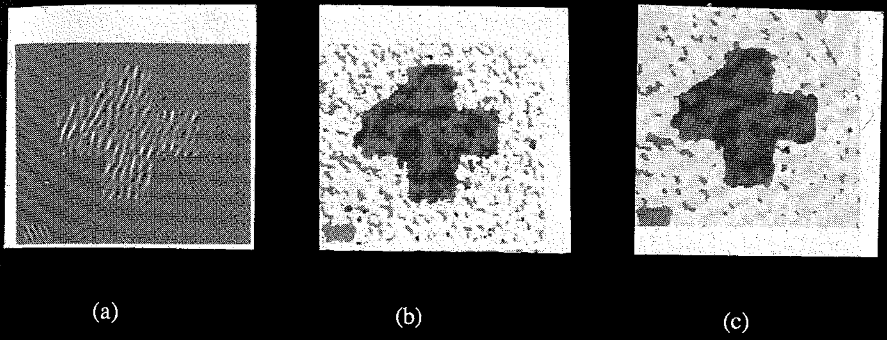


Fig. 4 Segmentation of a synthetic texture (a) Original texture; (b) segmentation using (8) and (9); (c) Segmentation using (10) and (9).

Figure 4 depicts an example of the segmentation approach (9) as applied to a simple narrowband textured image. Later, in Section 4, we will study a cooperative-competitive feedback mechanism to further improve the segmentations that are achieved using (9), as originally described in [19]. A method for using the intantaneous texture frequency to estimate surface orientation is given in [20].

4. NEURAL NETWORK APPROACHES

Cooperative processes such as relaxation labeling have been explored by the vision community for over a decade, without explicitly casting them in a neural network framework. These mechanisms are very akin to neural network models such as the Boundary Contour System [23], which is described in Section 4.2. This section also presents a hierarchical network that first extracts relevant features and then performs texture segmentation based on these features. Then we consider the SAWTA network that iteratively combines smoothing and categorizing functions to yield a versatile segmentation algorithm based on the multichannel model of texture given in Section 3.3 An alternative approach is to consider texture segmentation as an optimization problem, and maximize the *a posteriori* distribution of the intensity field based on a Markov random field model of texture This method is outlined in Section 4.5.

4.1 Cooperative Processes.

Local intensity edges typically form a part of a global line or boundary rather than occurring in isolation Thus the presence or absence of a nearby edge of similar angular orientation would tend to reinforce the hypothesis of the existence of an edge at a given point in the intensity field This idea is the basis of a cooperative process called relaxation labelling [13] that can be used to enhance lines and curve in an image First, a bank of filters that detect small lines at different orientations are used to generate an array of nodes, with each node corresponding to a location in the image, and having a line-orientation label assigned to it. These labels are updated by a relaxation process, such that they become more compatible with neighboring labels. Thus adjacent "no-line" labels support one another, and so do lines with similar orientation, while two adjacent labels corresponding to orthogonal orientations antagonize each other. A refined relaxation-labeling algorithm with associated convergence conditions is given in [33] Cooperative processes have also been used to obtain curvature estimates, for detecting corners in dotted line drawings, and for enhancing perceptually significant features [56]

A cooperative-competitive scheme called the boundary contour system (BCS) has been proposed by Grossberg and Mingolla [21] to explain how edges are filled in when part of a boundary is missing, and how illusory contours can emerge from appropriately positioned line-terminations. This scheme is elaborated on in the next section, which also presents a hierarchical network for texture segmentation that uses BCS

4.2 Perceptual Grouping and Emergent Segmentation.

A real-time visual processing model has been developed by Grossberg and Mingolla [21] to analyze and explain a variety of perceptual grouping and segmentation phenomena, including the grouping of textured images. A key component of this model is the Boundary Contour System (BCS). The BCS consists of a hierarchy of locally-tuned interactions that controls the emergence of image segmentation and also detects, enhances and completes boundaries. The interaction of BCS with a feature contour system and an object recognition system, as developed in [23] attempts to attain a unifying precept for form, color and brightness. While the BCS is largely preattentive, the model does allow feedback from the object recognition system to guide the segmentation process.

The BCS consists of several stages arranged in an approximately hierarchical organization. The image to be processed forms the input to the earliest stage Here, elongated and oriented receptive fields called masks are employed for local contrast detection at each image position and each orientation. Thus there is a family of masks centered at each location, and responding to a prescribed region around that location These elliptical masks respond to the amount of luminance contrast over their elongated axis of symmetry, regardless of whether image contrasts are due to differences in textural distribution, a step change in luminance or a smoother intensity gradient. The elongated receptive field makes the masks less sensitive to differences in average contrast in a direction orthogonal to the major axis. However, the penalty for making them sensitive to contrasts in the preferred orientation is the increased uncertainty in the exact locations of contrast. This positional uncertainty becomes acute during the processing of image line ends and corners. The authors assert that all line-ends are illusory in the sense that they are not directly extracted from the retinal image, but are created by some process that generates line terminations. One such mechanism that is hypothesized by them is based on two short-range competitive stages followed by long-range cooperation, as described next.

Firstly, each pair of masks at the same location that are sensitive to the same orientation but opposing direction of contrasts, input to a common cell. The output of such a cell at position (i,j) and orientation *k* is J_{ijk}, which is related to the two directional mask outputs, U_{ijk} and V_{ijk}, by:

$$J_{ijk} = \frac{[U_{ijk} - \alpha V_{ijk}]^+ + [V_{ijk} - \alpha U_{ijk}]^+}{1 + \beta (U_{ijk} + V_{ijk})}, \qquad (11)$$

where the notation $[p]^+$ stands for max(p,0)

These oriented cells are sensitive to the amount of contrast, but not the direction They in turn feed two short-range competitive stages. In the first stage, there is on-center, off-surround competition between a cell and all other cells of the same orientation and within a small neighborhood of that cell. Subsequently, push-pull opponent processes are activated at each position for the second competitive stage, resulting in competition between orthogonally oriented masks at each position. Let w_{ijk} represent the output signal for the cell corresponding to position (i,j) and orientation *k*, and w_{ijK} be the output for the cell at the same location but with orientation orthogonal to *k*, at the end of the first stage. The w_{ijk}s are obtained from:

$$\frac{d}{dt} w_{ijk} = -w_{ijk}(1 + B \sum_{(p,q)\,\varepsilon\, R} BJ_{pqk} A_{pqij}) + I + BJ_{ijk} \qquad (12)$$

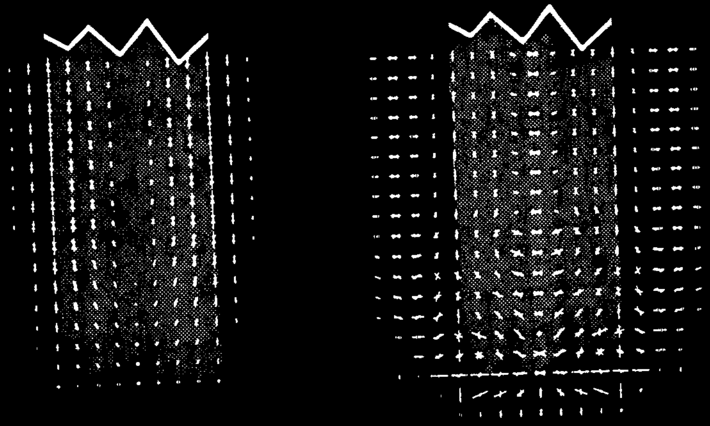
In Eq. 12, I is a tonic input, R a neighborhood of (i,j), and A_{pqij} the inhibitory interaction strength between positions (p,q) and (i,j). The activity potentials y_{ijk} of cell outputs in the second stage are governed by:

$$\frac{d}{dt} y_{ijk} = -Ay_{ijk} + (E - y_{ijk})\, O_{ijk} - y_{ijk} \sum_{m \neq k} O_{ijm}, \qquad (13)$$

where $O_{ijk} = C[w_{ijk} - w_{ijK}]^+$, and A, C and E are constants.

The behavior of the orientation field is shown in Figure 5, in which adjacent lattice points are one unit apart. Each mask has a total exterior dimension of 16 x 8 units Figure 5(b) shows the y_{ijk} responses at the end of the second competitive stage for the same input stimulus The two competitive stages together have generated end cuts as can be seen clearly on comparing with Fig. 5(a). Note that the second competitive stage is tonically active, that is, inhibition of a vertical orientation excites the horizontal orientation at the same position.

The outputs of the second stage are also used for the boundary completion process that involves long-range cooperation between similarly oriented pairs of input groupings This mechanism is able to complete boundaries across regions that receive no bottom-up inputs from the oriented receptive fields, and thus accounts for illusory line phenomena such as the completion of the square edges in a reverse-contrast Kanisza square The process of boundary



(a) Response of oriented masks. (b) Response of second competitive layer

Figure 5: (Adapted from Grossberg & Mingolla [21]) (a) Shows the output of the oriented masks superimposed on the input pattern (shaded area). Lengths and orientations of lines encode the relative sizes of the activations and orientations of the masks at the corresponding positions. 5(b) shows the output of the second competitive stage for the same input as in 5(a).

completion occurs discontinuously across space, using the gating properties of the cooperative cells to successively interpolate boundaries within progressively finer intervals. Unlike a low spatial frequency filter, this process does not sacrifice spatial resolution to achieve a broad spatial range. The cooperative cells used in this stage also provide positive feedback to the cells of the second competitive stage so as to increase the activity of cells of favored orientation and position, thereby providing them with a competitive edge over other orientations and positions. This feedback helps in reducing the fuzziness of boundaries. The detailed architecture, equations and simulation results can be found in [21,22].

A hierarchical neural network for texture segmentation and labelling has been proposed by Dupaguntla and Vemuri [14]. The underlying premise of their approach is that textural segmentation can be achieved by recognizing local differences in texels. The architecture consists of a feature extraction network whose outputs are used by a texture discrimination network. The feature extraction network is a multilayer hierarchical network governed by the BCS theory. The image intensities input is first preprocessed by an array of cells whose receptive fields correspond to a difference of Gaussian filter (see Sec. 2.1), and which follow the feed-forward shunting equations of Grossberg. The output of this array of cells form the input to a BCS system and are processed by oriented masks according to Eq. (11). These masks then feed into the two competitive stages of the BCS theory, and governed by Eq. (12) and (13). However, the long-range cooperative processes described above are not used. Instead, the outputs of the second competitive stage activate region encoding (RE) cells at the next level. Each RE cell gathers its activity from a region of orientation masks of the previous layer, as well as from a neighborhood of adjacent RE nodes of the same orientation.

The activity potential of an RE node is given by the following equation, where the y_{lmk}'s are obtained from the previous layer according to Eq. (13), (l,m) is in the neighborhood of (p,q), and the activation function, *f*, is sigmoidal:

$$\frac{d}{dt} z_{ijk} = - \mu z_{ijk} + \mu \sum_{(p,q)} (f(z_{pqk}) - f(z_{ijk})) + \sum_{(l,m)} y_{lmk} . \tag{14}$$

The RE cells are functionally analogous to the complex cells in the visual cortex, with the intra-layer connections helping to propagate orientation information across this layer of cells.

The outputs (z_{ijk}s) of the feature extraction network are used by a texture discrimination network which is essentially Kohonen's single-layered self-organizing feature map [37]. At each position, there are *T* outputs, one for each possible texture type, which is assumed to be known *a priori*. Model (known) textures are passed through the feature extraction network For a randomly selected position (i,j), the output cell of the texture discrimination network that responds maximally is given the known texture-type label. The weights in the texture discrimination network for that position are adapted according to the feature-map equations. Since these weights are the same for all positions, one can simply replicate the updated weights for all positions. The hierarchical scheme described above has been applied to natural images with good results. However, it is very computationally intensive, since there are cells corresponding to each orientation and position at every hierarchical level.

4.3 Discrete 2-D Gabor Transforms using neural networks.

This section explains how a three-layered network can be used to obtain the expansion coefficients for expressing two-dimensional discrete signals in terms of a set of basis functions, which need not be complete or orthogonal. In particular, a complete 2-D Gabor transform is of interest, because the desirable properties of Gabor functions, elaborated in Section 3.2, lead to viable image segmentation based on clustering the coefficients obtained The approach given below is adapted from Daugman [12].

We wish to represent a two dimensional signal I(x,y), which can be the image intensity values at pixel points (x,y) for example, in terms of a set of 2-D elementary functions, $\{G_i (x,y)\}$ I(x,y) is approximated by H(x,y), which is a linear combination of the elementary functions, i.e.,

$$H(x,y) = \sum_{i=1}^{n} a_i G_i(x,y)$$

If the *n* elementary functions form a complete orthogonal set, then projection coefficients $\{a_i\}$ that lead to an exact representation can be easily determined. Otherwise, one can choose the $\{a_i\}$s so as to minimize the squared error:

$$E = \sum_{x,y} [I(x,y) - H(x,y)]^2 \tag{15}$$

The desired $\{a_i\}$s are obtained by setting $\partial E/\partial a = 0$, which yields a system of *n* simultaneous equations in *n* unknowns:

$$\sum_{x,y} I(x,y) G_i (x,y) = \sum_{x,y} [\sum_{k=1}^{n} a_k G_k(x,y)] G_i (x,y) . \tag{16}$$

If we choose the elementary functions to be a set of Gabor functions, each characterized by the position (x_0,y_0), frequency (u_0,v_0) and scale (a,b) coordinates, then the inner products $G_i \cdot G_j$ will be non-zero in general. For this non-orthogonal case, Eq. (16) could still be solved by algebraic means, but it becomes computationally prohibitive even for moderately sized images.

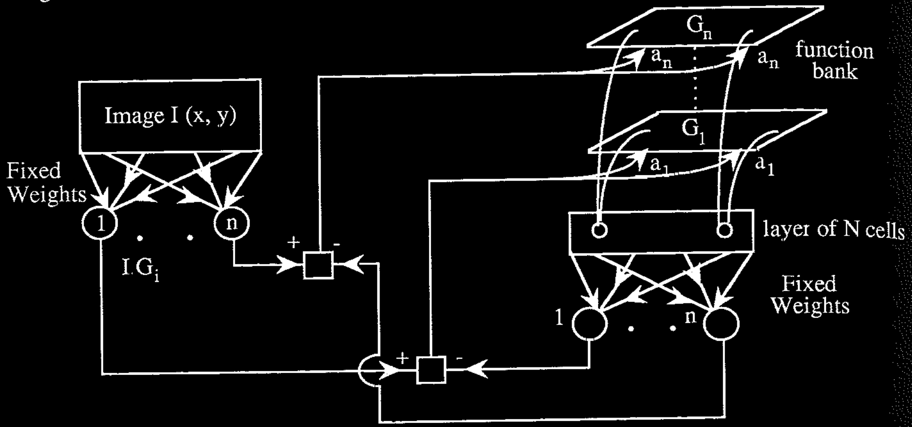


Figure 6 A three-layered network for determining the optimal coefficients of arbitrary image transforms

Fortunately, E given by Eq (15) is quadratic in each of the a_is This means that a unique global minimum for E exists and can be reached by a gradient descent along the error surface expressing this cost function's dependencies on all of the $\{a_i\}$ coefficients and realized using the "neural network" architecture of Fig 6 This network has three layers of cells. All cells perform linear summation of their weighted inputs The efferent connections from the image have fixed weights dictated by the elementary functions chosen, and provide input activation for a layer of n units The output of the ith unit of this layer is simply $I \cdot G_i$, i.e., the inner product of the ith elementary function with the image. This is analogous to the neurophysiological concept of a (linear) neuron's receptive field profile, which refers to the spatial weighting function by which a local region of the retinal image is multiplied and integrated to generate that neuron's response strength.

Let the image be of size N. The second layer has N cells, one for each image position. The cell (x,y) gets input from the n units of the function bank, corresponding to $G_1(x,y)$ through $G_n(x,y)$, through adaptive connection strengths that are the estimated $\{a_i\}$s The outputs of this layer provide input activation to another layer of n cells, through the same set of fixed weights that are used between the image and the first layer The weight change to a_i is Da_i , given by:

$$\Delta a_i = \mu \left(\sum_{x,y} I(x,y)\, G_i(x,y) - \sum_{x,y} \left[\sum_{k=1}^{n} a_k G_k(x,y) \right] G_i(x,y) \right) \qquad (17)$$

Note that the adaptation of weights does not require an external teacher Rather, the control signal arises only from the interlaminar network interactions. From Eq.(15) and (17), it can be seen that $\Delta a_i = -\frac{\mu}{2}\frac{\partial E}{\partial a_i}$, which means that a gradient descent in weight space is approximated by the weight adjustments. At equilibrium, the cost function reaches its minimum. An elaborate discussion of gradient descent is given in Widrow & Stearns [58] Indeed, the architecture of Fig. 6 is really an adaptive linear combiner couched in a "neural" framework. Daugman [12] has used the above architecture with Gabor functions for the G_is to perform image compression and segmentation based on clustering of the expansion coefficients obtained. In the following section, we describe a different approach to textured image segmentation that is based on the multiple channel model of Section 3.3.

4.4 The SAWTA Mechanism.

A cooperative-competitive feedback network called Smoothing, Adaptive Winner-Take-All Network (SAWTA) has been developed recently for performing texture-based segmentation using the same texture model, but with improved results [19]. The network consists of n layers of cells, with each layer corresponding to one Gabor filter, as shown in Fig 7. On the presentation of an image, a 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, as outlined in the preceeding paragraphs 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) Initially, the n cell activations at each point (x,y) are set proportional to the amplitude responses of n Gabor filters.

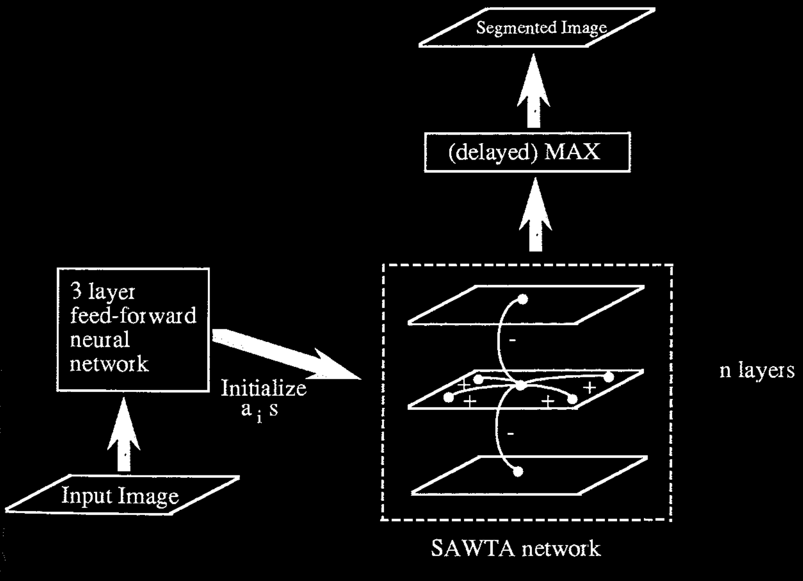


Figure 7: The SAWTA network for segmentation of textured images.

To implement the SAWTA 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 [21] that model on-center off-surround interactions among cells which obey membrane equations [35]. Thus, at each point (x,y), the evolution of the cell in the ith layer is governed by:

$$\tau \frac{d}{dt}(m_i) = - m_i + (A - m_i)\, J^+ - (B + C\, m_i)\, J^- , \tag{18}$$

where J^+, J^- are the net excitatory and inhibitory inputs respectively, and are given by

$$J^+ = \alpha \sum_{(x_n,\, y_n)\, \varepsilon\, R} m_i(x_n, y_n)\, e^{\frac{-[(x - x_n)^2 + (y - y_n)^2]}{2\sigma^2}} \; ; \text{ and } J^- = \sum_{j \neq i} f(m_j (x,y)).$$

Here, R is the neighboring region of support and f is a sigmoidal transfer function. The convergence of a system described by Eq.(4.9) has been shown for the case when the region of support R consists of the single point (x,y). The network is allowed to run for t_n iterations before region assignment is performed using Eq. (9).

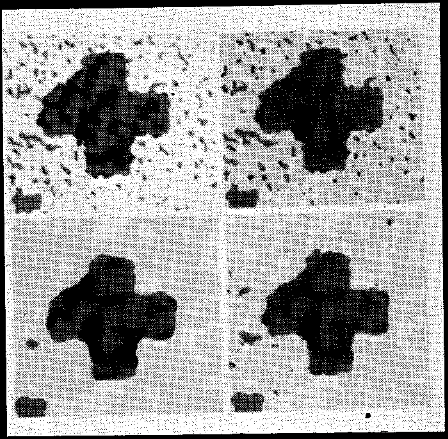


Figure 8: Segmentation of the synthetic texture of Fig. 4(a), using the SAWTA network (clockwise from top left): (a) segmentation after 10 iterations of the SAWTA network; (b) after 10 iterations, but with C = 3; (c) after 50 iterations; (d) after 100 iterations.

Figure 8 shows experimental results using the SAWTA network for segmentation. The 256x256 gray level images are prefiltered using a bank of Laplacian-of-Gaussians to remove high dc components, low-pass phase functions, and to suppress aliasing. Then, only sixteen circularly-symmetric Gabor filters are used to detect narrow-band components. 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 8 shows the segmentation achieved for the synthetic texture of Fig. 4(a). The constants A, B and C in Eq (18) were taken to be 1, 0 and 10 respectively. The activation function used is f(x) = tanh(2x). The results are seen to be superior to that shown in Fig 4(b) or (c), using Eq.(8) or Eq (10). Figure 8(b), (c) and (d) show 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.

The SAWTA network does not require a feature extraction stage as in [14] or computationally expensive masking fields. 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. It has been observed that usually four spatial frequencies are dominant in human visual systems This suggests the use of a mechanism for post-inhibitory response that suppress cells with activation below a threshold and speeds up the convergence of a SAWTA network The adaptive learning network of Kohonen [37] can be used to change both excitatory and inhibitory synaptic strengths (J^+, J^-), in response to a teaching input. Also, the SAWTA network can be easily extended to allow for multiple "winners". Then, it can cater to multicomponent textures, since a region that contains two predominant frequencies of comparable amplitude will not be segmented but rather viewed as a whole.

4.5 An Optimization Framework for Texture Segmentation.

The use of Markov Random Field (MRF) models for modeling texture has been investigated by several researchers [10,8] It can be used to model the texture intensity process as well as to describe the texture labelling process. In this framework, segmentation of textured images is posed as an optimization problem. Two optimality criteria considered in Manjunath *et al* [41] are (i) to maximize the posterior distribution of the texture label field given the intensity field, and (ii) to minimize the expected percentage of misclassification per pixel by maximizing the posterior marginal distribution Corresponding to each criteria, an energy (cost) function can be derived that is a function of MxMxK binary labels, one for each of the K possible texture labels that a pixel in an MxM can take.

The energy function can be minimized through deterministic relaxation using the discrete Hopfield-Tank formulation [27]. In general, an algorithm based on MRF models can be mapped trivially onto neural networks with local interconnections. While the deterministic relaxation algorithm is simple and usually converges in 20-30 iterations, its quality is quite sensitive to initial conditions as it has a penchant for settling into local optima Alternatively, a stochastic algorithm such as simulated annealing can be used to minimize the energy function Indeed, any problem formulated in terms of minimizing an energy function can be given a probabilistic interpretation by use of the Gibbs distribution [48] The two approaches are related in that a mean field approximation of the stochastic algorithm yields the Hopfield equations, with the free parameter, l, being proportional to the inverse of the annealing temperature [28]

For the segmentation problem, a constraint on a valid solution is that each image position should have only one of the K labels "on". This constraint is usually incorporated in a soft fashion by adding bias terms to the energy function Peterson & Soderberg [48] have incorporated the 1-of-K constraint in a Potts glass, and derived a mean field solution for that formulation. It has been shown recently that the alternative of putting global constraints on the set of allowable states in the corresponding stochastic formulation leads to significantly better solutions [61] An iterated hill climbing algorithm that combines fast convergence of the deterministic relaxation with the sustained exploration of the stochastic approach has also been proposed in [41] for the segmentation problem. Here, two-stage cycles are used, with the equilibrium state of the relaxation process providing the initial state for a stochastic learning automaton within each cycle. The relation between neural network techniques and MRFs is explored in detail in Chapter 6 of this book.

5. CONCLUDING REMARKS

Texture can provide useful cues for segmenting scenes into objects, and for determining their shape and surface properties Statistical approaches to texture analysis have only achieved limited success, partly because the texture modelling problem itself is very difficult Neural network approaches to analysis of textured images are amenable to a massively parallel implementation in VLSI, and hold forth the promise of real-time visual processing The recent progress in developing a silicon retina is particularly exciting [45]. Analog VLSI chips described in Mead [45] use networks of resistive grids and operational amplifiers to perform edge detection, smoothing, segmentation, compute optic flow, etc. Such chips have been incorporated in toy autonomous vehicles that can track edges or movements [36]. Further progress towards the development of low-power, real-time vision hardware requires an integrated approach encompassing image modeling, parallel algorithms and the underlying implementation technology.

6. References

1 J A. Anderson and E. Rosenfeld, (eds). *Neurocomputing: Foundations of Research.* MIT Press, Cambridge, Mass , 1988.

2 M. A. Arbib and A. R. Hanson. *Vision, Brain and Cooperative Computation.* MIT Press, Boston, MA, 1987.

3 R Bajcsy. Computer description of textured surfaces. In *Proc Int Joint Conf. Artif. Intell.*, Stanford, CA, Aug 20-23, 1973.

4 D. Ballard and C.M Brown. *Computer Vision* Prentice-Hall, Englewood Cliffs, NJ, 1982.

5 A C. Bovik, M Clark and W. S. Geisler. Multichannel texture analysis using localized spatial filters *IEEE Trans. Pattern Anal Machine Intell.*, vol PAMI-12, pp. 55-73, Jan. 1990.

6 J. Canny. A computational approach to edge detection. *IEEE Trans. on PAMI*, 8:679-698, 1986

7 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.

8 R. Chellappa and S Chatterjee. Classification of textures using gaussian-markov random fields. *IEEE Trans ASSP*, ASSP-33:959-963, 1985.

9 J M Coggins and A. K Jain A spatial filtering approach to texture analysis. *Pattern*

Recog. Lett , vol. 3, pp. 195-203, 1985.

10 G.R Cross and A. K Jain. Markov random field texture models. *IEEE Trans on PAMI*, PAMI-5:25-39, Jan. 1983.

11 J G. Daugman Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters *J. Opt Soc Am* 2, 1160-1169, 1985.

12 J. G. Daugman. Complete discrete 2-D Gabor transforms by neural networks for image analysis and compression. *IEEE Trans Acoust. Speech, Signal Processing*, 36, 1169-79, 1988.

13 L. S. Davis and A. Rosenfeld. Cooperating processes for low-level vision: A survey. *Artificial Intelligence*, 3:245-264, 1981.

14 N R Dupaguntla and V. Vemuri. A neural network architecture for texture segmentation and labeling. In *Proc International Joint Conference on Neral Networks*, pages 127-144 (I), June 1989.

15 O. D. Faugeras. Texture analysis and classification using a human visual model In *Proc Int Conf. Pattern Recogn.*, Kyoto, Japan, Nov. 7-10, 1978.

16 M A. Fischler and O Firschein (eds.) *Readings in Computer Vision* Morgan Kaufmann, 1987

17 D. Gabor Theory of communication, *J Inst. Elect. Eng.*, vol. 93, pp 429-457, 1946.

18 J Ghosh and K. Hwang. Mapping neural networks onto message-passing multicomputers *J of Parallel and Distributed Computing*, 6:291-330, April, 1989.

19 J. Ghosh, N Gopal and A. C. Bovik Textured image segmentation using localized receptive fields. In *Proc Int'l Joint Conf Neural Networks,* Washington, DC, Jan. 15-19, pp. 283-286, 1990

20 N. Gopal, A. C Bovik and J. Ghosh. Multiple channel surface orientation from texture. In *SPIE/SPSE Conf. Human Vision Elect Imaging: Models, Methods, Appl.,* Santa Clara, CA, Feb. 11-16, 1990.

21 S. Grossberg and E. Mingolla Neural dynamics of perceptual grouping: Textures, boundaries and emergent segmentations *Perception and Psychophysics*, 38:141-171, 1985

22 S. Grossberg and E. Mingolla Computer simulations of neural networks for perceptual psychology. *Behavior Research: Methods, Instruments and Computers*, 18:601-607, 1986.

23 S. Grossberg. Cortical dynamics of three-dimensional form, color and brightness perception, I: Monocular theory *Perception and Psychophysics,* 41:87-116, 1987

24 R. M Haralick. Statistical and structural approaches to texture. In *Proc IEEE*, vol. 67, pp. 786-804, 1979.

25 D. O. Hebb. *Organization of Behavior*. Wiley, New York, 1949.

26 H.V.B. Hirsh and D.N Spinelli. Visual experience modifies distribution of horizontally and vertically oriented receptive fields in cats. *Science*, 168:869-871, 1970.

27 J. J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings National Academy of Science*, 79:2554-2558, Apr 1982.

28 J. J. Hopfield and D. W. Tank. "Neural" Computation of decisions in optimization problems. *Biol. Cybernetics*, 52:141-152, 1985.

29 B K. P. Horn. *Robot Vision.* Cambridge: The MIT Press, 1986.

30 D. H. Hubel and T. N Weisel. Receptive fields of single neurons in the cat's striate cortex. *J. Physiol. London*, vol. 148, pp 574-591, 1959.

31 D. H Hubel and T. N. Wiesel. Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex. *J. Physiol. London*, vol 160, pp. 106-154, 1962

32 D.H. Hubel and T N Wiesel Receptive fields, binocular interaction and functional architecture in the cat's visual cortex *Jl. of Physiology,* 160:215-243, 1968.

33 R. A. Hummel and S. W. Zucker. On the foundations of relaxation labelling processes *IEEE Trans on Pattern Anal and Machine Intelligence,* PAMI-5:267-287, May 1983

34 B. Julesz and J R Bergen Textons, the fundamental elements in preattentive vision and perception of textures. *Bell Syst. Tech. J.*, vol. 62, pp. 1619-1645, 1983.

35 E. Kandel and J Schwartz. *Principles of Neuroscience*. Elsevier Publishing, New York, 1985.

36 C. Koch, W. Bair, J. G. Harris, T. Horiuchi, A. Hsu, and J. Luo. Real-Time Computer Vision and Robotics using analog VLSI circuits In *Neural Information Processing Systems, II*, (ed D. Touretzky), pp. 750-757, 1990

37 T Kohonen. *Self-Organization and Associative Memory*. Springer-Verlag, Berlin, 3rd Ed. 1989.

38 K I. Laws. Textured image segmentation *Ph D. dissertation*, Dep. Elect Eng., Univ. Southern Califonia, 1980.

39 R. Linsker. From basic network principles to neural architecture *Proc. National Academy of Sciences, USA*, 83:7508 -7512, 8390 - 8394, 8779 - 8783, 1988.

40 C. van der Malsburg. Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, 14:85-100, 1973.

41 B S Manjunath, T. Simchony, and R. Chellappa. Stochastic and deterministic networks for texture segmentation. *IEEE Trans on Acoustics, Speech and Signal Processing*, 38:1039-1049, June 1990.

42 S. Marcelja. Mathematical description of the responses of simple cortical cells, *J Opt. Soc. Am.* 70, 1297-1300, 1980.

43 D. Marr. *Vision*. W. H. Freeman Publishing Co., San Francisco, 1982.

44 D. Marr and E. Hildreth. Theory of edge detection. *Proc. Roy Soc London B*, vol. 207, pp. 187-217, 1980.

45 C.A. Mead. *Analog VLSI and Neural Systems*. Addison-Wesley: Reading, MA., 1989.

46 B. A. Olshausen. A survey of visual preprocessing and shape representation techniques. *Technical Report*, RIACS TR-38 35, 1988.

47 A. Pantle and R. Sekular Contrast response of human visual mechanisms sensitive to orientation and direction of motion, *Vision Res*, vol 9, pp. 397-406, 1970.

48 G. Parisi. *Statistical Field Theory*. Addison-Wesley, Reading, Mass., 1988.

49 C. Peterson and B. Soderberg. A new method for mapping optimization problems onto neural networks. *International Journal of Neural Systems*, Vol 1., 3-22, March 1989.

50 S Pinker (ed.) *Visual Cognition*. MIT Press, Cambridge, Mass., 1985

51 A Rosenfeld (ed). *Man and Machine Vision II*. Academic Press, 1986.

52 A. Rosenfeld Image analysis and computer vision *Computer Vision, Graphics, and Image Processing*, 42:234-293, 1988.

53 M R. Turner Texture discrimination by Gabor functions. *Biol Cybern.*, vol. 55, pp. 71-82, 1986.

54 D.C Van Essen. Functional organization of primate visual cortex. In *Cerebral Cortex* Vol.3, pages 259-329. Plenum Press, 1985.

55 H. Voorhees and T Poggio. Detecting textons and texture boundaries in natural images. In *Proc First Int. Conf. Comput. Vision*, London, England, June 8-11, 1987

56 D. Walters. Selection of image primitives for general purpose visual computing. *Computer Vision, Graphics, and Image Processing*, 37:261-298, 1987.

57 H. Wechsler Texture analysis—A survey. *Signal Processing*, vol. 2, pp. 271-282, 1982.

58 B. Widrow and S.D. Stearns. *Adaptive Signal Processing*. Prentice-Hall, Englewood Cliffs, N J., 1985.

59 H.R. Wilson and J.R. Bergen. A four mechanism model for spatial vision *Vision Research*, 19:19-32, 1979.

60 R. Wilson and M Spann. Finite prolate spheroidal sequences and their applications II: Image feature description and segmentation. *IEEE Trans. Pattern Anal Machine Intell.*, vol. 10, pp. 193-203, Mar. 1988.

61 A L. Yuille. Generalized models, statistical physics, and matching problems *Neural Computation*, 2:1-24, 1990.

Artificial Neural Networks and Statistical Pattern Recognition
Old and New Connections
I.K. Sethi and A.K. Jain (Editors)
© 1991 Elsevier Science Publishers B.V. All rights reserved

Markov Random Fields and Neural Networks with Applications to Early Vision Problems

A. Rangarajan, R. Chellappa and B. S. Manjunath

Signal and Image Processing Institute, Powell Hall of Engineering (PHE) 306, University of Southern California, Los Angeles, CA 90089-0272

Abstract

The current resurgence of interest in Neural Networks has opened up several basic issues. In this chapter, we explore the connections between this area and Markov Random Fields. We are specifically concerned with early vision problems which have already benefited from a parallel and distributed computing perspective. We explore the relationships between the two fields at two different levels of a computational approach. Applications highlighting specific instances where ideas from the two approaches intertwine are discussed.

1 INTRODUCTION

Markov Random Fields (MRFs) and Neural Networks (NNs) have a history dating back over thirty years. While MRFs primarily serve as tools for model building, NNs are much more ambitious; their applications range from computing least square estimates to building parallel computers. However, both fields offer paradigms for the construction of early vision modules and there are numerous connections between them.

We are primarily interested in exploring the relationships between the two fields as they relate to the construction of early vision modules. In this regard, we suggest that the frameworks and principal ideas be compared at different levels of a computational approach. Marr [1] was the first to suggest that there are three levels of a computational approach; computational, algorithmic and hardware. We feel that comparisons at the first two levels are appropriate and illuminating for this audience. In review, the computational level concerns itself with *what* is being computed. The specific scheme of computation is not of interest, only the goal of computation is important. The algorithmic level concerns itself with *how* the goal is computed. An example of the goal of computation is the Traveling Salesman Problem and an example of a method of solution is the Hopfield network [2]. We will not be discussing the relationships at the level of hardware implementation; it is important albeit premature.

MRFs are an important class of stochastic models and have been applied to problems like image estimation and texture segmentation. Once the model variables have been chosen, an MRF is completely specified by the joint distribution over these variables

The important characteristic of MRFs in early vision is that the conditional densities are dependent only on a small neighborhood surrounding the variable of interest. When this is the case, the conditional densities are called *local characteristics*. In order to perform feature extraction, the joint density is usually specified over a set of variables which are closely related to the data and a further set of attribute variables which categorize the data. The attribute variables are termed *unobservable* since they are not part of the observed data. The joint density of the data variables (computed by integrating out the unobservables) is usually fully connected [3]. This property suggests that fully connected, non trivial models can emerge from an intuitive choice of attributes and local characteristics, thereby allowing us to create hierarchical models which are capable of extracting higher order features. The higher order features constitute an efficient *internal* representation. The unobservables are specified by the designer rendering the internal representation immutable. The goal of computation is the minimum mean square error estimate (MMSE), maximum *a posteriori* (MAP) estimate or in general any estimate based on minimum expected cost. At the algorithmic level, there are several methods, both deterministic and stochastic, of solving the problem. However, in addition to the state estimation problem, we have to solve the parameter estimation problem. The joint distribution is a function of these parameters which have to be specified *apriori* before the cost minimization is done. The long-term goal is rapid feature extraction and classification with continual adaptation to incoming data.

There are a large number of NN related models. The basic idea drawn from biology and neurobiology is to have networks with neuron-like processing elements with simple dynamics and massive interconnections which are capable of parallel, global computation and learning internal representations. In most neural net schemes, the aim is to let the internal representation of the data emerge from a set of *hidden* units which are connected to the data. The connections serve as the long term memory and are modified in order to better approximate the distribution of the incoming data. Higher order statistics are represented by interconnections (which are second-order) and hidden units. In order to take decisions, neuronal dynamics have to be non-linear [4]. The goal of computation is decided by a teacher who then trains the network (with a given number of hidden units) on a set of training samples [4]. There is no need to restrict the interconnections to second-order. Higher-order interconnections can more closely approximate the incoming distribution [5]. In self-supervised schemes, self-organization plays the role of the external teacher [6]. Category formation and knowledge representation are now machine driven with no supervision. The goal of the computation is for the machine to discover the intrinsic complexity (or the information needed for a minimal description) of the data [7]. The algorithmic level problems are the choice of the number of hidden units, the learning algorithm etc. The long term goal once again is rapid feature extraction and classification with continual adaptation to incoming data.

The two paradigms are strongly interrelated. Several ideas are common due to mutually dependent co-origination a few years ago. We wish to re-examine the basic ideas and suggest the different levels at which the two approaches can continue to benefit each other. Section 2 is devoted to this issue. In Section 3, we present two applications highlighting some of the ideas presented in Section 2. Conclusions are presented in Section 4.

2 RELATIONSHIP BETWEEN THE TWO FIELDS

2.1 PRELIMINARIES

In this section, we examine the basic ideas in MRFs and NNs.

2.1.1 MARKOV RANDOM FIELDS

An MRF can be completely specified by the joint distribution over the variables We denote the data by **Y** Associated with the data, we have the process **X** and in addition, we have the attribute process **L** One of the most interesting aspects of MRFs is that the joint distribution is an MRF if and only if it is also Gibbsian [3]. The Gibbs distribution is written as follows

$$\mathcal{P}(\mathbf{X} = \mathbf{x}, \mathbf{L} = \mathbf{l} | \mathbf{Y} = \mathbf{y}) = \frac{1}{Z} \exp(-\beta \mathcal{H}(\mathbf{x}, \mathbf{l}, \mathbf{y})) \tag{1}$$

where Z is the *partition function*, $\beta = \frac{1}{T}$ and T is the temperature, and $\mathcal{H}$ is the *energy function* defined over all the processes. The partition function is a function of the observed data $\mathbf{y}$ and the parameters involved in the specification of $\mathcal{H}$ Adopting a Bayesian viewpoint, it is easy to switch between the various distributions. Equation (1) is the *posterior* distribution arising from the *degradation* model $\mathcal{P}(\mathbf{Y} = \mathbf{y} | \mathbf{X} = \mathbf{x}, \mathbf{L} = \mathbf{l})$ and the *prior* distribution $\mathcal{P}(\mathbf{X} = \mathbf{x}, \mathbf{L} = \mathbf{l})$ The prior distribution encodes our beliefs about the processes **X** and **L** and is Gibbs The degradation model is specified by the (usually) known transformation from the representation to the observed data and is also Gibbs. In principle, the recipe for inference is clear once these distributions have been specified. The parameters are estimated using maximum likelihood (ML) on the marginal density of the data (which is a function of the parameters) and then the expected cost is minimized with respect to **X** and **L** In practice, both problems (parameter estimation and cost minimization) are intractable in their pure form. For example, if cost minimization reduces to finding the MAP estimate, then the problem is usually NP-complete and only simulated annealing (SA) is guaranteed to asymptotically reach the optimum solution In the case of parameter estimation, ML cannot be performed due to the intractability of the distribution of **Y**

In recent years, several alternatives have been proposed It is possible to design efficient deterministic algorithms that obtain good sub-optimal MAP or MMSE estimates. For instance, the Iterated Conditional Mode (ICM) [8] algorithm iteratively maximizes the conditional density of each variable with the others held fixed. Similarly the Iterated Conditional Expectations (ICE) algorithm [9] iteratively computes the conditional expectation of each variable with the others held fixed ICM and ICE yield good, sub-optimal MAP and MMSE estimates respectively Both methods exploit the Markov structure In addition, continuation methods [10] have been developed which are akin to some form of deterministic annealing since they involve the gradual change of a control parameter. The problem of parameter estimation has been partially solved by the introduction of pseudo-likelihood (PL) [11] techniques which maximize the product of the conditional densities with respect to the unknown parameters. This avoids the intractable problem of having to compute the partition function. Unfortunately, PL requires prior knowledge of **X** and **L** which is unavailable. The well known Expectation-Maximization (EM) algorithm [12] has been adapted to jointly solve for the parameters and states.

2.1.2 NEURAL NETWORKS

There are several models of learning and perceptual inference in the Neural Network literature. In this section, we explore the fundamental ideas of the some of these models and of the Boltzmann Machine (BZM) [13] in particular. The BZM has very close ties to the general area of MRFs.

The BZM consists of two phases, a fixed phase and a free phase. As previously mentioned, internal representations are generated by a set of hidden binary units and learning is accomplished by modifying the interconnections between the units. In the fixed phase, the environmental patterns clamp the input/output units and the hidden units are relaxed using SA. In the free phase, all units (sometimes the inputs are clamped) are relaxed using SA. Co-occurrence statistics are collected in both phases and the weights are modified based on the difference between the statistics. It can be shown that this kind of weight modification corresponds to gradient descent on the Kullback information gain [14] which measures the distance between the distributions generated during the fixed and free phases. In essence, the free phase mimics the distribution over the incoming patterns and the weights are modified in order to better approximate the incoming distribution. The order in which the patterns are presented to the system is unimportant, but obviously their frequency is. A pure implementation is infeasible in vision problems due to the excessive computational requirements of the SA algorithm.

The most important alternative that has emerged so far is the Mean Field Theory (MFT) Learning Algorithm [15]. MFT involves tracking the expected value of the units instead of generating their values using a probabilistic (heat bath) algorithm. A very interesting point in relation to all that has already transpired is that MFT exploits an approximation to the partition function (for binary units) that was first formulated in statistical physics. With this approximation in place, a deterministic relaxation algorithm results in place of SA and the equation for the weight change is left intact. MFT has close ties to ICE in the context of MMSE estimation since the expected value of the state variables can be obtained from the partition function. Recall that the MMSE estimate involves computing the conditional expectations of the variables given the data.

The BZM is a supervised pattern classifier. There exist several alternatives to classical unsupervised classifiers in the NN literature. The most notable are Adaptive Resonance Theory (ART) and its variants [6], the Neocognitron [16] and most competitive learning models [4]. The key in most of these models is that learning and category formation are conducted by learning rules which are not obtained by matching the internal representation with an environmentally imposed set of associations. Instead, the learning rules are specified as a dynamical system on the weights. Unsupervised learning results in an organization of the data into categories or features. This process is called self-organization since the system discovers the categories without supervision.

2.2 RELATIONSHIP AT THE COMPUTATIONAL LEVEL

The key differences emerging at the computational level are as follows. MRFs rely heavily on conditional distributions whose local characteristics have small neighborhoods. Also, the models are usually adopted in advance and held fixed In addition, the models are not restricted to second-order distributions. There is no restriction on the range space of the variables. In NNs, the network is typically fully connected with very simple neuron-like digital processing elements In the BZM, only second-order connections are allowed and it is hoped that adequate representation emerges from a reasonable number of hidden units. Fixed models are not specified; the network learns from a set of examples In the case of unsupervised learning, only the learning rules are specified in advance. These learning rules are heuristically specified or are derived from further principles in some cases.

However, there are several common themes. Specifically in BZM, the distributions (incoming and generated) are Gibbsian. There also exists a close analogy between the hierarchical attribute process **L** in the MRF setup and the hidden units. This should come as no surprise since the inspiration for the hidden units did come from the area of Hidden MRFs (HMRF) [17] The clear difference here is that NNs implicitly build distributed, internal representations using hidden units and interconnections, whereas MRFs build explicit representations using attribute variables and model parameters The improvement of the model in NNs arises through learning where the long term memory (in the interconnections) is modified taking into account all past and present exchanges between the network and the environment. This is analogous to parameter estimation in MRFs if it is not performed off-line but adaptively as more information is present to the system The EM algorithm is an archetype of adaptive parameter estimation

MRFs and NNs answer the question of what is to be computed in different ways. While MRFs exploit the structure of the local characteristics, NNs mostly use fully connected networks with second-order connections Both paradigms build internal representations on hidden (unobservable) units, MRFs explicitly and NNs implicitly Adaptation to new data is provided for in both models with NNs incorporating learning and MRFs performing adaptive parameter estimation

In MRFs, prior distributions typically constrain the number of all possible distributions and are sometimes counter-intuitive A proper choice of attribute variables and prior distributions is crucial to the success of MRFs. Important areas where NNs can be useful in MRFs are adaptive parameter and state estimation. Similarly, NNs can benefit from the results of MRF based modeling The choice of the number of hidden units can be made more assuredly with the knowledge of explicit modeling using MRFs Another important topic is higher-order connections MRFs utilize local, higher-order connections to obtain better models Although, in principle, hidden characteristics can be approximated using hidden units (integrating out the hidden units generates higher-order distributions), it is not clear how well hidden units and second-order connections perform with respect to this point in vision tasks More work needs to be done with higher-order, partially connected NNs [5, 18]

We now move to the algorithmic level of description

2.3 RELATIONSHIP AT THE ALGORITHMIC LEVEL

The main problems in MRFs can be divided into two sub-problems; state estimation and parameter estimation. Once the criteria have been imposed on the system, state estimation is usually cast into a MAP or MMSE problem or their variants. The impetus in MRFs dramatically increased with the introduction of SA [3] and it has not waned even after the computational limitations of SA have been pointed out. There are several alternatives to SA especially for MAP estimation. These range from general purpose techniques like ICM (not confined to MAP) to deterministic annealing techniques or continuation methods. Interestingly enough, Hopfield networks and MFT (the two are equivalent for MAP estimation) have been quite successful in early vision problems like image estimation and surface reconstruction. It is important to realize that MAP estimation performed in isolation is not the goal. We also need to perform parameter estimation. Due to the intractability of the partition function, EM like techniques have become popular.

Deterministic networks for MAP estimation have borrowed heavily from neural networks, MFT in particular. MFT as applied to MAP estimation uses approximations to the partition function and ends up minimizing a cost function which in some ways is a smoother version of the original cost function. As a control parameter is varied, the original cost function is better approximated. Also, new methods of solving the *winner-take-all* (WTA) problem (which also suffers from local minima) have emerged [19]. The impetus in NNs began with a novel approach to the TSP problem. There is tremendous potential in further applications of NN related ideas for MAP and MMSE estimation of MRFs. In Section 3, we present different NN related approaches for solving well known problems in early vision.

The parameter estimation problem can be successfully tackled using PL techniques and the EM algorithm. The consistency of these estimators has been proven for very general Gibbs distributions [20]. BZM learning algorithms are very closely related to ML techniques. This is because the BZM minimizes the Kullback information gain [14] which is related to ML. The difference is that ML specifies the parameters by maximizing the distribution of the data whereas the Kullback information gain minimizes the distance between the distributions of the fixed and free phases with respect to the weights.

3 APPLICATIONS

In this section, we concentrate on two applications which illustrate some of the comments made in Section 2.3.

The first application is in image estimation and segmentation. The problem can be succintly stated as follows. We wish to obtain an estimate of a noisy image which at the same time yields a convenient representation in terms of piecewise, homogeneous regions. Adopting the MRF viewpoint, the degradation is assumed to be corruption by additive noise and the prior belief is that the image is composed of piecewise, homogeneous regions. Then the goal of the computation is conveniently expressed as a MAP estimation problem. The noisy image is the **D** process, the prior is the **F** process and the boundaries are the attribute process **L**. The MAP estimation problem is NP-complete and we suggest continuation methods which find good, sub-optimal solutions We show that our algorithm is equivalent to those of other reseachers when several constraints on the boundary process **L** are removed (the uninteresting case) Specifically, our continuation method has close ties to SA, MFT and other non NN related methods

The second application is in texture segmentation The problem of interest is to locate the boundaries of the textures present in an image As in the case of image estimation, we set up the problem as a MAP estimation of the texture labels **L** when we know the texture intensities **Y**. This problem is also NP-complete and we present several algorithms, stochastic and deterministic which solve the problem Of particular interest is a learning algorithm adapted from the stochastic approximation literature which refines the optimization algorithm by experience Comparisons with SA and others are provided which facilitate choice of any algorithm in a particular situation

3.1 IMAGE ESTIMATION AND SEGMENTATION

3.1.1 A GENERALIZED FRAMEWORK FOR IMAGE ESTIMATION AND SEGMENTATION

When Gibbs distributions are used for prior and degradation models, the posterior distribution is still Gibbs. The attractive feature of a Gibbs distribution is in the convenience of its specification; it can be completely specified by an energy function. The energy function is defined over the intensity (**f**), the vertical (**v**) and horizontal (**h**) line processes

$$\begin{aligned}\mathcal{H}(\mathbf{f},\mathbf{v},\mathbf{h}) &= \sum_{\{i,j\}}(f(i,j)-d(i,j))^2 + \sum_{\{i,j\}}(\lambda^2 f_x^2(i,j)(1-v(i,j)) + \alpha v(i,j)) \\ &+ \sum_{\{i,j\}}(\lambda^2 f_y^2(i,j)(1-h(i,j)) + \alpha h(i,j)) + \mathcal{H}_c(\mathbf{v},\mathbf{h}) \qquad (2)\end{aligned}$$

where **d** is the observed data, $v(i,j), h(i,j) \in \{0,1\}$, $f_x(i,j) \stackrel{\mathrm{def}}{=} f(i+1,j) - f(i,j)$ and $f_y(i,j) \stackrel{\mathrm{def}}{=} f(i,j+1) - f(i,j)$.

When the $\mathcal{H}_c(.)$ term (corresponding to prior knowledge of image contours) is absent, this energy function reduces to the popular *weak membrane* [21] The term weak membrane arises from the physical nature of this energy function If discontinuities are absent, the reconstruction would be like a membrane which is continuous everywhere Our objective is to construct a continuation method which finds good suboptimal MAP estimates. The energy function parameters are λ and α

The weak membrane energy function can be written in a more compact form.

$$\mathcal{H}(\mathbf{f}) = \sum_{\{i,j\}}(f(i,j)-d(i,j))^2 + \sum_{\{i,j\}} g^*(f_x(i,j)) + \sum_{\{i,j\}} g^*(f_y(i,j)) \qquad (3)$$

where the new function $g^*(f_p)$ arises from the elimination of the line processes and f_p is the generic symbol used for the intensity gradient The g^* function is written as;

$$g^*(f_p) = \begin{cases} \lambda^2 f_p^2 & \lambda^2 f_p^2 < \alpha \\ \alpha & \lambda^2 f_p^2 \geq \alpha \end{cases} \qquad (4)$$

We also define $g_z^*(f_p, z) \overset{def}{=} \lambda^2 f_p^2(1-z) + \alpha\, z,\ z \in \{0,1\}$ where z is the generic symbol for the line process.

The energy function (3) as it stands is non-convex due to the nature of the g^* function. Several researchers [9, 10, 21] have proposed a variety of continuation methods or convex formulations to deal with this problem. A continuation method essentially tracks minima through the variation of a control parameter; the original energy function is increasingly closely approximated during this variation. All of these approaches can be conceptually synthesized by replacing the g^* function by either a solitary g function or by a sequence of $g^{(k)}$ functions; the integer k is the index of the sequence. Henceforth, we will refer to this sequence simply by the g function since our generalized framework is valid for all members of a given sequence.

We define a new sequence of $g_u^{(k)}$ functions which are related to the old sequence $g^{(k)}$ as follows

$$g_u(f_p, u) = g(u) + \frac{g'(u)}{2u}(f_p^2 - u^2) \tag{5}$$

where u is a new process.

The new sequence of g_u functions is derived from the $g^{(k)}$ sequence using the basic idea that elimination of the u processes in g_u should yield g.

Examining $g_u(f_p, u)$ as a function of u^2, we get

$$g_s(t, s) = g_2(s) + g_2'(s)(t - s) \tag{6}$$

where $t = f_p^2$, $s = u^2$ and $g_2(u^2) = g(u)$.

Obviously, $g_s(t, s)$ is just a Taylor series expansion in t around s truncated at the first term. The interesting point is that both t and s emerge as full-fledged processes on which relaxation is performed. Considering $g_s(t, s)$ as a function of s, we can find the minimum with respect to s keeping t constant.

$$\frac{\partial g_s(t, s)}{\partial s} = g_2''(s)(t - s) = 0 \tag{7}$$

One of the solutions is $s = t$ which can be reformulated as $u^2 = f_p^2$. When this is substituted back into (6), we obtain

$$g_s(t, t) = g_2(t) = g(f_p) \tag{8}$$

The sufficient condition for the minimum is

$$g_2''(t) < 0 \tag{9}$$

Interpreting our technique as a *first* order Taylor series expansion giving rise to a new process s has important consequences. The unobservable process s emerges from the Taylor series expansion. The condition for a minimum to exist at $s = t$ requires $g_2(t)$ to be concave. When s is eliminated dynamically by resubstitution, we obtain the original function $g_2(t)$. Constraints can be added in the s space. The consequence of this result is that the sequence $g_u^{(k)}$ and $g^{(k)}$ are equivalent when no further constraints are added in the space of u. Further constraints aid in the organization of the u process which as we show in Section 3.1.2 has the properties of a boundary process.

3.1.2 RECOVERY OF THE LINE PROCESS

The relationship $t = s$ or $u = f_p$ immediately confers the notion of a gradient upon u. We refer to u as the gradient (GRAD) process The line process can be recovered from the GRAD process keeping in mind that equal, positive and negative values of u must map to the same points in z We find it convenient to first transform the energy function using $u_{GM} = |u|$, the Gradient-Magnitude (GMAG) process. The criteria for a minimum w r t u_{GM} become, $u_{GM} = |f_p|$ and $(g'(u_{GM}) - u_{GM}g''(u_{GM})) > 0$ The sufficient condition arises from rewriting the concavity condition in terms of u_{GM}. A transformation from the GMAG process to the line process $z = z(u_{GM})$ can be obtained. The transformation should not cause the formation of spurious minima. Every minimum of u_{GM} should be a minimum of z and vice versa It is easy to show that this condition is satisfied when $z = z(u_{GM})$ is monotonic and $\frac{dz}{du_{GM}}$ is finite.

We suggest two transformations; $z_1 = 1 - l_1(u_{GM})$, and $z_2 = l_2(u_{GM})$ The monotonicity condition requires

$$\frac{dz_1}{du_{GM}} = \frac{(g'(u_{GM}) - u_{GM}g''(u_{GM}))}{2\lambda^2 u_{GM}^2} > 0 \tag{10}$$

for the first transformation, and

$$\frac{dz_2}{du_{GM}} = \frac{1}{\alpha}(g'(u_{GM}) - u_{GM}g''(u_{GM})) > 0 \tag{11}$$

for the second Note that the relation $(g'(u_{GM}) - u_{GM}g''(u_{GM}))$ also shows up in the sufficient condition for a minimum. Combining the conditions for monotonicity and for a minimum to exist, we get

$$g'(u_{GM}) - u_{GM}g''(u_{GM}) > 0 \tag{12}$$

Each of the transformations suggested have corresponding energy functions that are now defined over the intensity and the *analog line* processes.

3.1.3 A CONTINUATION METHOD USING THE GMAG PROCESS

The continuation method used is based on Blake and Zisserman's Graduated Non-Convexity (GNC) algorithm [21] The sequence of g functions used here is derived from the GNC criterion The first function in the sequence is constructed to make the energy function convex with respect to the intensities Successive g functions push the energy function closer to the weak membrane. The functions are created out of piecewise polynomials and are gradually modified by a control parameter c until the function g^* is reached. The g function sequence is shown below

$$g^{(k)}(u) = \begin{cases} \lambda^2 u^2 & 0 \le |u| \le q \\ \alpha - \frac{c}{2}(r - |u|)^2 & q < |u| < r \\ \alpha & |u| > r \end{cases} \tag{13}$$

where $c = c_* 2^k$, $k = 0, 1,$ and $r^2 = \alpha(\frac{2}{c} + \frac{1}{\lambda^2})$, $q = \frac{\alpha}{\lambda^2 r}$ c_* is the initial value of the control parameter The other parameters q and r arise as a consequence of creating a convex energy function (corresponding to $c = c_*$)

Proceeding as outlined above we can obtain the two functions $l_1(u_{GM})$ and $l_2(u_{GM})$. The relations can be simplified by modifying the definition of the GMAG process

The GMAG process u_{GM} is now defined over the interval $[q, r]$ Note that the interval does not stay fixed but keeps shrinking as c is increased. Now the two functions $l_1(u_{GM})$ and $l_2(u_{GM})$ can be written as

$$l_1(u_{GM}) = \frac{c}{2\lambda^2}\frac{(r - u_{GM})}{u_{GM}},\ l_2(u_{GM}) = \frac{c}{2\lambda^2}\frac{(u_{GM} - q)}{q},\ q \leq u_{GM} \leq r \tag{14}$$

We check if relation (12) is satisfied.

$$g'(u_{GM}) - u_{GM}g''(u_{GM}) = 2cr > 0 \tag{15}$$

We drop the subscript on u_{GM} since the GRAD process plays no role in subsequent discussions. The line process can be obtained from either of the two transformations.

3.1.4 THE GENERALIZED GRADUATED NON-CONVEXITY ALGORITHM

We now suggest introducing interactions on the GMAG process All line interactions can be transfered to the GMAG domain

We choose two basic kinds of interaction terms This has been inspired by the work of Geiger and Girosi [9] and by Geiger and Yuille [19] Consider the following modification of (2)

$$\begin{aligned}\mathcal{H}_{G^2NC} &= \sum_{\{i,j\}} \Big((f(i,j) - d(i,j))^2 + (g_u(f_x(i,j), u_v(i,j)) + g_u(f_y(i,j), u_h(i,j))\Big)\\
&-\frac{c}{2}\epsilon_1\sum_{\{i,j\}}(u_v(i,j) - q)(u_v(i,j+1) - q) + \frac{c}{2}\epsilon_2\sum_{\{i,j\}}(u_v(i,j) - q)(u_v(i+1,j) - q)\\
&-\frac{c}{2}\epsilon_1\sum_{\{i,j\}}(u_h(i,j) - q)(u_h(i+1,j) - q) + \frac{c}{2}\epsilon_2\sum_{\{i,j\}}(u_h(i,j) - q)(u_h(i,j+1) - q)\end{aligned} \tag{16}$$

where all the indices are analogous to the earlier line process case Physically, we are trying to decrease the penalty on a vertical (horizontal) line if its vertical (horizontal) neighbors are "on" and increase the penalty on a vertical (horizontal) line if its adjacent vertical (horizontal) neighbors are "on". This corresponds to encouraging hysteresis which leads to the formation of unbroken contours and non-maximum suppression which prevents the formation of multiple edges and adjacent parallel lines The penalty is controlled by the parameters ϵ_1 and ϵ_2

We now proceed with the algorithm itself The improved line process in the GNC formulation prompted us to call this algorithm the Generalized GNC or G^{2}NC algorithm We first solve for u keeping the intensities fixed. Now, u becomes a function of the neighbors as well as the intensity gradient. We choose to run ICM on the u process. This requires a black and white updating scheme since ICM is guaranteed to converge only when the updating is asynchronous

In order to run ICM on the GMAG process, we have to solve for u when interactions are present The criterion for a minimum for the vertical GMAG process is:

$$\frac{\partial\mathcal{H}_{G^2NC}}{\partial u_v(i,j)} = 0 \tag{17}$$

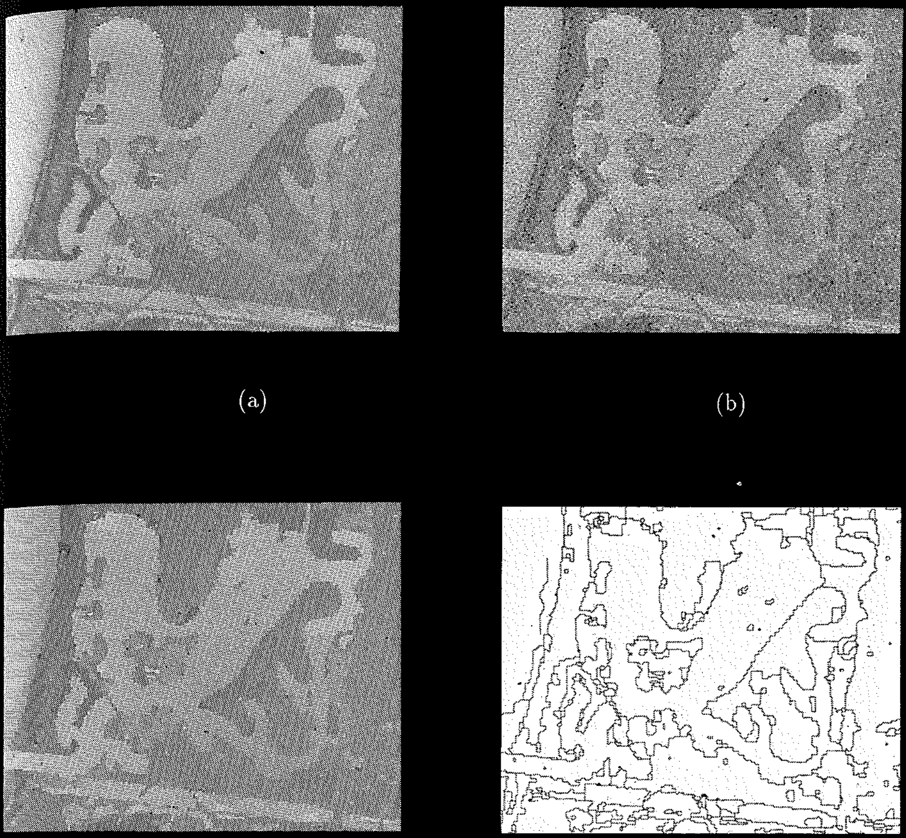


Figure 1: (a) Original image of Mission Bay (San Diego), (b) noisy image, (c) restored image using the G^{2}NC algorithm, and (d) line process image using the G^{2}NC algorithm

From this we get

$$u_v(i,j) = \frac{\mid f_x(i,j) \mid}{\sqrt{1 - \frac{2\left(\epsilon_1 \overline{(u_v^v(i,j)-q)} - \epsilon_2 \overline{(u_v^h(i,j)-q)}\right)}{r}}} \tag{18}$$

and $\overline{(u_v^v(i,j) - q)} = \frac{(u_v(i,j+1)-q)+(u_v(i,j-1)-q)}{2}$ and $\overline{(u_v^h(i,j) - q)} = \frac{(u_v(i+1,j)-q)+(u_v(i-1,j)-q)}{2}$. The formula for the horizontal GMAG process can be similarly derived. This solution for u is not guaranteed to be within $[q,r]$. If the solution lies outside, the end point energies are again compared and the solution with the lower energy is chosen. The solution depends heavily on ϵ_1 and ϵ_2. We have used the generic symbol ϵ for both ϵ_1 and ϵ_2 when they assume the same value.

We have chosen to run the Conjugate Gradient (CG) algorithm with an optimal step on the intensities. This is because the CG algorithm converges much faster than steepest descent (SD). The course we pursue is to apply ICM on the GMAG processes until they converge. We alternate between ICM on the GMAG processes and CG on the intensities. We have noticed that ICM takes very few (one to five) iterations to converge (for our case studies). The algorithm is as follows.

1. Set $c = c_*$ (usually $c_* = 0.25$).
2. Run the CG algorithm on the intensities.
3. Update GMAG processes using ICM until convergence.
4. Return to Step 2 till convergence.
5. Increase c, usually $c = 2^k c_*, k = 0, 1, \ldots$
6. Return to Step 2 until convergence of the GMAG processes.

More details on our approach can be found in [22].

We show results for our energy function and compare it to the GNC algorithm. Figures 1(a) and 1(b) contain an aerial view of Mission Bay, San Diego and the corresponding noisy image. Figures 1(c) and 1(d) show the results of applying our algorithm with the interaction terms. The parameter α was set to 676, the parameter λ to 8 and ϵ to 0.5.

The technique we have used to incorporate interaction terms is a general one and not restricted to the GNC algorithm.

3.2 TEXTURE SEGMENTATION

3.2.1 IMAGE MODEL

We use a fourth order Gauss-Markov Random Field (GMRF) to model the conditional probability density of the image intensity array given its texture labels. The texture labels are assumed to obey a first or second order discrete Markov model with a single parameter β, which measures the amount of clustering between adjacent pixels.

Let Ω denote the set of grid points in the $M \times M$ lattice, i.e., $\Omega = \{(i,j)\ ,\ 1 \leq i,j \leq M\}$ Following Geman and Graffigne [20] we construct a composite model which accounts for texture labels and gray levels. Let $\{L_s\ ,\ s \in \Omega\}$ and $\{Y_s\ ,\ s \in \Omega\ \}$ denote the labels and gray level arrays respectively. Let N_s denote the symmetric fourth order neighborhood of a site s Then assuming that all the neighbors of s also have the same label as that of s, we can write the following expression for the conditional density of the intensity at the pixel site s:

$$\mathcal{P}(Y_s = y_s \mid Y_r = y_r, r \in N_s, L_s = l) = \frac{e^{-U(Y_s=y_s \mid Y_r=y_r, r \in N_s, L_s=l)}}{Z(l|y_r, r \in N_s)} \tag{19}$$

where $Z(l|y_r, r \in N_s)$ is the partition function of the conditional Gibbs distribution and

$$U(Y_s = y_s \mid Y_r = y_r, r \in N_s, L_s = l) = \frac{1}{2\sigma_l^2}(y_s^2 - 2\sum_{r \in N_s} \Theta^l_{s,r} y_s y_r) \tag{20}$$

In (20), σ_l and Θ^l are the GMRF model parameters of the l-th texture class. The model parameters satisfy $\Theta^l_{r,s} = \Theta^l_{r-s} = \Theta^l_{s-r} = \Theta^l_r$

We view the image intensity array as composed of a set of overlapping $k \times k$ windows W_s, centered at each pixel $s \in \Omega$. In each of these windows we assume that the texture label L_s is homogeneous (all the pixels in the window belong to the same texture) and compute the joint distribution of the intensity in the window conditioned on L_s The corresponding Gibbs energy is used in the relaxation process for segmentation Let $\mathbf{Y}^*_s$ denote the 2-D vector representing the intensity array in the window W_s. Using the Gibbs formulation and assuming a free boundary model, the joint probability density in the window W_s can be written as

$$\mathcal{P}(\mathbf{Y}^*_s = \mathbf{y}^*_s | L_s = l) = \frac{e^{-U_1(\mathbf{y}^*_s | L_s = l)}}{Z_1(l)}$$

where $Z_1(l)$ is the partition function and

$$U_1(\mathbf{Y}^*_s | L_s = l) = \frac{1}{2\,\sigma_l^2} \sum_{r \in W_s} \left\{ y_r^2 - \sum_{\tau \in N^* | r+\tau \in W_s} \Theta^l_\tau y_r (y_{r+\tau} + y_{r-\tau}) \right\} \tag{21}$$

N^* is the set of shift vectors corresponding to a fourth order neighborhood system:

$$\begin{aligned} N^* &= \{\tau_1, \tau_2, \tau_3, \quad , \tau_{10}\} \\ &= \{(0,1),(1,0),(1,1),(-1,1),(0,2),(2,0),(1,2),(2,1),(-1,2),(-2,1)\} \end{aligned}$$

The label field is modeled as a first or second order discrete MRF. If $\hat{N}_s$ denotes the appropriate neighborhood for the label field, then we can write the distribution function for the texture label at site s conditioned on the labels of the neighboring sites as:

$$\mathcal{P}(L_s | L_r\ ,\ r \in \hat{N}_s) = \frac{e^{-U_2(L_s \mid L_r)}}{Z_2}$$

where Z_2 is a normalizing constant and

$$U_2(L_s \mid L_r,\ r \in \hat{N}_s) = -\beta \sum_{r \in \hat{N}_s} \delta(L_s - L_r)\ ,\ \beta > 0 \tag{22}$$

In (22), β determines the degree of clustering, and $\delta(i-j)$ is the Kronecker delta. Using the Bayes rule, we can write

$$\mathcal{P}(L_s \mid \mathbf{Y}_s^*,\ L_r,\ r \in \hat{N}_s) = \frac{\mathcal{P}(\mathbf{Y}_s^* \mid L_s)\,\mathcal{P}(L_s \mid L_r,\ r \in \hat{N}_s)}{\mathcal{P}(\mathbf{Y}_s^*)} \tag{23}$$

Since $\mathbf{Y}_s^*$ is known, the denominator in (23) is just a constant. The numerator is a product of two exponential functions and can be expressed as,

$$\mathcal{P}(L_s \mid \mathbf{Y}_s^*,\ L_r,\ r \in \hat{N}_s) = \frac{1}{Z_p}\, e^{-U_p(L_s \mid \mathbf{Y}_s^*,\ L_r,\ r\in\hat{N}_s)} \tag{24}$$

where Z_p is the partition function and $U_p(\ )$ is the posterior energy corresponding to (23). From (21) and (22) we write

$$U_p(L_s \mid \mathbf{Y}_s^*,\ L_r,\ r \in \hat{N}_s) = w(L_s) + U_1(\mathbf{Y}_s^* \mid L_s) + U_2(L_s \mid L_r,\ r \in \hat{N}_s) \tag{25}$$

Note that the second term in (25) relates the observed pixel intensities to the texture labels and the last term specifies the label distribution. The bias term $w(L_s) = \log Z_1(L_s)$ is dependent on the texture class and it can be explicitly evaluated for the GMRF model considered here using the toroidal assumption (the computations become very cumbersome if toroidal assumptions are not made). An alternate approach is to estimate the bias from the histogram of the data as suggested by Geman and Graffigne [20]. Finally, the posterior distribution of the texture labels for the entire image given the intensity array is

$$\mathcal{P}(\mathbf{L} \mid \mathbf{Y}^*) = \frac{\mathcal{P}(\mathbf{Y}^* \mid \mathbf{L})\,\mathcal{P}(\mathbf{L})}{\mathcal{P}(\mathbf{Y}^*)} \tag{26}$$

Maximizing (26) gives the optimal Bayesian estimate. We note that a stochastic relaxation algorithm requires only the computation of (24) to obtain the optimal solution. The deterministic relaxation algorithm given in the next section also uses these values and performs descent on a related energy function.

3.2.2 A NEURAL NETWORK FOR TEXTURE CLASSIFICATION

We describe the network architecture used for segmentation and the implementation of deterministic relaxation algorithms. Let $U_1(i,j,l) = U_1(\mathbf{Y}_s^*, L_s = l) + w(l)$ where $s = (i,j)$ denotes a pixel site and $U_1(\ )$ and $w(l)$ are as defined in (25). The network consists of K layers, each layer arranged as an $M \times M$ array, where K is the number of texture classes in the image and M is the dimension of the image. The elements (neurons) in the network are assumed to be binary and are indexed by (i,j,l) where $(i,j) = s$ refers to their position in the image and l refers to the layer. The (i,j,l)-th neuron is said to be ON if its output V_{ijl} is 1, indicating that the corresponding site $s = (i,j)$ in the image has the texture label l.

A solution for the MAP estimate can be obtained by minimizing (26). We approximate the posterior energy by

$$U(\mathbf{L}|\mathbf{Y}^*) = \sum_s \{U(\mathbf{Y}_s^*|L_s) + w_{L_s} + U_2(L_s)\} \tag{27}$$

and the corresponding Gibbs energy to be minimized can be written as

$$E = \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M}\sum_{l=1}^{K} U_1(i,j,l)V_{ijl} - \frac{\beta}{2}\sum_{l=1}^{K}\sum_{i=1}^{M}\sum_{j=1}^{M}\sum_{(i',j')\in\hat{N}_{ij}} V_{i'j'l}V_{ijl} \tag{28}$$

where $\hat{N}_{ij}$ is the neighborhood of site (i,j) In (28), it is implicitly assumed that each pixel site has a unique label, i.e. only one neuron is active in each column of the network For the deterministic relaxation algorithm, a simple method is to use a WTA circuit for each column so that the neuron receiving the maximum input is turned on and the others are turned off

The network model is a version of the ICM algorithm of Besag [8]. We observe that in general any algorithm based on MRF models can be easily mapped on to neural networks with local interconnections.

3.2.3 STOCHASTIC ALGORITHMS FOR TEXTURE SEGMENTATION

The MAP rule searches for the configuration L that maximizes the posterior probability distribution. This is equivalent to maximizing $\mathcal{P}(\mathbf{Y}^* \mid \mathbf{L})\ \mathcal{P}(\mathbf{L})$ as $\mathcal{P}(\mathbf{Y}^*)$ is independent of the labels and $\mathbf{Y}^*$ is known The right hand side of (26) is a Gibbs distribution. To maximize (26) we use simulated annealing [3]. It samples from the conditional distribution

$$\frac{e^{-\frac{1}{T_k}U_p(L_s \mid \mathbf{Y}_s^*,\ L_r, r\in\hat{N}_s)}}{Z_{T_k}}$$

in order to maximize

$$\frac{e^{-U_p(\mathbf{L} \mid \mathbf{Y}^*)}}{Z}$$

T_k being the time varying parameter, referred to as the temperature The process is guaranteed to converge to a uniform distribution over the label configuration that corresponds to the MAP solution

The choice of the objective function for optimal segmentation can significantly affect its result. In many implementations, the most reasonable objective function is the one that minimizes the expected percentage misclassification per pixel The solution to the above objective function can be obtained by maximizing the marginal posterior distribution (MPM) of L_s given the observation $\mathbf{Y}^*$, for each pixel s

$$\mathcal{P}\{L_s = l_s \mid \mathbf{Y}^* = \mathbf{y}^*\} \propto \sum_{\mathbf{l}\mid L_s = l_s} \mathcal{P}(\mathbf{Y}^* = \mathbf{y}^* \mid \mathbf{L} = \mathbf{l})\ \mathcal{P}(\mathbf{L} = \mathbf{l})$$

The summation above extends over all possible label configurations keeping the label at site s constant. To find the optimal solution we use the stochastic algorithm suggested in [23] The algorithm samples out of the posterior distribution of the texture labels given the intensity at $T = 1$ The Markov chain associated with the sampling algorithm converges with probability one to the posterior distribution

3.2.4 STOCHASTIC LEARNING AND NEURAL NETWORKS

The texture classification discussed in the previous sections can be treated as a relaxation labelling problem and stochastic automata can be used to learn the texture labels. A stochastic automaton is a decision maker operating in a random environment [24] and is assigned to each of the pixel sites in the image. The actions of the automata correspond to selecting a label for the pixel site to which it is assigned. Thus for a K class problem each automaton has K actions and a probability distribution over this action set. Initially the labels are assigned randomly with equal probability. Since the number of automata involved is very large, it is not practical to update the action probability vector at each iteration. Instead we combine the iterations of the neural network described in the previous section with the stochastic learning algorithm. After each convergence of the deterministic relaxation algorithm, the action probabilities $(p_{s,l})$ are updated as follows

$$\begin{aligned} p_{s,l_s}(t+1) &= p_{s,l_s}(t) + a\ \lambda(t)\ [1 - p_{s,l_s}(t)] \\ p_{s,l}(t+1) &= p_{s,l}(t)[1 - a\ \lambda(t)],\ \forall l \neq l_s \text{ and } \forall\ s \end{aligned} \quad (29)$$

where l_s denotes the label of site s at equilibrium and $p_{s,l}(t)$ is the probability of choosing label l for site s at time t. The response $\lambda(t)$ is derived as follows: Suppose the present label configuration resulted in a lower energy state compared to the previous one, then it results in a $\lambda(t) = \lambda_1$ and if the energy increases we have $\lambda(t) = \lambda_2$ with $\lambda_1 > \lambda_2$. In our simulations we used $\lambda_1 = 1$ and $\lambda_2 = 0.25$. A new configuration for the relaxation network is then generated from the updated action probabilities.

Thus the system consists of a relaxation network and a learning network. The relaxation network is similar to the one in Section 3.2.2, except that the initial state is decided by the learning network. This results in an iterative hill climbing type algorithm which combines the fast convergence of deterministic relaxation with the sustained exploration of the stochastic algorithm. The stochastic part prevents the algorithm from getting trapped in local minima and at the same time "learns" from the search by updating the state probabilities. Each cycle now has two phases: the first consists of the deterministic relaxation network converging to a solution; the second consists of the learning network updating its state. For further details, the reader is referred to [25].

3.2.5 EXPERIMENTAL RESULTS

The segmentation results using the above algorithms are given on one example. The least-square estimates (LSE) of the parameters σ_l and Θ_l corresponding to the fourth order GMRF for each texture class were pre-computed from 64×64 images of the textures.

Figure 2(a) shows a 256×256 image having six textures: leather, grass, wool, wood, pig skin and sand. The maximum likelihood solution is shown in Figure 2(b) and Figure 2(c) is the solution obtained by the deterministic relaxation network with the result in Figure 2(b) as the initial condition. The MAP solution using simulated annealing is shown in Figure 2(d). Figure 2(e) shows the MPM result. As indicated in Table 1, simulated annealing has the lowest percentage error in classification. Introducing learning into deterministic relaxation considerably improves the performance (Figure 2(f)).

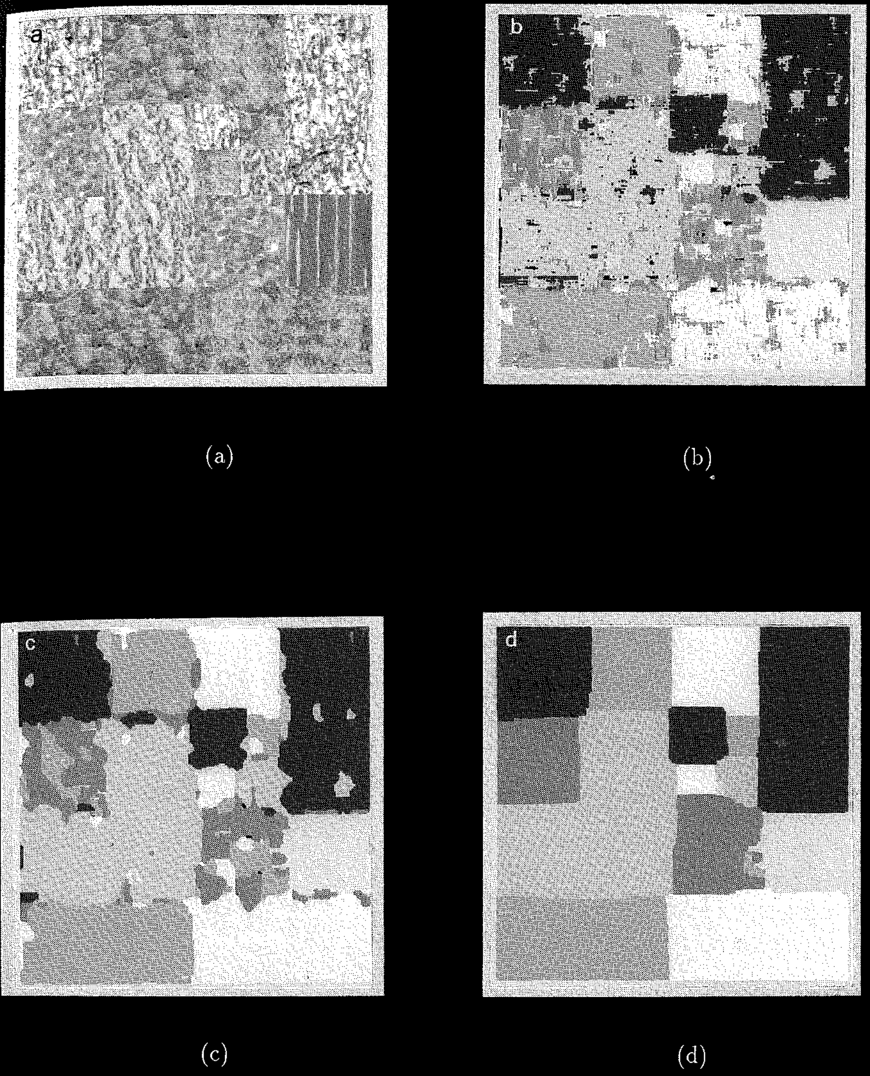
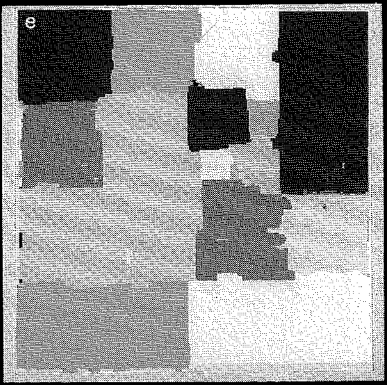
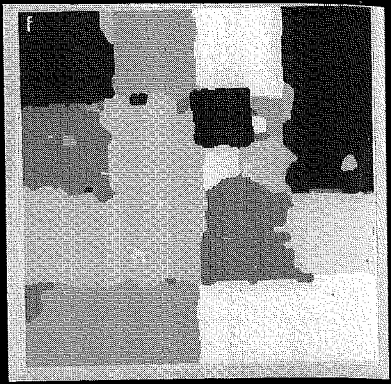


Figure 2: (a) Original image consisting of six textures, (b) maximum likelihood solution, (c) deterministic relaxation with (b) as initial condition, (d) MAP estimate using simulated annealing





(e) (f)

Figure 2: (contd.) (e) MPM solution and (f) network with stochastic learning

Table 1
Percentage misclassification for the six class problem

Algorithm	Percentage Error
Maximum Likelihood Estimate	22 17
Neural network (MLE as initial state)	16 25
Simulated annealing (MAP)	6 72
MPM algorithm	7 05
Neural network with learning	8 7

4 CONCLUSIONS

We have presented two applications which highlight the close links between MRFs and NNs There are several issues which have not been addressed and are beyond the scope of this chapter. We briefly mention a few of these issues The use of MRFs in integrating visual modules [26] has been an important area of investigation in recent years. Density function estimation [27] is an important area where NNs and MRFs can be related Finally, the important problem of binding syntactical structures is addressed by the Dynamic Link Architecture [28] which uses a fast synaptic plasticity term in addition to the more conventional slow weight modification term This is an alternative to using either higher-order connections or additional hidden units to model hidden structural relationships in scenes

References

[1] D Marr, *"Vision a computational investigation into the human representation and processing of visual information"*, W. H Freeman and Co , San Francisco, CA, 1982.

[2] J J. Hopfield, "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, 1984.

[3] S Geman and D Geman, "Stochastic relaxation, Gibbs Distributions and the Bayesian restoration of Images", *IEEE Trans on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, pp. 721–741, November 1984.

[4] D E Rumelhart, J L. McClelland, and the PDP Research Group, editors, *"Parallel Distributed Processing · Explorations in the Microstructure of Cognition Volume 1· Foundations"*, MIT Press, Cambridge, MA, 1986

[5] T Maxwell, C. L Giles, and Y. C Lee, "Nonlinear Dynamics of Artificial Neural Systems", In *Proceedings of the Neural Networks for Computing Conference*, Snowbird, Utah, April 1986

[6] 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, pp 54–115, 1987

[7] S Lloyd and H R. Pagels, "Complexity as Thermodynamic Depth", *Annals of Physics*, vol 188, pp 186–213, 1988

[8] J. Besag, "On the statistical analysis of dirty pictures", *Journal of the Royal Statistical Society B*, vol. 48(3), pp 259–302, 1986.

[9] D. Geiger and F Girosi, "Parallel and deterministic algorithms for MRFs: surface reconstruction and integration", A I Memo, No 1114, Artificial Intelligence Lab, M.I T, June 1989

[10] Y G Leclerc, "Constructing Simple Stable Descriptions for Image Partitioning", *International Journal of Computer Vision*, vol 3, pp 73–102, 1989.

[11] J Besag, "Efficiency of Pseudo-likelihood Estimation for Simple Gaussian Fields", *Biometrika*, 64, pp 616–618, 1977

[12] A P Dempster, N M Laird, and D B Rubin, "Maximum Likelihood from incomplete data via the EM algorithm", *Journal of the Royal Statistical Society B*, 39, pp 1–38, 1977

[13] D H. Ackley, G E Hinton, and T J Sejnowski, "A learning algorithm for Boltzmann machines", *Cognitive Science*, 9, pp 147–169, 1985

[14] S Kullback, *"Information Theory and Statistics"*, Dover Publications, New York, NY, 1968.

[15] C Peterson and J. R. Anderson, "A mean field theory learning algorithm for neural networks", *Complex Systems*, 1, pp 995–1019, 1987.

[16] K Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position", *Biological Cybernetics*, 36(4), pp. 193–202, 1980

[17] L R. Rabiner and B.-H. Juang, "An Introduction to Hidden Markov Models", *IEEE ASSP Magazine*, vol. 3(1), pp 4–16, January 1986.

[18] H Haken, *"Information and Self-Organization: a macroscopic approach to complex systems"*, volume 40 of *Springer Series in Synergetics*, Springer-Verlag, 1989

[19] D Geiger and A L Yuille, "A Common Framework for Image Segmentation", In *Proceedings of the International Conf. on Pattern Recognition, ICPR-90*, Atlantic City, NJ, June 1990

[20] S Geman and C Graffigne, "Markov random fields image models and their application to computer vision", In *Proc. of the Int. Congress of Mathematicians 1986*, Ed A.M. Gleason, American Mathematical Society, Providence, 1987

[21] A Blake and A. Zisserman, *"Visual Reconstruction"*, MIT Press, Cambridge, MA, 1987

[22] A. Rangarajan and R Chellappa, "Generalized Graduated Non-Convexity Algorithm for Maximum A Posteriori image estimation", In *Proceedings of the International Conf on Pattern Recognition, ICPR-90*, Atlantic City, NJ, June 1990

[23] J. L Marroquin, S Mitter, and T. Poggio, "Probabilistic solution of ill-posed problems in computational vision", *J. Am Stat Assoc.*, 82, pp. 76–89, 1987.

[24] K S Narendra and M A. L Thathachar, *"Learning Automata : an introduction"*, Prentice-Hall, Englewood Cliffs, NJ, 1989.

[25] B S Manjunath, T Simchony, and R Chellappa, "Stochastic and Deterministic Networks for Texture Segmentation", *IEEE Trans on Acoust., Speech and Sig. Proc*, vol ASSP-38(6), pp 1039–1049, June 1990

[26] T. Poggio, E. B. Gamble, and J J Little, "Parallel Integration of Vision Modules", *Science*, vol 242, pp. 436–440, October 1988

[27] T. Kohonen, *"Self-Organization and Associative Memory"*, volume 8 of *Springer series in Information Sciences*, Springer-Verlag, New York, NY, 3rd edition, 1989.

[28] C von der Malsburg and E Bienenstock, "Statistical coding and short-term synaptic plasticity: A scheme for knowledge representation in the brain", In E Bienenstock, F Fajelman Soulié, and G Weisbuch, editors, *Disordered Systems and Biological Organization*, Springer-Verlag, Berlin, 1986

Artificial Neural Networks and Statistical Pattern Recognition
Old and New Connections
I.K. Sethi and A.K Jain (Editors)
© 1991 Elsevier Science Publishers B.V All rights reserved

Connectionist Models and their Application to Automatic Speech Recognition

Yoshua Bengio and Renato De Mori

School of Computer Science, McGill University, 3480 University str., H3A2A7, Montreal, Que., Canada

Abstract

The purpose of this chapter is to study the application of some connectionist models to automatic speech recognition. Ways to take advantage of a-priori knowledge in the design of those models are first considered. Then algorithms for some recurrent networks are described since they are well-suited to handling temporal dependences such as those found in speech. Some simple methods that accelerate the convergence of gradient descent with the back-propagation algorithm are discussed. An alternative approach to speed-up the networks are systems based on Radial Basis Functions (local representation). Detailed results of several experiments with these networks on the recognition of phonemes for the TIMIT database are presented. In conclusion, a cognitively relevant model is proposed. This model combines both a local representation and and a distributed representation subnetworks to which correspond respectively a fast-learning and a slow-learning capability.

1. INTRODUCTION

Artificial neural networks are simplified models of neural computation simulated in computers. Connectionist models are models of computation based on massively parallel networks of simple computational units, inspired by the organization of the brain, simulated in order to help construct and explain connectionist theories of cognition [Rumelhart & McClelland 86]. A lot of recent research results [*e.g.* Bengio *et al.* 89a; LeCun *et al.* 89; Lippman 89; Waibel *et al.* 88; Giles *et al.* 90; Pomerleau 89] indicate that they could be useful in addressing several artificial intelligence problem areas, in particular those relating to perception, such as for example automatic speech recognition [Lippman 89].

One of the advantages of connectionist models is that they can integrate learning from examples with some a-priori knowledge. In section 2 we consider several aspects of the design of neural networks systems which can benefit from a-priori knowledge by improving generalization.

A class of neural network models of special interest for the problem of speech recognition is the class of recurrent neural networks which contain cycles enabling them to retain and use some information about their past history. This feature seems well suited to the problem of speech recognition: transform a sequence (of undetermined length) of vectors of acoustic descriptors into a corresponding sequence of symbols representing speech units (*e.g.*, phonemes, diphones, words) or alternatively, into a sequence of degrees of evidence for those symbols. In section 3 some learning algorithms for recurrent neural network are described: a general purpose training algorithm for discrete-time networks of arbitrary connectivity which buffers past activations, as well as an algorithm for a constrained type of network allowing only self-loops but for which it is not necessary to keep the past history and which has the same computation time and space complexity as the backpropagation algorithm for feedforward networks.

Some simple methods that can greatly accelerate the convergence of gradient descent with the back-propagation algorithm are discussed in section 4. In particular we introduce an original technique that provides a different learning rate to different layers of a multi-layered sigmoid network. It is based on the idea that each layer should be allowed to modify itself independently of the damping of the gradient due to layers closer to the outputs of the network. The experiments described in this section show the significant acceleration obtained with this technique.

A large section (5) of this chapter is devoted to a particular type of connectionist model based on a local representation and often called Radial Basis Functions network (RBF). Each unit in the network may be understood as representing a prototypical input pattern while the outputs are formed by interpolating between these prototypes, i.e., by combining linearly the gaussian-shaped outputs of the hidden units. This type of model has several advantages: solid mathematical foundations [Poggio & Girosi 89], neurological [Mel & Koch 90] and psychological [Kruschke 90] plausibility as well as fast implementation. Implementation can be fast for several reasons: the networks can be initialized with prototypes rather than with random values, and, because of the local nature of their responses, the small subset of hidden units that respond significantly can be found rapidly with a fast search. We propose in section 5.5 a simple algorithm to implement that search. The results of several comparative phoneme recognition experiments performed on the TIMIT database of continuous speech are described in section 5. In particular the advantage of using information about the classes in the preprocessing is seen (to find the prototypes), the effect of architecture and the representation of context on generalization are explored with several experiments, concluding with an experiment with a recurrent network which combines RBF units with sigmoid units.

Sigmoid networks - which imply a distributed internal representation - and RBF networks - which are based on a local representation - both have advantages that seem to be complementary. In the last section (6), we propose a new idea on ways to construct networks that combine those two types of representations. A constructive algorithm for a network made of a local and a distributed representation subsystem is described, in order to implement respectively fast and slow learning. This algorithm is characterized by a reorganization phase in which the distributed subsystem tries to incorporate what is represented in the local subsystem.

2. USE OF A-PRIORI KNOWLEDGE

We are seeking in our research effort the advantages of combining the flexibility and learning abilities of neural networks with as much knowledge from speech science as possible in order to advance the construction of a speaker independent speech recognition system. A-priori knowledge can be used in many steps of this construction, *e.g.*, preprocessing, input coding, input representation, output coding, output supervision, and architectural design.

2.1 Preprocessing

Previous work had indicated that the choice of preprocessing significantly influences the performance of a neural network recognizer [*e.g.*, Bengio & De Mori 89]. Different types of preprocessing processes and acoustic features can be utilized at the input of a neural network. Experiments with various acoustic features were performed, using filters derived from the Fast Fourier Transform (FFT), cepstrum transforms, energy levels (of both the signal and its derivative), counts of zero crossings and energy ratios [Bengio, Gori & De Mori 89], as well as an ear model and synchrony detector [Cosi, Bengio & De Mori 90].

2.1.1 Cepstral *vs* Spectral parameters

Previous experiments had shown that spectral representations produced better results than cepstrum coefficients with neural networks of sigmoid units [Bengio *et al*, NIPS 89]. It seems to be also the case for RBF networks, as was shown in an experiment where the task was the same as defined in section 5.4, *i.e.*, the recognition of 39 TIMIT phonemes. The networks had 25 input units with 4 delays (0, 1, 2 and 3) between the inputs and the hidden units and 78 hidden units (2 clusters/ class) initialized with k-means and with 3 delays (0, 2 and 4) between the hidden and the output units for the 39 classes. The output weights were obtained with the pseudo-inverse method [Penrose 55; Broomhead & Lowe 88]. The frame by frame error on the test set was 54.5% for the cepstral input and 46.5% for the spectral input. In both cases, we used a mel scale with 24 filters and provided the frame energy to the system, *i.e.*, the network had 25 inputs. With a slightly different architecture, with additional direct connections from inputs to outputs, the errors were reduced but the spectrum still produced better generalization: the error on the test set with cepstral input was 48.1% and the error with the spectral input was 45.6%.

2.1.2 Ear model vs. FFT

In recent years, basilar membrane, inner cell and nerve fiber behavior have been extensively studied by auditory physiologists and neurophysiologists and knowledge about the human auditory pathway has become more accurate. A number of studies have been accomplished and a considerable amount of data has been gathered in order to characterize the responses of nerve fibers in the eight nerve of the mammalian auditory system using tone, tone complexes and synthetic speech stimuli [Seneff 85; Delgutte & Kiang 84].

Experiments on the speaker-independent recognition of 10 English vowels were performed on isolated words, comparing the use of an ear model with an FFT as preprocessing [De Mori, Bengio & Cosi 89; Cosi, Bengio & De Mori 90]. The FFT was computed using a mel scale and the same number of filters (40) as for the ear model. The ear model was derived from the one proposed by Seneff (1985). Performance was significantly better with the ear model than with the FFT: the error on the test set with the ear model was 4.6% while with the FFT, the error was 13.0% [Bengio *et al.* 89b; De Mori, Bengio & Cosi 89]. However, the ear model required about two orders of magnitude more processing time.

2.2 Output coding

If each output of the network is interpreted as representing a phonetic property, then an output value can be seen as a degree of evidence with which that property has been observed in the data. We have explored the use of an output coding scheme based on phonetic features defined by the way speech is produced. This is generally more difficult to learn but results in better generalization, especially with respect to new sounds that had not been seen by the network during the training. This was demonstrated in experiments on vowel recognition in short isolated words in which the networks were trained to recognized the place and the manner of articulation [De Mori, Bengio & Cosi 89; Cosi, Bengio & De Mori 90]. In addition, the resulting representation is more compact than when using one output for each phoneme. However, it was much more difficult to apply such a coding based on articulatory features to the recognition of continuous speech. Experiments on the TIMIT [Garofolo 88] database showed that the target (ideal) values for place of articulation of phonemes in continuous speech do not correspond well to the acoustic signal. Coarticulation effects and the short duration of phonemes in continuous speech often prevent the place of articulation to reach its (stable) target value. Instead, the place of articulation is continuously changing and is strongly influenced by the context.

2.3 Architectural design

Hypothesis about the nature of the processing to be performed by the network based on a-priori knowledge of speech production and recognition processes enables to put constraints on the architecture. These constraints result in a network that generalizes better than a fully connected network [Baum & Haussler 89]. Here are several examples of application of modularization: the same architectural constraints do not have to apply to all of the subtasks. One can modularize according to acoustic context (different networks are triggered when various acoustic contexts are detected)[Bengio, Cardin, De Mori, Merlo 89]. Another solution is modularization by independent articulatory features (vertical and horizontal place of articulation) [in De Mori, Bengio & Cosi 89]. Another type of modularization, by subsets of phonemes, was explored in [Waibel 89].

A striking example of the influence of architecture on recognition performance is demonstrated in the case of nasals recognition in [Bengio *et al.* 90]. In this experiment the architecture of the network was modified in order to consider a speech theory stating that very significant discriminatory information is to be found in the transition between the vowel and the nasal. This resulted in a drastic improvement in generalization (from 15% to 2.6% error).

3. RECURRENT BACK-PROPAGATION NETWORKS

Feedforward networks have less expressive power than recurrent networks. However recurrent networks are more complicated, because of their learning algorithms which must consider the influence of past events on future errors, and because of their dynamics, which can be unstable or even chaotic.

3.1 Time Unfolding

The idea of time unfolding was proposed in [Rumelhart, Hinton & Williams 86]. It consists in considering back-propagation of gradients to previous times by unfolding the network into an equivalent network with a different layer for each time slice.

Let us consider the general case of a network with multiple links between pairs of units (associated to multiple discrete delays) and units that can compute any differentiable parametric function of their inputs. In particular we will consider the following functions:

- asymmetric sigmoid of the weighted sum of the inputs,
- symmetric sigmoid of the weighted sum of the inputs,
- linear weighted sum of the inputs,
- product of the inputs,
- elliptic gaussian function (radial basis function).

The operation of the network can be divided into a forward phase - to compute the outputs of each unit and the outputs of the network - and a backpropagation phase, to compute the gradient of a cost function (explicitly expressed in terms of the outputs of the output units) w.r.t. all the parameters of the system. The forward phase operation is defined as follows:

$$X_i(t) = h_i(\mathbf{Y}(t), \mathbf{Y}(t-1), \mathbf{Y}(t-2), \ldots) \quad (1)$$
$$Y_i(t) = f_i(X_i(t)) \quad (2)$$

where $\mathbf{Y}(t)$ is the vector of outputs of all the units in the network at time t, *i.e.*, $\mathbf{Y}(t)=[Y_1(t), Y_2(t), \ldots Y_n(t)]'$. $X_i(t)$ is an intermediate variable sometimes called activation and which might be considered to be related to the activation potential of a neuron while $Y_i(t)$ corresponds to its firing rate. We will consider the following cases for $h()$ and $f()$, however, any continuous differentiable function is acceptable:

Table 1: Various neuron transfer functions.

	$h() = X_i(t)$	$Y_i(t) = f(X_i(t))$
asymmetric sigmoid	$\sum_j W_{ij} Y_{s_{ij}}(t-d_{ij})$	$Y_i(t) = 1/(1+e^{-X_i(t)})$
symmetric sigmoid	$\sum_j W_{ij} Y_{s_{ij}}(t-d_{ij})$	$\tanh(X_i(t))$
linear	$\sum_j W_{ij} Y_{s_{ij}}(t-d_{ij})$	$X_i(t)$
product	$\pi_j Y_{s_{ij}}(t-d_{ij})$	$X_i(t)$
gaussian	$\sum_j (W_{ij} - Y_{s_{ij}}(t-d_{ij}))^2 \beta_{ij}$	$e^{-X_i(t)}$

Here s_{ij} is the node number of the source unit for the j^{th} input link of unit i and d_{ij} is the discrete delay associated to that link. For gaussian units ß is the inverse of the variance of the gaussian, assuming a diagonal covariance matrix. We could also consider ß fixed for all dimensions (spherical gaussian) or a full covariance matrix.

For the backward phase, the basic idea of this algorithm is to compute $\partial E/\partial W_{ij}$. In order to do so one computes recursively in a backpropagation phase $\partial E/\partial X_i(t)$ for all units i, starting from the last time frame L and going backward in time until the first frame of the input sequence, again starting for each time t from the last unit Nu down to the first unit (note that this order does not matter if all links have a positive delay).

First we compute $\partial E/\partial X_i(t)$ recursively using back-propagation:

$$\partial E/\partial X_i(t) = \sum_{k:s_{kj}=i} \partial E/\partial Xk(t+d_{kj}) \;\; \partial Xk(t+d_{kj})/\partial Y_i(t) \;\; \partial Y_i(t)/\partial X_i(t) + \\ \delta_{(\text{i is an output unit})} \;\; (Y_i(t)-Y_i^*(t)) \;\; \partial Y_i(t)/\partial X_i(t)) \tag{3}$$

where the term on the second line of (3) is only for output units at times when there is supervision. It is shown here for the case of minimization of the Least Mean Squares criterion:

$$E = 0.5 \sum_t \sum_i (Y_i(t)-Y_i^*(t))^2 \tag{4}$$

where $Y_i^*(t)$ is the target output for unit i at time t. $\partial Y_i(t)/\partial X_i(t)$ is different for different types of units, it depends on the choice of $f_i()$ while $\partial Xk(t+d_{kj})/\partial Y_i(t)$ depends on the choice of $g_i()$:

Table 2: Derivatives for the various types of neurons.

	$\partial Y_i(t)/\partial X_i(t)$	$\partial Xk(t+d_{kj})/\partial Y_i(t)$
asymmetric sigmoid	$Y_i(t)\ (1 - Y_i(t))$	W_{kj}
symmetric sigmoid	$0.5\ (1+Y_i(t))(1-Y_i(t))$	W_{kj}
linear	1	W_{kj}
product	1	$Xk(t+d_{kj})/Y_i(t)$
gaussian	$- Y_i(t)$	$-2\ (W_{kj}- Y_i(t))\ ß_{kj}$

The gradient of the criterion w.r.t. the parameters can then be computed as follows:

$$\partial E/\partial W_{ij} = \sum_t \partial E/\partial Xi(t) \;\; \partial Xi(t)/\partial W_{ij} \tag{5}$$

and for gaussian units one also needs to compute

$$\partial E/\partial ß_{ij} = \sum_t \partial E/\partial Xi(t) \;\; \partial Xi(t)/\partial ß_{ij} \tag{6}$$

where $\partial Xi(t)/\partial W_{ij}$ and $\partial Xi(t)/\partial ß_{ij}$ depend on the choice of $g_i()$:

• asymmetric sigmoid, symmetric sigmoid, linear:
$\partial Xi(t)/\partial W_{ij} = Ys_{ij}(t-d_{ij})$ (7)

• gaussian:
$\partial Xi(t)/\partial W_{ij} = 2\ (W_{ij} - Ys_{ij}(t-d_{ij}))\ \beta_{ij} = -\ \partial Xi(t)/\partial Ys_{ij}(t-d_{ij})$ (8)
$\partial Xi(t)/\partial \beta_{ij} = (W_{ij} - Ys_{ij}(t-d_{ij}))^2$ (9)

This algorithm has a time complexity of O(L . Nw) for both the forward and the backward phase and needs space O(L . Nu) in order to store the activation of all units for the whole sequence. L is the length of a sequence, Nw is the number of weights and Nu is the number of units. Back Propagation (BP) for feedforward networks needs O(Nu) space and O(Nw) time per frame or O(L . Nw) time per sequence of length L. Hence this algorithm has the same time complexity (per epoch) as ordinary BP for feedforward networks but needs L times more space.

3.2 BPS

BPS (originally, Back-Propagation for Sequences) is a learning algorithm for a certain type of constrained recurrent networks that was proposed in [Bengio, Gori & De Mori 89]. It applies to networks which have the following architectural characteristics:

- the network has a multi-layer architecture
- static and dynamic neurons are distinguished. The former are as in static BP whereas the latter have a local feedback connection, thus their activation evolves as follows:

$$Xi(t+1) = W_{ii}\ Xi(t) + \sum_j W_{ij}\ f(Xj(t+1)) \tag{10}$$

- the input connections from dynamic neurons only come from the input layer

The learning algorithm is based on the forward recurrent computation of $\partial Xi(t)/\partial W_{ij}$:

$$\begin{aligned} \partial Xi(t+1)/\partial W_{ij} &= f(Xj(t+1)) + W_{ii}\ \partial Xi(t)/\partial W_{ij} \qquad \text{for } i\neq j,\\ \partial Xi(t+1)/\partial W_{ij} &= Xi(t) + W_{ii}\ \partial Xi(t)/\partial W_{ii} \qquad \text{for } i=j \end{aligned} \tag{11}$$

The rest of the algorithm follows exactly the same lines as for the algorithm for strictly feedforward networks:

$$\partial E/\partial W_{ij} = \sum_t \partial E(t)/\partial W_{ij} = \sum_t \partial E(t)/\partial Xi(t)\ \ \partial Xi(t)/\partial W_{ij} \tag{12}$$

where $\partial E(t)/\partial Xi$ is computed with the backpropagation recursion as follows, assuming a Least Mean Squares criterion:

For output units i: $\partial E(t)/\partial Xi = (Yi(t) - Yi(t)^*)\ \partial Y_i(t)/\partial X_i(t)$ (13)
For hidden units j: $\partial E(t)/\partial Xj = \sum_k \partial E(t)/\partial Xk(t)\ \ \partial Xk(t)/\partial Yj(t)\ \ \partial Yj(t)/\partial Xj(t)$ (14)

Depending on the type of unit, the partial derivatives $\partial Y_i(t)/\partial X_i(t)$, $\partial Xk(t)/\partial Yj(t)$ and $\partial Xi(t)/\partial Wij$ are computed for example as shown in table 2 and equation7.

The cost of running the learning algorithm is the same as for a feedforward back-propagation network. Furthermore this algorithm has the advantage that it is local in time: one does not need to keep a buffer of past events in order to take into account influences from the past in the calculation of the gradient. This algorithm can be seen as a compromise between unfolding in time (using only backward recursion) and full feedforward (using only forward recursion) [Williams & Zipser 88; Kuhn 87]. However it has the drawback of being limited to a certain type of architectures. On the other hand this constraint could be appropriate to some problems of speech recognition [Bengio, Gori & De Mori 89] and in that case help to improve generalization.

By expressing the Wii as a bounded function of another parameter Di, one can force stability of the network and put arbitrary bounds upon the magnitude of Wii:

$$W_{ii}(Di) = B\,(1 - \exp(-Di))/(1 + \exp(-Di)) \qquad (15)$$

with

$$\partial E/\partial Di = \partial E/\partial Wii \;\; \partial W_{ij}/\partial Di \qquad (16)$$

4. FAST IMPLEMENTATION OF BP

One of the main criticisms of Back-Propagation (BP) algorithms with gradient descent has been the slowness of their convergence. However, several simple techniques can significantly accelerate the training. The most common observation concerns the use of stochastic weight updates rather than batch weight update. With stochastic weights update one adapts the weights after each pattern, whereas with batch update one accumulates the gradients from all the patterns and only then the weights are updated. Stochastic update was found to be significantly faster than batch update [see Bottou *et al* 90], especially in pattern recognition problems (*e.g.*, speech recognition, written digits recognition). Yann Le Cun (1989) suggested as an explanation that for such problems one might explain this improvement by the redundancy present in the training set. Since the gradient gives only the slope of the error function, if many gradient contributions (from different patterns) all point in a similar direction then batch update wastes a lot of time in order to refine that direction when a gross estimate might have been sufficient (since at the next step we might have to chose a different direction anyway). Another advantage of stochastic gradient descent is that it may allow to escape from local minima or from regions of the weight space from which straight gradient descent might need a lot of iterations to escape. Randomness is introduced by the noisy evaluation of the gradient based on very little data (*e.g.*, one pattern). This noise is proportional to the size of the learning rate. Hence if one starts with a large learning rate and slowly decreases it (for example as 1/t) then one might approach the global minimum and escape local minima in a way that is perhaps similar to simulated annealing [L. Bottou 90, personal communication].

Many schemes have been proposed to accelerate convergence by 1) adapting the learning rate, and 2) using different rates for different weights. For example, Robert Jacobs [Jacobs 88] proposes such a method, called the delta-bar-delta rule, which basically increases the rate associated to a weight when the current gradient for that weight has the same sign as a decaying average of previous gradients and decreases this rate if the signs are opposite. However, this method, like many acceleration methods (*e.g.*, momentum, use of second order information) rely on an exact evaluation of the gradient, *i.e.*, on batch update. Consequently the advantages they bring are often offset by the disadvantage of using batch update, especially for large pattern recognition problems.

4.1 Layer-dependent weight change

One of the reasons for the slowness of gradient descent for multilayer networks is that the gradient tends to become exponentially smaller for layers that are far from the outputs. Let's consider the backpropagation step to compute the gradient w.r.t. to the activations at layer l given the gradient w.r.t. the activations of layer l+1 (layer 0 is the input):

$$\partial E/\partial \mathbf{X}_l = f'(\mathbf{X}_l)\ Wt_{l+1,l}\ \partial E/\partial \mathbf{X}_{l+1} \tag{17}$$

where $Wt_{l+1,l}$ is the transpose of the matrix of weights from layer l to layer l+1, E is the error to minimize, f'() is the derivative of the sigmoid-function f() (*e.g.*, defined in (6)), and $\mathbf{X}_l$ is the vector of activations of layer l. The decrease of the magnitude of $\partial E/\partial \mathbf{X}_l$ as l decreases may come from the weight matrix (*e.g.*, in general for pyramidal networks: size of layer l > size of layer l+1) producing a dispersion of the gradient and from the derivative of the sigmoid. In the case of the asymmetric sigmoid (in [0,1]) f'() has a maximum value of 0.25 (0.5 for symmetric sigmoid in [-1,1]) and decreases almost exponentially with the magnitude of the activations (getting away from 0 activation). However, 1) the gradient represents direction information, *i.e.*, what is important is the relative value of the gradient for different units, and 2) a reasonable conjecture is to assume that the relative magnitude of the gradient for two different layers is not an important information: the most important information concerns the relative change that each unit's activation within a layer should take in order to decrease the error.

According to that idea, we developped an algorithm that adapts the learning rate *of each layer* so as to make the average weight change for that layer equal to a user defined value (or one that may decrease with time, similarly to the learning rate for ordinary BP). Derthick (1984) had already proposed an algorithm in which each weight is updated by a fixed amount depending on the sign of the weight's derivative. However, in his case, one loses completely the information about the relative impact of changing one weight w.r.t. another one. On the other hand, with the layer-dependent learning rates, information about the relative importance of changing one weight *vs* another one on the same layer is kept and permits very fast convergence. Here is simple implementation of this idea for rate update:

$$\Delta W_{l+1,l}(t) = \varepsilon_l(t)\ \partial E/\partial W_{l+1,l}(t) \tag{18}$$

$$D_l(t) = \mu\ D_l(t) + (1-\mu) < |\ \Delta W_{l,l-1}(t)\ | > \tag{19}$$

$$\varepsilon_l(t) = \varepsilon_l(t-1)\ .\ \gamma(t)\ /\ D_l(t) \tag{20}$$

where is $\varepsilon_l(t)$ the learning rate of layer l at iteration t, $D_l(t)$ is a decaying average of $< |\ \Delta W_{l,l-1}(t)\ | >$, the average absolute value of the weight change for layer l, μ is the decay factor of $D_l(t)$ (between 0 and 1, typically 0.5), and $\gamma(t)$ is the target average weight change, which can be set by the user, or automatically reduced as training progresses.

In experiments with the XOR function, using a network with two inputs, one hidden and one output unit, and stochastic weight updates, the following results were obtained:

Table 3: Comparison of constant learning rate *vs* constant weight change per layer for the XOR problem. The standard deviation of the number of epochs is given in parenthesis The percentage of failures is the ratio of trials which had not converged after 1000 epochs.

	ε=0 5	ε=0 5	$<\lvert\Delta W\rvert>$=.05	$<\lvert\Delta W\rvert>$=.10	$<\lvert\Delta W\rvert>$=0.15
<#epochs>	622(238)	401(197)	35.9(18)	26.0(9.7)	34.9(22)
% failures	55%	15%	0%	0%	0%

All experiments were performed 20 times. The first two use the standard gradient descent with a fixed global learning rate. The last three use the above-described layer-dependent weight change, with a fixed target average weight change.

5. RADIAL BASIS FUNCTIONS MODELS

Radial Basis Function (RBF) units produce an output which depends on the distance between the input point and a prototype point in the input space (which is defined by the parameters of the RBF unit). The output of the network can be written as the weighted sum of the outputs of those RBF units:

$$Fi(\mathbf{X}) = \textstyle\sum_j W_{ij}\, h(\|\, \mathbf{X} - \mathbf{P}j \,\|) \tag{21}$$

$$h(r) = e^{-r^2} \tag{22}$$

where **X** is the input pattern vector, **P**j is the prototype point associated to RBF unit j and we can choose for h() for example a gaussian, as in (22). The norm in (21) can be weighted by a covariance matrix. Poggio and Girosi show [Poggio & Girosi 89] that if our objective is function approximation (approximate a $\mathbb{F}(\mathbf{X})$ with **F**(**X,** some parameters)), seen as hypersurface reconstruction, given a noisy training set of example patterns $\{\mathbb{F}p(\mathbf{X}p),\mathbf{X}p\}$ and a-priori knowledge that the resulting mapping should be smooth, then generalized RBFs satisfy sufficient conditions to be used for this approximation problem. These generalized RBFs are mathematically related to the well-known RBFs used for strict interpolation tasks. However, there are less RBF units in the proposed network than examples in the training set. These generalized RBFs are also closely related to methods such as Parzen windows, generalized splines and vector quantization. Note that the network defined by (21) constitutes a network with two layers of weights and can be shown to approximate arbitrarily well any smooth function [Poggio & Girosi 89].

5.1 Relation to Neurobiological Models

A multidimensional gaussian can be represented as the product of lower dimensional gaussians. This property suggests a way for neurons to possibly compute RBFs. Gaussian radial basis functions in one or two dimensions can be implemented as coarse coded receptive fields *i.e.*, one dimension is represented by an array of neurons, each reacting only to values of the variables in a certain range. These kinds of representations actually exist and were found in the visual system. Some special type of synapse has two incoming inputs and perform a kind of product of the two incoming signals [Durbin & Rumelhart 89]. Hence a RBF could be implemented with gaussian receptive fields and product synapses without explicitly computing the norm of the exponential [Mel & Koch 90]. New work by Kruschke (1990) seem to indicate that such local models have also some psychological plausibility.

5.2 Relation to Vector Quantization

RBFs are related to vector quantization (VQ) [Gray 84]: VQ partitions the input space into mutually disjoint regions (for example Voronoi polygons, separated by line segments at equal distance between each two neighboring cluster centers). VQ encoding approximates each input point by the nearest cluster center. This would be like having RBFs with all-or-none output, with only the closest RBF responding. Instead, RBF networks represent the input point by a vector of proximity measures between 0 and 1 for all the RBF units in the network.

RBFs are thus also related to Kohonen's neural network models for vector quantization [Kohonen 88]. These algorithms can be seen as special forms of the k-means algorithm [MacQueen 67] (often used for VQ) for finding the centers of n clusters in a set of points. In Kohonen's algorithms, which are examples of competitive learning algorithms, only one output unit is active at a time, the one "closest" to the input vector. In some of his algorithms (feature maps), the competitive units are laid out in a spatial structure in which a neighborhood is defined so that adjacent units will respond to similar vectors.

5.3 Implementation advantages of RBFs

The basic implementation advantages of RBFs derive from their representation: parameters have a simple meaning w.r.t. the pattern examples. Here is a possible fast training method to take advantage of this fact in simulations (see [Poggio & Girosi 89]):

1) Initialize the parameters of the gaussian units (**P**j in (21)) with a random subset of the examples, or with the results of a cluster analysis, such as the cluster centers produced by the k-means algorithm or Kohonen's LVQ2 algorithm [Kohonen *et al.* 89]. This step can be interpreted as an unsupervised, competitive learning step which encodes the input pattern in a local representation.

2) Find the output weights of the RBF network (Wij in (21)) with a matrix pseudo-inverse calculation, or with gradient descent. This step is a supervised learning step and can be accomplished rapidly since it is a linear problem, with no local minimum.

3) All the parameters of the system can be tuned to improve performance and perform a global optimization, using gradient descent since the RBF units outputs can be differentiated w.r.t. the parameters of these units (mean and possibly variance of the gaussian). See section 5.4.2 where it is shown that this step indeed improves performance.

5.4 Experiments of phoneme recognition with RBFs

The results of several experiments are presented here to evaluate the performance of RBFs in a difficult speech recognition problem. The task was phoneme recognition on the TIMIT [Garofolo 88] database, with 39 phoneme classes (as [Lee & Hon 89]). We used regions 2, 3 and 6 of TIMIT (135 speakers and 292623 frames for training and 28 speakers and 61428 frames for test). The input preprocessing produced 24 mel scaled spectral coefficients plus the energy for each frame of 20 ms, with a step of 10 ms. This resulted in a flow of 25 parameters per frame of 10 ms coming into the network. For the architectures presented below the hidden units look at 4 consecutive frames *i.e.*, a 100 dimensional input. Output units are sigmoid units rather than linear units as in (34). Note that this is equivalent since the sigmoid is invertible but it has the advantage that the outputs are limited to the range [0,1]. In all nets there was a bias unit feeding all sigmoid units in the net a constant 1.0 value.

It is difficult to compare directly the results obtained here with those reported by other researchers for phoneme recognition on TIMIT. The best result reported here was 41.8% error on the test set (section 5.4.5). This is a frame by frame error obtained by scanning the network on the preprocessed input sentence, and not using any language model, known segmentation, bigrams or duration information. Lee & Hon (1989) obtained 41.3% error with HMMs and no language model and 26.2% using a context-dependent bigram model. Leung and Zue (1990) obtained 30% error but used the known segmentation and phoneme duration (network is not placed across phoneme boundaries: it is centered on each phoneme). Robinson and Fallside (1990) obtained 24 9% error with a recurrent network plus a dynamic programming postprocessor using a bigram and duration model (the error is about twice that amount without the postprocessor).

5.4.1 Supervised *vs* unsupervised initialization

For the initialization of the gaussian units parameters, the k-means clustering algorithm was chosen. It generates a set of clusters with input patterns associated to each cluster. The variance of patterns within each cluster is used to initialize the spread (variance) of the gaussian units.

Should one use a **completely unsupervised** algorithm to find those parameters? What if some points from two classes form two clusters that mostly overlap? K-means will probably choose to represent them with only one cluster. A simple but not optimal solution to that problem is to apply k-means separately for each of the classes. For simplicity, the same number of cluster centers per class was always used. An experimental comparison of the use of k-means with no class information *vs* k-means per class showed a significant improvement with k-means per class. A simple experiment was performed with 78 gaussian hidden units.

Output weights were found with pseudo-inverse computation [Penrose 55]. The result with 78 clusters (no class information) was 58.1% error on the test set, while using class information, with 2 clusters per class (39 classes), the error was 52.2%. All other experiments with RBFs described in this chapter henceforth used k-means per class.

5.4.2 RBFs *vs* Sigmoid Units

The next set of experiments are comparative experiments in order to verify if the time gained with RBF networks instead of sigmoid units (mostly because of the non-random initialization) is not lost in performance. Experiments were performed on the same task (recognition of TIMIT phonemes) with the same inputs, same targets and same architecture for both a network of RBF units and one of sigmoid units. The networks had 78 hidden units.

The network with sigmoid hidden units was initialized with random weights and trained with BP for 22 epochs. The error on the test set was 51.2 % error. The network with gaussian hidden units was initialized with k-means, (2 clusters/class, 39 classes). The output weights were found with the pseudo-inverse. The error on the test set was 52.2 %. After doing 10 epochs of gradient descent on the RBF net, *i.e.,* optimizing all the parameters (output weights, cluster centers and cluster spreads), the error on the test set was reduced to 47.8 %.

Hence for much less CPU time (no gradient descent) the RBF network did almost as well as the sigmoid networks. However, with additional training (gradient descent) representing less than half of the CPU time used to train the sigmoid network, the RBF network performed better.

5.4.3 Effect of Context and Architecture

It is well known that successive speech spectra are not independent. Watrous (1989) showed in some simple examples how the addition of context can make feasible the separation of two classes which would otherwise overlap. However, a very large input window implies a large number of free parameters and may result in poor generalization on unseen data [Baum & Haussler 89]. Another problem lies with the non-linear distortions in time that occur among instantiations of a given phoneme. Various architectures are thus explored here in order to optimize generalization on the phoneme recognition problem, for example, using multiple delays between the hidden layer and the output layer (as in Waibel's TDNNs [Waibel *et al.* 87]).

Table 4: Performance on the TIMIT 39 phonemes recognition task for various sets of delays between the hidden layer and the output layer, using a static network of gaussian hidden units and initialization with k-means per class.

clusters/class	delays	error on test set
2	0	52.2%
2	0,4	48.6%
2	0,4,8	47.7%
2	0,2,4	46.5%
3	0,2,4	45.6%

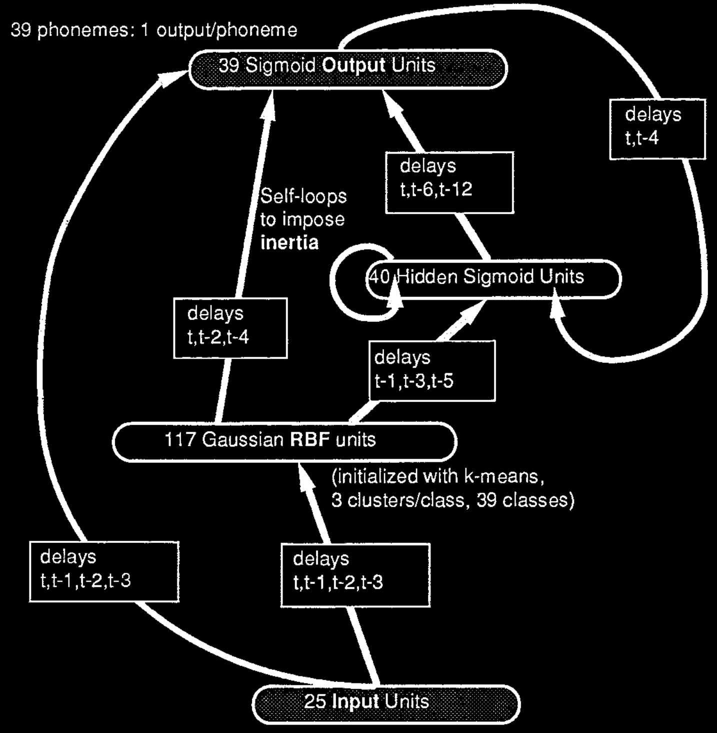


Figure 1. Recurrent neural network with Radial Basis Functions.

5.4.4 Adding a Recurrent Hidden Layer

The improvement shown in table 4 by providing more context with delay links show the importance of context for the recognition of phonemes. However, it seems natural to consider recurrent networks as a more powerful way to represent context (see section 3). This motivated the next experiment, in which a layer was added to the best net obtained in the previous section. The initial network had 3 clusters/class, *i.e.*, 117 gaussian hidden units, and delays of 0, 2 and 4 frames between these gaussian hidden units and the output units. The input units feed both the gaussian units and the output units with delays of 0, 1, 2 and 3 frames. The architecture after the addition of a second hidden layer with 40 sigmoid units is shown in figure 1. In the resulting network, the gaussians feed the hidden sigmoids with 3 delays (1, 3 and 5) and the outputs with 3 delays (0, 2 and 4) while the 40 hidden sigmoids feed the outputs with 3 delays (0, 6, and 12). Cycles are introduced in the network by the outputs feeding the hidden sigmoids with 2 delays (2, 4), and the hidden sigmoids having self-loops with a fixed decay weight of 0.93 which means a time constant (half-life) of 9.5 frames. This represents a very large network with about 54000 weights. The weights of the connections between the hidden sigmoid units and the output units were initialized to low values (in the range [-0.01, 0.01]) so as to not disturb too much the network by the introduction of this layer. However, after a few cycles of training, significant improvement was observed. After 11 epochs on the training set the error obtained with this network on the training set was 40.9% and the error and the test set was **41.8%**.

5.5 Acceleration with a Fast Search

This acceleration is obtained by taking advantage of the local nature of the response of the RBF units by searching rapidly for the subset of those units that can respond with an output significantly greater than zero. The chosen method is based on the use of a grid and table look-ups to find the appropriate subset of clusters. The major problem when using a grid method is that the size of the grid grows exponentially with the number of dimensions of the patterns. Thus the number of dimensions represented in the grid was reduced by using only the first few principal components of the input patterns (from the training set) to build the grid. A principal component analysis is performed once and for all on the training set and used to map input patterns to a low dimensional space in which it is much easier to perform the search. This mapping can be expressed as follows:

1) normalization: $X'ip = (Xip - \overline{Xi})/\sigma_{Xi}$ (23)

where Xip is the ith dimension of a pattern p, $\overline{\mathbf{X}}$ is the average over the training patterns and σ_{Xi} is the standard deviation of the training patterns (per dimension).

2) linear mapping through the principal components: $\mathbf{X}'' = A\,\mathbf{X}'$ (24)

where A is the matrix formed by concatenating the first few eigenvectors with highest eigenvalues of the matrix B, and B is formed by concatenating all the **X**'p (*i.e.*, all the normalized input patterns of the training set) as computed in (33). Note that X" has only of few dimensions (typically we chose about 3 dimensions, corresponding to the first 3 principal components).

In order to use grid cells more efficiently information about the pattern density in each of the reduced dimensions was taken into account in order to use storage more uniformly. The grid representation was improved by translating positions computed with (23) and (24) to the physical grid by using the following map. For each of the reduced dimensions d, a new position is computed from the old position as follows:

$X'''_d = F_d(X''_d)$ (25)

where $F_d(X''_d) = \sum_{i<X''_d} \text{density}_d(i)$ (26)

where $density_d(i)$ is the fraction of cluster centers falling in the i^{th} interval for dimension d of the principal components space. Hence a very fine quantization of the principal components space is performed in order to compute the density array. It only requires storage space linearly proportional to the number of dimensions (rather than exponential) because we consider each dimension independently. The approximate cumulative probability density F() is implemented as an array and (25) requires only one table look-up per dimension to produce the appropriate cell in the physical grid. The resulting grid is regular in the transformed space (after transformation by (25)) but is not regular in the principal components space. Instead, it attempts to use density information to produce less empty cells and a more uniform distribution of clusters among the cells. With this method, the initialization of the search data structure takes O(N) time and the search is proportional to the size of the retrieved list, *i.e.*, the number of selected RBF units.

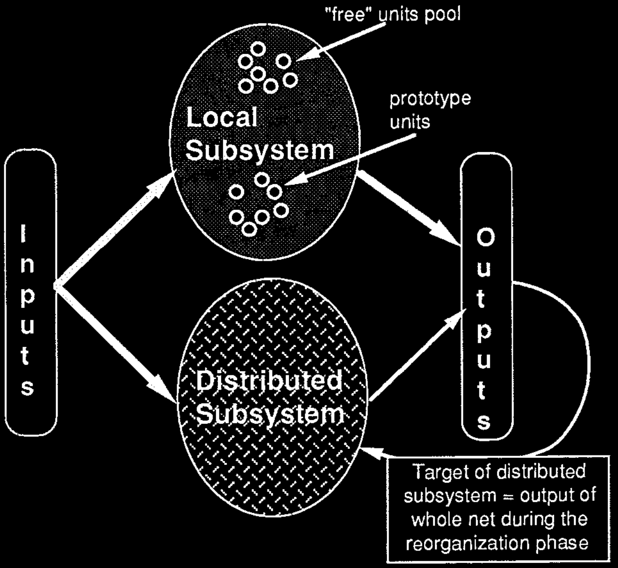

"free" units pool
prototype units
Local Subsystem
Distributed Subsystem
Inputs
Outputs
Target of distributed subsystem = output of whole net during the reorganization phase

Figure 2. Network combining both a local and a distributed subsystem.

6. COMBINING LOCAL AND DISTRIBUTED REPRESENTATIONS

In section 5.4.4 gaussian units and sigmoid units were combined in one network. This was motivated by the desire to combine the advantages of both representations (local and distributed). In this section we consider a more cognitively motivated alternative to combine these two complementary kinds of representations in the same network.

With a local representation, units respond to a specific and localized subset of the input patterns, *i.e.*, each unit represents a prototype and responds to patterns close in some metric space to that prototype (for example: Radial Basis Functions (RBF), Kohonen's LVQ and LVQ2 algorithms [Kohonen *et al.* 89] and other competitive learning algorithms). Usually, learning algorithms for these local representation networks are very fast. On the other hand, with a distributed representation, each hidden unit usually represents a "global" feature and it is the activation pattern over all the hidden units which represents an input pattern. These representations are thus more compact than local representations but in general this means that the learning task is more complex and needs more time, *i.e.*, many iterations on a training set (for example: backpropagation for networks of units computing the sigmoid of the weighted sum of their inputs).

It is interesting to note that humans appear to possess both a fast-learning and a slow-learning ability. We can remember for a long time a pattern seen only once (fast-learning), for example something seen at a highly emotional moment, but we may need a very long training period and a lot of drill to learn some other tasks (slow-learning), such as for professional expertise. These more difficult tasks probably involve creating compact internal representations that attempt to extrapolate and generalize over many situations.

Our current research work involves the design of constructive learning algorithms for architectures that combine both local and distributed representations. We consider a system with a "local representation" subsystem and a "distributed representation" subsystem. The whole network's output is formed by combining (*e.g.*, linearly) the outputs of both subsystems in such a way that the local subsystem has much more weight and thus has priority (whenever it produces a significantly non-zero output). Both subsystems take their input from the same set of units. The local subsystem is constructive and allocates new units when a new pattern generates a large error. The local subsystem however does not grow indefinitely because of a reorganization phase in which information about the input/output distribution that is represented in the local subsystem is compressed and gradually transferred from the local to the distributed subsystem.

A simple solution to implement this reorganization phase is the following. The fraction of the training set which has already been learnt by the network is sampled. In order to transfer information from the local to the distributed representation, the distributed subsystem alone should, without forgetting the examples it has already learnt, learn the examples "known" by the local subsystem. Thus the distributed subsystem is supervised by giving it as target the output of the whole network. As the distributed subsystem evolves a more complex model of the environment, local units which are redundant with the behavior of the distributed subsystem (within their local area of response) are made available to "store" new outlier patterns. The general organization of the network is depicted in figure 2.

7. CONCLUSION

In this chapter we have described some connectionist models and their application to speech recognition. We have described with examples some of the important phases of the design of these models that can take advantage of a-priori knowledge. We compared several preprocessing alternatives for connectionist models and found that a spectral representation produced better results than a cepstral representation and that an ear model yielded better results than the FFT.

Because speech recognition is an inherently sequential problem we have detailed several algorithms to train recurrent networks with back-propagation. We have discussed some considerations about accelerating convergence with gradient descent. In particular we have introduced a new method in which the average weight change of each layer is individually controlled. We have studied an alternative to the traditional networks of sigmoid units: RBF networks. For this local representation network we presented the results of several experiments on phoneme recognition which permit us to draw several conclusions. When initializing the RBF units with k-means, it is preferable to take into account information about the class of each pattern and perform k-means separately for each class. RBF networks performed as well or better than sigmoid networks but required less CPU time. Combining RBF units and sigmoid units in a recurrent network resulted in even better performance. This has brought us to propose in the last section a new algorithm that combines RBF units (local representation) and sigmoid units (distributed representation) in a way that is cognitively relevant.

REFERENCES

Baum E.B & Haussler D (1989), "What size net gives valid generalization?", Neural Computation 1, pp 151-160.

Bengio Y. & De Mori R (1988a), "Use of Neural Networks for the Recognition of Place of Articulation", ICASSP-88, pp 103-106

Bengio Y. & De Mori R (1988b), "Speaker Normalization and Automatic Speech Recognition Using Spectral Lines and Neural Networks", Proceedings of the Canadian Conference on Artificial Intelligence (CSCSI-88), pp 213-220

Bengio Y , De Mori R & Cardin R (1988),"Data-Driven Execution of Multi-Layered Networks for Automatic Speech Recognition", Proceedings of the American Association for Artificial Intelligence (AAAI-88).

Bengio Y , Cardin R , De Mori R & Merlo E. (1989a), "Programmable Execution of Multi-Layered Networks for Automatic Speech Recognition", in Communications of the Association of Computing Machinery, February 1989, vol 32, number 2, pp. 195-199

Bengio Y , Cosi P , Cardin R. & De Mori R. (1989b),"Use of Multi-Layered Networks for Coding Speech with Phonetic Features", in Advances in Neural Information Processing Systems 1, ed D.S. Touretzky, Morgan Kaufmann Publishers, pp 224-231

Bengio Y., Gori M & De Mori R. (1989c),"BPS: a Learning Algorithm for Capturing the Dynamic Nature of Speech", in the Proceedings of the International Joint Conference on Neural Networks 1989, pp II-417 - II-424

Bengio Y & De Mori R (1989),"Use of Multilayer Networks for the Recognition of Phonetic Features and Phonemes", in Computational Intelligence, 5, pp 134-141

Bengio Y , Cardin R., De Mori R (1990),"Speaker Independent Speech Recognition with Neural Networks and Speech Knowledge", in Advances in Neural Information Processing Systems 2, ed. D S Touretzky, Morgan Kaufmann Publishers

Bengio Y & De Mori R. (1990),"Recurrent Networks with Radial Basis Functions for Speech Recognition", presented at the Neural Networks for Computing, Snowbird, UTAH.

Bottou L , Fogelman F , Blanchet P., Lienard J S (1990), "Speaker-independent isolated digit recognition: multilayer perceptrons vs. dynamic time warping", Neural Networks 3(4), pp.453-465

Broomhead D S. & Lowe D. (1988), "Multivariable functional interpolation and adaptive networks", Complex Systems, 2: pp 321-355

Cosi P , Bengio Y. & De Mori R (1990), "Phonetically-based multi-layered neural networks for vowel classification", in *Speech Communications*, 9, 1990, pp 15-29.

Delgutte B & Kiang N.Y S (1984),"Speech coding in the auditory nerve", Journal of the Acoustical Society of America, no 75, pp 866-907

De Mori R., Bengio Y & Cosi P (1989),"On the Generalization Capability of Multi-Layered Networks in the Extraction of Speech Properties", in the Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-89), pp.1531-1536.

Derthick M. (1984), "Variations on the Boltzmann machine learning algorithm", tech rep CMU-CS-84-120, Pittsburgh, PA; Carnegie-Mellon University

Durbin R & Rumelhart D E (1989),"Product Units: A Computationally Powerful and Biologically Plausible Extension to Backpropagation Networks", Neural Computation, 1(1), pp 133-142.

Garofolo J S. (1988),"Getting started with the DARPA TIMIT CD-ROM: an acoustic phonetic continuous speech database", National Institute of Standards and Technology (NIST), Gaithersburgh, MD.

Giles C L., Sun G Z , Chen H H., Lee Y C & Chen D. (1990),"Higher Order Recurrent Networks & Grammatical Inference", in Advances in Neural Information Processing Systems 2, D S. Touretzky (ed), Morgan Kaufmann.

Gray R M (1984), "Vector Quantization", IEEE ASSP Magazine, April 1984, pp.4-29

Jacobs R.A , (1988), "Increased rates of convergence through learning rate adaptation", Neural Networks, vol. 1, no 4, pp 295-307

Kohonen T (1988), "Self-Organization and Associative Memory", Springer-Verlag, New-York, Second Edition

Kohonen T , Barna G & Chrisley R , (1989) "Statistical Pattern Recognition with Neural Networks: Benchmarking Studies", Technical Report, Laboratory of Computer and Information Science, Helsinki University of Technology, Finland

Kuhn G. (1987), " A first look at phonetic discrimination using a connectionist network with recurrent links", SCIMP working paper No 4/ 87, Communications Research Division, Institute for Defence Analysis, Princeton, NJ

Kruschke J.K. (1990), " ALCOVE: a connectionist model of category learning", Technical Report 19, Cognitive Science Program, Indiana University

LeCun Y (1989), "Generalization and network design strategies", in Connectionism in Perspective (Pfeifer, Schreter, Fogelman & Steels eds), North-Holland, pp.143-155

LeCun Y , Boser B , Denker J S., Henderson D., Howard R E , Hubbard W & Jackel L D. (1989),"Backpropagation Applied to Handwritten Zip Code Recognition", Neural Computation, 1(4): pp 541-551

Lee K F. & Hon H W (1989), "Speaker-Independent Phone Recognition Using Hidden Markov Models", IEEE Trans. on ASSP, vol. 37, no 11, pp 1641-1648.

Leung H C. & Zue V (1990), "Phonetic classification using multi-layered perceptrons", ICASSP 90, pp 525-528.

Lippmann R P. (1987), "An introduction to computing with neural nets", IEEE ASSP Magazine, 4(2), pp 4-22

Lippman R P. (1989),"Review of Neural Networks for Speech Recognition", Neural Computation, 1(1), pp 1-38

MacQueen J. (1967),"Some methods of classification and analysis of multivariate observations", in L M. LeCam & J Neyman eds , Proc 5th Berkeley Symposium on Math., Stat., and Prob., page 281, U California Press, Berkeley CA

Mel B W. & Koch C (1990), "Sigma-Pi Learning: A Model for Associative Learning in Cerebral Cortex", Advances in Neural Networks Information Processing Systems 2, ed D.S. Touretzky, Morgan Kauffman

Penrose R. (1955), "A generalized inverse for matrices", Proc. Cambridge Philos. Soc., 51: pp 406-513.

Poggio T & Girosi F (1989), "A Theory of Networks for Approximation and Learning", MIT A.I. Memo No. 1140.

Pomerleau D A (1989),"ALVINN: An Autonomous Land Vehicle in a Neural Network", Advances in Neural Networks Information Processing Systems 1, ed D S Touretzky, Morgan Kauffman, pp 305-313

Robinson T. & Fallside F. (1990), "Phoneme recognition from the TIMIT database using recurrent error propagation networks", CUED/F-INFEG/TR 42, Cambridge University Engineering dept

Rumelhart D E., Hinton G , Williams R J. (1986), "Learning internal representation by error propagation ", in Parallel Distributed Processing, vol 1, (eds Rumelhart & McClelland), Bradford Books / MIT Press

Rumelhart D E & McClelland (eds)(1986), Parallel Distributed Processing, vol 1, Bradford Books / MIT Press

Seneff S. (1985),"Pitch and spectral analysis of speech based on an auditory synchrony model", RLE Technical report 504, MIT.

Waibel A., Hanazawa T., Hinton G , Shikano K. & Lang K. (1987),"Phoneme Recognition using Time Delay Neural Networks", Technical Report TR-I-0006, ATR Interpreting Telephony Research Laboratories

Waibel A (1989),"Modularity in neural networks for speech recognition", Advances in Neural Networks Information Processing Systems 1, ed. D.S. Touretzky, Morgan Kauffman, pp 215-223.

Waibel A., Hanazawa T. & Shikano K. (1988),"Phoneme recognition: neural networks vs hidden Markov models", ICASSP-88, pp 107-110.

Watrous R (1989), "Context-Modulated Discrimination of Similar Vowels using Second-Order Connectionist Networks", tech rep University of Toronto

Williams R J. & Zipser D (1988),"A Learning Algorithm for Continuously Running Fully Recurrent Neural Networks", ICS Report 8805, Institute for Cognitive Science, University of California, San Diego.

Artificial Neural Networks and Statistical Pattern Recognition
Old and New Connections
I.K. Sethi and A K Jain (Editors)
© 1991 Elsevier Science Publishers B,V All rights reserved

DYNAMIC ASSOCIATIVE MEMORIES

Mohamad H. Hassoun

Department of Electrical and Computer Engineering
Wayne State University
Detroit, Michigan, 48202, USA

1. INTRODUCTION

Associative learning and retrieval of information in parallel neural-like systems is a powerful processing technique with a wide range of applications ranging from content addressable memories to robust pattern classification and control. Dynamic associative memories (DAMs) are a class of artificial neural networks which utilize a supervised recording/learning algorithm to store information as stable memory states, thus realizing mapping between a set of key/target memory pairs. The retrieval of the stored memories is accomplished by first initializing the DAM state with a noisy or partial input pattern (key) and then allowing the memory to perform a collective relaxation search to find the closest associated stored memory.

DAMs are characterized by a regular layered architecture of highly distributed and densely interconnected processing layers with feedback. Each processing layer consists of a set of non interacting nodes; each node receives the same set of data (input pattern or output from a preceding layer), processes this data, and then broadcasts its single output to the next processing layer. The transfer function of a given DAM node can vary in complexity; however, all nodes are assumed to have the same functional form. The most common node transfer function is equivalent to a weighted sum of the input data followed by a nonlinear activation function. The weighted sum processing step represents a local identification of the input data based on a similarity computation (or projection) between the data vector and a locally-stored weight vector. The nodes' weight vectors also describe an interconnection (communication) pattern between the nodes of adjacent layers. The node weights are assumed to be synthesized, during a learning/recording session, from a given training set. On the other hand, a node's activation function is usually a monotone-increasing function with saturation (e g., a tanh or a unit-step function) which can be thought of as implementing a "local decision" on the preceding similarity computation. In theory, DAM mapping dynamics can be understood and controlled through the network architecture, the learning/recording algorithm used, and the encoding of stored associations.

Several associative neural memories have been proposed over the last two decades [1-10]. These memories can be classified in various ways depending on their retrieval mode (dynamic vs. static and/or synchronous vs. asynchronous), the nature of the stored associations (autoassociative vs. heteroassociative and/or binary vs. continuous), the type of training algorithm (adaptive vs. nonadaptive), or the complexity and capability of the training algorithm. In this chapter, dynamic synchronous binary-state neural memories are emphasized. These memories have been extensively studied and analyzed by several researchers [11-31].

This chapter is intended as a review of the fundamental concepts relating to basic DAM architectures, the various training algorithms and recording strategies, and DAM capacity and performance. Section 2 presents the basic architectures, transfer characteristics, and general retrieval dynamics for auto- and heteroassociative DAMs. Section 3 summarizes several desirable characteristics of associative memories which serve as DAM performance measures. Several DAM recording/learning algorithms, including correlation, generalized inverse, and Ho-Kashyap training algorithms and variations, are presented and analyzed in Section 4. General training strategies for controlling and enhancing DAM dynamics are discussed in Section 5. In Section 6, DAM capacity and retrieval dynamics are presented and compared for several recording/learning techniques.

2. DAM ARCHITECTURES AND GENERAL MEMORY DYNAMICS

The simplest associative neural memory architectures exhibiting dynamical behavior are considered and their transfer characteristics are formulated. Potential DAM state-space trajectories are also outlined. This section deals with two basic DAM architectures: autoassociative and heteroassociative. Some important effects of various activation functions and state update strategies on DAM stability and dynamics are also presented.

2.1 Autoassociative DAM

The autoassociative DAM is basically a Hopfield [14] memory employing a single-layer of perceptrons with hard-clipping activations. The perceptrons are fully interconnected through a feedback path, as shown in Figure 1(a), and are assumed to operate in a synchronous (parallel) retrieval mode. Figure 1(b) depicts a block diagram of such an autoassociative DAM. Theoretically, the interconnection weight matrix **W** has real valued components w_{ij} connecting the jth perceptron to the ith perceptron. It is to be noted that, due to the hard-clipping nature of the activation function operator **F** and the presence of feedback, the memory can only store and retrieve binary memories.

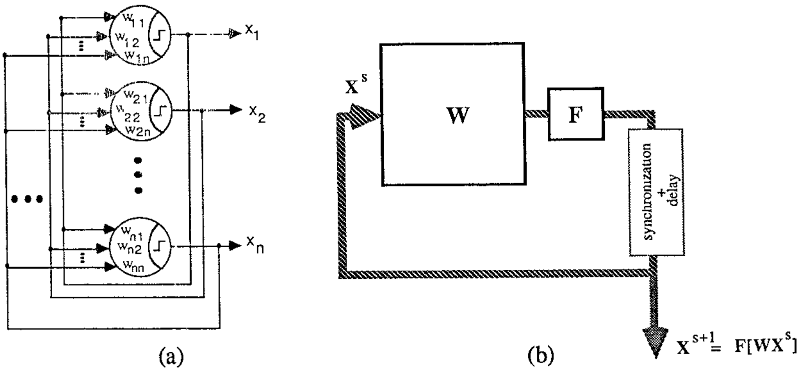


Figure 1. (a) Interconnection pattern for an n-perceptron autoassociative DAM. (b) A block diagram representation of the DAM in (a).

Let the DAM output of Figure 1 be represented by an n-dimensional binary valued pattern (column vector) $\mathbf{x}^s$ and call it the state of the DAM at discrete time "s." Hence, the DAM state evolves according to the difference equation:

$$\mathbf{x}^{s+1} = \mathbf{F}\left[\mathbf{W}\,\mathbf{x}^{s}\right] \tag{2.1.1}$$

where $\mathbf{F}$ operates component-wise on its n-dimensional vector argument. The operator $\mathbf{F}[\mathbf{W}\,\mathbf{x}(.)]$ is referred to as the state transition operator. The weight matrix and the threshold activations are computed during a training session in such a way as to store a set of m binary (bipolar or unipolar) patterns $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_k, ..., \mathbf{x}_m\}$ satisfying the condition $\mathbf{x}^{s+1} = \mathbf{x}^s = \mathbf{x}_k$; i.e., synthesized $\mathbf{W}$ and $\mathbf{F}$ guarantee that $\mathbf{x}_k$ is a fixed DAM state. The training pattern $\mathbf{x}_k$ will be referred to as a fundamental memory. All other fixed states which are not fundamental memories will be referred to as spurious memories. In addition to the above two types of dynamics, the DAM can also converge to a limit cycle. It has been shown by Fogelman [32] that if $\mathbf{W}$ is symmetric, the above DAM has limit cycles of period 2, at most.

The autoassociative mapping performed by the above DAM may seem trivial. However, the DAM is intended to act as a filter which corrects noisy, distorted, and/or partial input versions, $\mathbf{x}'_k$, of the fundamental memories $\mathbf{x}_k$. Theoretically, the DAM converges to $\mathbf{x}_k$, when initialized with $\mathbf{x}^0 = \mathbf{x}'_k$. This suggests that a *basin of attraction* exists around each one of the fundamental memories Under certain conditions, discussed later in this chapter, the above DAM is capable of realizing such basins of attraction. Unfortunately, the complex dynamics of the DAM also give rise to attractor spurious memories, thus degrading performance. These and additional DAM characteristics are considered in Section 3.

2.2 Heteroassociative DAM

A heteroassociative DAM [8-10,28] may be thought of as an extension of the autoassociative DAM described above. Here, two single-layer feed-forward neural nets are connected in a closed loop as shown in Figure 2 . This architecture allows for simultaneous hetero- and autoassociative recollection of stored data. Ideally, a heteroassociative DAM realizes the two mappings $\mathbf{M}$ and $\mathbf{M^*}$ between a set of m binary input patterns $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_k, ..., \mathbf{x}_m\}$ and another corresponding set of m output patterns $\{\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_k, ..., \mathbf{y}_m\}$ according to:

$$\mathbf{M}: \ \mathbf{x}_k \Rightarrow \mathbf{y}_k \ \text{ and } \mathbf{M}^*: \ \mathbf{y}_k \Rightarrow \mathbf{x}_k;\ k = 1, 2, ..., m \tag{2.2.1}$$

The above DAM consists of a forward processing path and a feedback processing path. The forward path, considered alone, constitutes a unidirectional (static) heteroassociative memory that is potentially capable of realizing the mapping $\mathbf{M}$ of equation (2.2.1) by recalling $\mathbf{y}_k$ from $\mathbf{x}_k$ according to :

$$\mathbf{y}_k = \mathbf{F}\left[\mathbf{W}_1\,\mathbf{x}_k\right]\ ;\ k = 1, 2, ..., m \tag{2.2.2}$$

where $\mathbf{y}_k$ and $\mathbf{x}_k$ are assumed to be binary column vector patterns of dimensions L and n, respectively, and $\mathbf{W}_1$ is an L x n weight matrix which is assumed to be computed during a training session. $\mathbf{F}$ is the same activation function operator defined earlier.

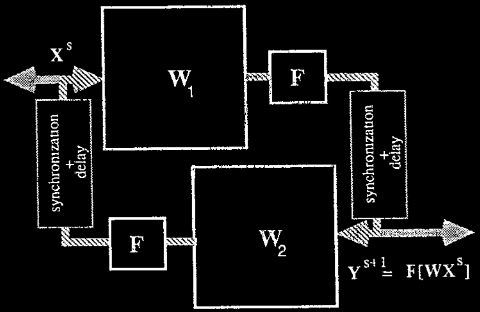


Figure 2. A block diagram representation of a dynamic heteroassociative neural memory.

One of the most appealing features of an associative memory is its ability to tolerate noisy and/or partial input; that is, given an input x_k' that is *similar* to the pattern x_k of the stored association pair $\{x_k, y_k\}$, the memory will respond with the correct association y_k according to:

$$y_k = F\left[W_1 x_k'\right] \tag{2.2.3}$$

However, the above equation may not hold true when relatively large numbers of associations are stored and/or the test input pattern is *slightly similar* to x_k. This problem can be partially alleviated in the case of autoassociative retrieval ($y_k = x_k$) by feeding the output of the unidirectional memory directly into the input and simultaneously removing the original input x'_k. This gives rise to the autoassociative DAM architecture of Figure 1. However, in the heteroassociative case (x_k has a different size and/or encoding than y_k), direct feedback is not compatible and a natural and simple remedy would be to feed the output back through the inverse **M** mapping, **M***, defined in Equation (2.2.1) or, explicitly, by the equation:

$$x_k = F\left[W_2 y_k\right] \; ; k = 1, 2, \ldots, m \tag{2.2.4}$$

where W_2 is an n x L real valued matrix. The resulting DAM is the one shown in Figure 2

Now we are ready to write the nonlinear difference equations governing the dynamics of the heteroassociative DAM. These equations are given by:

$$y^{s+1} = F\left[W_1 F\left[W_2 y^s\right]\right] \tag{2.2.5}$$

$$x^{s+1} = F\left[W_2 F\left[W_1 x^s\right]\right] \tag{2.2.6}$$

where s is the iteration number, x^0 is the initial state, and y^0 is given from x^0 through equation (2.2.2). The DAM dynamics can also be completely described through equations

(2.2.4) and (2.2.5) or (2.2.2) and (2.2.6). These equations suggest that the DAM is a two-output (y_k, x_k) dynamic system, as shown in Figure 2. The first output represents heteroassociative recollections and the second output represents autoassociative recall. The physical interpretation of these two outputs is determined by the application at hand. This class of DAMs is potentially powerful in robust pattern classification and pattern identification applications. Here, the output y_k may encode a classification of the test input x' or it may encode a specific *action* or *decision*. On the other hand, the output x_k gives a reconstruction/correction of the input pattern; this output may be used as an identification output which verifies the classification/action output by acting as a confidence measure. The x_k output process may also be viewed as a filtering process.

2.3 Other DAM Variations

In the above, the architectures of the synchronous (parallel update) discrete DAM were described, for both auto- and heteroassociative retrieval. Variations on such architectures for autoassociative DAMs have been proposed and analyzed in the literature. These variations deal with DAM state update strategies, and assume various types of activation functions. The dynamics and stability of such DAMs are highly affected by these variations.

In his original neural memory model, Hopfield [14] employs asynchronous (random) updating in the autoassociative DAM of Figure 1 Each perceptron is assumed to update its binary state stochastically independent of the times of firing of the other n - 1 neurons in the network. This asynchrony was introduced in order to model the propagation delays and noise in real neural systems. The discrete dynamics of this model are given by:

$$x_i^{s+1} = f\left(\mathbf{W}\,x^s + I_i\right) \tag{2.3.1}$$

where I_i is an external bias, which exists for all times "s." The perceptron label "i" in equation (2.3.1) is stochastically determined, and thus allows only one perceptron to change its activity in the transition from time s to s+1 Hopfield, by employing a discrete-time energy function approach, showed that a sufficient condition for the stability (no oscillations) of DAMs with the dynamics of equation (2.3.1) is to have a symmetric zero-diagonal interconnection matrix **W**. Fogelman [32] arrived at a similar result, which states that it is sufficient to have a symmetric non-negative diagonal **W** for stability, assuming random or sequential (elements change state one at a time in a prescribed fixed order) perceptron update.

It was also shown by Fogelman that when the sharp threshold activation function (Figure 3(a)) is replaced by a saturation piece-wise linear function, shown in Figure 3(b), the resulting randomly or sequentially updated DAM is also stable if **W** is symmetric with a non-negative diagonal. Golden [33] and Greenberg [34] extend the above stability results to the "brain-state-in-a-box" (BSB) [35] DAM described by the dynamics:

$$\mathbf{x}^{s+1} = \mathbf{F}\left[\rho\mathbf{W}\,\mathbf{x}^s + \mathbf{x}^s\right] \tag{2.3.2}$$

where ρ is a positive constant (step size) and **F** is the activation function operator shown in Figure 3(b). It was shown by Greenberg that if **W** is symmetric and diagonal-dominant (w_{jj} is larger than the sum of the absolute value of all off-diagonal elements w_{ij}, for $j = 1, 2, \ldots, n$) then the BSB DAM is stable and that the only stable points are the corners of the n-cube.

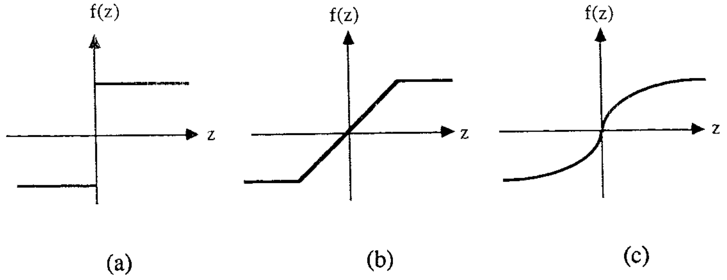


Figure 3. Various types of perceptron activation functions employed in DAMs: (a) hard clipping, (b) saturation piece-wise linear, and (c) sigmoidal activations.

In his continuous DAM model, Hopfield [36] assumes an analog electronic amplifier-like implementation of a perceptron which results in the following deterministic DAM retrieval dynamics:

$$\rho\frac{dv(t)}{dt} = \mathbf{W}\mathbf{x}(t) - \sigma\, v(t) + \mathbf{I}\,;\ \text{with}\ \ v = \mathbf{F}^{-1}(\mathbf{x}) \tag{2.3.3}$$

where ρ and σ are positive constants and the activation operator **F** takes the form of a sigmoid function (e.g , $\tanh(\beta v)$) as depicted in Figure 3(c) above. Here, $\mathbf{F}^{-1}$ is the inverse of **F**; that is, the ith component of v is given by $v_i = f^{-1}(x_i)$. Hopfield has shown that if **W** is symmetric, then the continuous autoassociative DAM is stable. Furthermore, if the amplifier gains (slopes of the activation functions) are very large, then the only stable states of the continuous DAM have a simple correspondence with the stable states of the stochastic Hopfield DAM described above. Marcus and Westervelt [37] have investigated a synchronous discrete-time variation of the continuous Hopfield model having the same form as equation (2.1.1). It was shown that if **W** is symmetric and if the activation functions are single-valued, monotonically increasing, and rise less rapidly than linearly for large arguments, then all attractors are either fixed points or period-two oscillations. Furthermore, if the system obeys the condition $\beta < |1/\lambda_{min}|$, where $\beta > 0$ is the maximum slope of the activation function and λ_{min} the most negative eigenvalue of **W**, then all period-two oscillations are eliminated and convergence to stable attractors is guaranteed.

3. CHARACTERISTICS OF A HIGH-PERFORMANCE DAM

A trained DAM is expected to exhibit a number of characteristics, such as noise and/or distortion tolerance, high capacity of stored associations, and well-behaved dynamics. In general, after training a DAM, a set of fundamental memories are recorded which are expected to behave as stable attractive states of the system. However, and in addition to the recorded fundamental memories, spurious and/or oscillatory attractors can exist which negatively affect the performance of a DAM. A spurious attractor is a stable memory which is not part of the recorded memories. These spurious memories are not desirable, yet they exist in all of the DAMs discussed above. Therefore, if one cannot train a DAM to exhibit no spurious states, then it is highly desirable to have them exist in a region of state-space that is far from regions of fundamental memories. Oscillations may be controlled by employing some of the DAM variations discussed in Section 2.3.

A set of performance characteristics must be met in order for a DAM to be efficient for associative processing applications. Depending on the encoding of the training memory associations, two classes of DAM mappings are distinguished: linearly separable and nonlinearly separable. Due to its single-layer architecture, the autoassociative DAM of Figure 1 can realize only linearly separable mappings; i.e., it can only store linearly separable memory vectors. This is also true for the heteroassociative DAM of Figure 2, since the stability of a given association pair hinges on the ability of the single-layer forward and backward subnets to realize perfectly the m $\mathbf{x}_k$-to-$\mathbf{y}_k$ and $\mathbf{y}_k$-to-$\mathbf{x}_k$ associations, respectively. On the other hand, multiple-layer architectures [38-40] are needed to store a set of nonlinearly separable associations (auto- or heteroassociations). Multiple-layer DAMs are more difficult to analyze than single-layer ones and are not considered here. In the rest of this chapter, training associations are assumed to be linearly separable.

The following is a summary of some of the important characteristics of a DAM: (1) Tolerance to noisy, partial, and distorted inputs. This implies a high probability of convergence to fundamental memories. (2) High capacity of stored memories. (3) Existence of relatively few spurious memories and few or no orbits, and a low convergence to such states. (4) Convergence within a few retrieval cycles. (5) Provision for a *no decision* state. DAM inputs with relatively low signal-to-noise ratios must have a high probability of convergence to this state. (6) Autoassociative and heteroassociative processing capabilities. Depending on the nature of the application, association of identical or distinct data vectors may be required. Some of these desirable dynamics are depicted in Figure 4(a) for an autoassociative DAM. On the other hand, Figure 4(b) depicts the state-space of a low-performance DAM.

These characteristics can be used to compare different DAM architectures and/or recording (or learning) algorithms. It is to be noted that, with given memory associations and architecture, all of the above characteristics are dependent on the recording algorithm used.

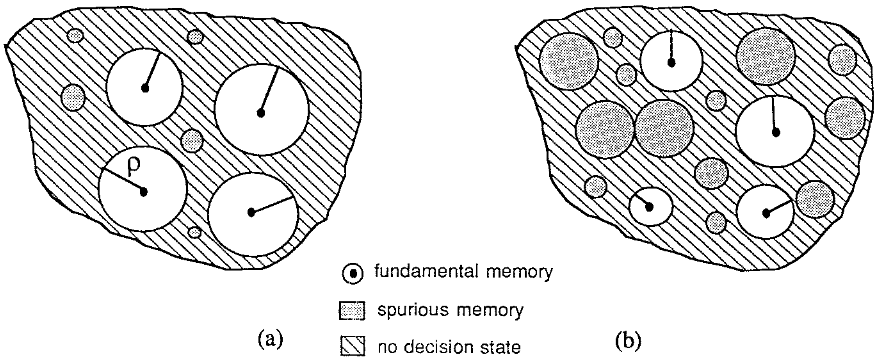


Figure 4. A conceptual diagram comparing (a) high-performance and (b) low-performnce autoassociative DAMs.

4. ASSOCIATIVE LEARNING IN A DAM

In theory, there exist an infinite number of interconnection matrices and thresholds that realize the mapping of a fixed set of associations in a DAM. However, different solutions may lead to different DAM dynamics that affect storage capacity, convergence rates to fundamental memories, number and location of spurious attractors, number and location of orbits, robustness, and other DAM characteristics. For high-performance DAMs, the best solution is the one that gives rise to the desirable characteristics discussed in Section 3. Such a solution is very difficult to achieve, since it requires an optimization process that involves many constraints and parameters and which may be too complicated and computationally expensive to implement. Another alternative is to synthesize interconnection weights that will guarantee a perfect recording of only the $\{x_k\}$ memories ($\{x_k, y_k\}$ associations in the heteroassociative case) and hope that such a solution will also give rise to acceptable DAM performance. In fact, all of the existing DAM recording/learning techniques proposed in the literature are based on this latter approach.

The training phase is responsible for synthesizing the interconnection matrix **W** from a training set of associations of the form $\{x_k, y_k\}$, for $k = 1, 2, \ldots, m$, for the heteroassociative case (notice that autoasociative training can be arrived at as a special case of heteroassociaitve training by setting $y_k = x_k$). Here, x_k and y_k belong to the n and L dimensional binary spaces, respectively. Therefore, the objective here is to solve the set of equations

$$y_k = W x_k; \quad k = 1, 2, \ldots, m \tag{4.0.1}$$

or in matrix form,

$$Y = WX \tag{4.0.2}$$

where $Y = [\, y_1\, y_2 \ldots y_k \ldots y_m \,]$ and $X = [\, x_1\, x_2 \ldots x_k \ldots x_m \,]$. The assumption of binary valued associations and the presence of a clipping nonlinearity **F** operating on **WX** relaxes some of the constraints imposed by equation (4.0.2); that is, it is sufficient to solve the equation:

$$Z = W X; \text{ and } Y = F[Z] \tag{4.0.3}$$

Next, several DAM training techniques will be derived and analyzed.

4.1 Correlation Recording

One of the earliest associative memory recording techniques is the correlation technique [2-4] which was originally proposed for the synthesis of a linear associative memory (LAM). This is a simple recording technique for generating **W** according to:

$$W = Y X^T \tag{4.1.1}$$

where "T" is the transpose operator. This is a *direct* method for computing the correlation weight matrix **W** which assumes that all the associations are present simultaneously during recording. A more practical method for computing the correlation matrix is to use the following equivalent form of equation (4.1.1):

$$W = \sum_{k=1}^{m} y_k x_k^T \tag{4.1.2}$$

where the term inside the summation is the outer product of the vectors $\mathbf{y}_k$ and $\mathbf{x}_k$. This in turn allows us to derive the *adaptive* correlation recording technique according to:

$$\mathbf{W}^{\text{new}} = \mathbf{W}^{c} + \mathbf{y}_k \mathbf{x}_k^{T}; \quad \text{for } k = 1, 2, \ldots, m \tag{4.1.3}$$

where $\mathbf{W}^c$ is the current weight matrix (initialized as the zero matrix). This makes it very convenient, if at some time after the initial recording phase is complete, we want to add a new memory or delete an already recorded memory.

Let us now investigate the requirements on the $\{\mathbf{x}_k,\mathbf{y}_k\}$ associations which will guarantee the successful retrieval of all recorded memories $\mathbf{y}_k$ from perfect key inputs $\mathbf{x}_k$. Employing equation (4.0.1) and (4.1.2) and assuming that the key input $\mathbf{x}_h$ is one of the $\mathbf{x}_k$ vectors, we get an expression for the retrieved pattern as:

$$\tilde{\mathbf{y}}_h = \left[\sum_{k=1}^{m} \mathbf{y}_k \mathbf{x}_k^{T}\right]\mathbf{x}_h = \sum_{k \neq h}^{m} \left(\mathbf{y}_k \mathbf{x}_k^{T}\right)\mathbf{x}_h + \mathbf{y}_h \mathbf{x}_h^{T}\mathbf{x}_h = \sum_{k \neq h}^{m} \left(\mathbf{y}_k \mathbf{x}_k^{T}\mathbf{x}_h\right) + \mathbf{y}_h \|\mathbf{x}_h\|^2 \tag{4.1.4}$$

The first term in equation (4.1.4) represents the "cross-talk" between the input key $\mathbf{x}_h$ and the remaining $m - 1$ $\mathbf{x}_k$ patterns. This term can be reduced to zero if the $\mathbf{x}_k$ vectors are orthogonal. The second term is proportional to the desired memory $\mathbf{y}_h$ with a proportionality constant equal to the square of the norm of the key vector $\mathbf{x}_h$. Hence, the necessary and sufficient condition on the retrieved memory to be the desired perfect recollection is to have *orthonormal* key vectors $\mathbf{x}_k$, and is independent of the encoding of the $\mathbf{y}_k$ (note how the $\mathbf{y}_k$ affects the cross-talk if the $\mathbf{x}_k$ are not orthogonal). However, recalling the nonlinear nature of the DAM reflected in equation (4.0.3), perfect recall of the binary $\mathbf{y}_k$ vectors is in general possible even when the key vectors are only pseudo-orthonormal. The correlation recording of autoassociations is identical to the above but with $\mathbf{y}_k = \mathbf{x}_k$.

Hopfield [14] uses a slightly modified correlation technique for recording his proposed autoassociative DAM, for improved performance. The Hopfield memory recording recipe transforms the unipolar binary $\mathbf{x}_k$ vectors into bipolar vectors before recording, and assumes no perceptron self-connections (i.e., $\mathbf{W}$ has zero diagonal), according to:

$$\mathbf{W} = \sum_{k=1}^{m} (2\mathbf{x}_k - \mathbf{1})(2\mathbf{x}_k - \mathbf{1})^{T} - \operatorname{diag}\left\{\sum_{k=1}^{m} (2\mathbf{x}_k - \mathbf{1})(2\mathbf{x}_k - \mathbf{1})^{T}\right\} \tag{4.1.5}$$

where $\mathbf{1}$ is an n-dimensional column vector of 1's. Here, a hard-clipping nonlinearity with a threshold of zero may be used as the perceptron activation function. Weisbuch and Fogelman [41] propose an optimal off-line method for choosing these thresholds.

4.2 Generalized-Inverse Recording

The correlation associative recording technique is restrictive in many applications, due to the requirement that the $\mathbf{x}_k$ be orthonormal. This technique does not make optimal use of the DAM interconnection weights. A more optimal recording technique is the generalized-inverse recording technique proposed by Kohonen [5]. The generalized inverse technique was originally proposed for the synthesis of the **W** matrix of the optimal linear associative memory (OLAM) [15], employing perceptrons with linear activations and no feedback. Starting from equation (4.0.2) and multiplying both sides of the equation by $\mathbf{X}^T$ one gets:

$$\mathbf{Y}\mathbf{X}^T = \mathbf{W}\mathbf{X}\mathbf{X}^T \tag{4.2.1}$$

The motivation behind the choice of the multiplier $\mathbf{X}^T$ is that it makes equation (4.2.1) consistent with the correlation solution for **W** (equation 4.1.1) for the special case of orthonormal $\mathbf{x}_k$, which make $\mathbf{X}\mathbf{X}^T = \mathbf{I}$. It can also be shown [42] that the **W** which satisfies equation (4.2.1) is the mean-square-error (MSE) solution (here we assume that the training set has more associations than the number of components of $\mathbf{x}_k$; i.e., $m > n$) that minimizes the objective function J(**W**) given by:

$$J(\mathbf{W}) = \| \mathbf{Y} - \mathbf{W}\mathbf{X} \|_E^2 = \mathrm{tr}\left[(\mathbf{Y} - \mathbf{W}\mathbf{X})^T(\mathbf{Y} - \mathbf{W}\mathbf{X})\right] = \sum_{k=1}^{m}\sum_{i=1}^{L}\left(\varepsilon_i^k\right)^2 \tag{4.2.2}$$

where $\|.\|_E$ is the Euclidian norm, "tr" is the trace operator, and ε_i^k is the error between the ith components of the estimated and desired vector $\mathbf{y}_k$. Going back to equation (4.2.1) and multiplying both sides of the equation by the inverse of $\mathbf{X}\mathbf{X}^T$, the following solution for **W** is achieved:

$$\mathbf{W} = \mathbf{Y}\mathbf{X}^T\left(\mathbf{X}\mathbf{X}^T\right)^{-1} = \mathbf{Y}\mathbf{X}^+ \tag{4.2.3}$$

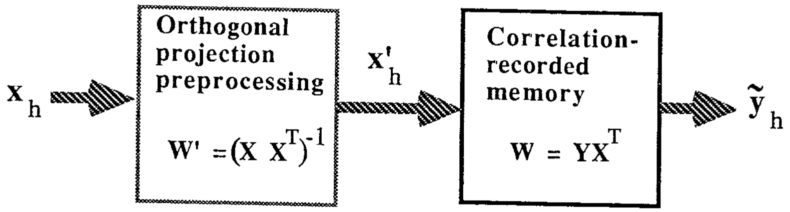
where $\mathbf{X}^+$ is the pseudo- or generalized-inverse of **X**. This solution is only valid if the inverse of $\mathbf{X}\mathbf{X}^T$ exists, which requires that the rows of $\mathbf{X}\mathbf{X}^T$ be linearly independent. Also, note that for an arbitrary **Y**, a sufficient condition for an exact solution for **W** is that $\mathbf{X}^+\mathbf{X} = \mathbf{I}$, which means that the $\mathbf{x}_k$ are linearly independent (compare this to the more restrictive orthonormal condition on $\mathbf{x}_k$ for correlation recording). However, if binary associations are assumed, then the use of hard-clipping according to equation (4.0.3) can relax the linear independence condition on $\mathbf{x}_k$, for an exact solution.

Next, let us investigate the retrieval characteristics of the OLAM, presented by an input $\mathbf{x}_h$, by substituting equation (4.2.3) in (4.0.1) and arrive at:

$$\tilde{\mathbf{y}}_h = \mathbf{W}\mathbf{x}_h = \mathbf{Y}\mathbf{X}^T\left[\left(\mathbf{X}\mathbf{X}^T\right)^{-1}\mathbf{x}_h\right] \tag{4.2.4}$$

which shows that the OLAM can be viewed as a correlation recorded associative memory with a preprocessing stage attached, as shown in Figure 5. The preprocessing stage performs an orthogonal projection [15], defined by the term inside brackets in equation (4.2.4), of $\mathbf{x}_h$ onto the space spanned by the $\mathbf{x}_k$ vectors. Again, if the $\mathbf{x}_k$ are linearly independent, then the preprocessing block in Figure 5 maps the hth training vector $\mathbf{x}_h$ into a vector $\mathbf{x}'_h$ which is orthogonal to the $\mathbf{x}_k$ (k different from h) vectors stored in the correlation memory block and with $\mathbf{x}_h^T\,\mathbf{x}'_h = 1$, thus outputting $\mathbf{y}_k$.

It was assumed in the above discussion that the training set is over-determined, m > n. It is shown later in Section 6 that the number of distinct memories m must be smaller than the



x_h
Orthogonal projection preprocessing
$W' = (X X^T)^{-1}$
x'_h
Correlation-recorded memory
$W = YX^T$
$\tilde{y}_h$

Figure 5. A block diagram of the OLAM showing its decomposition into a correlation associative memory in cascade with an orthogonalization transformation.

dimension n, if DAM overloading is to be avoided. Therefore, it is important also to consider the problem of recording an under-determined (m < n) set of associations. When m < n, equation (4.0.2) has multiple exact solutions **W***. Here, the minimum-norm solution $\mathbf{W} = \min \|\mathbf{W}^*\|_E$ is selected as the solution leading to the most robust associative retrieval [15]. Assuming that the x_k are linearly independent, a direct computation of **W** is given by:

$$\mathbf{W} = \mathbf{Y}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T = \mathbf{Y}\mathbf{X}^+ \tag{4.2.5}$$

Again, for this under-determined case, associative retrieval can be thought of as that of a correlation-recorded memory with preprocessed inputs. To see this, the identity $X^+ = X^T(XX^T)^+$ is used in equation (4.2.5), which upon substitution in equation (4.0.1) gives:

$$\tilde{y}_h = \mathbf{W}x_h = \mathbf{Y}\mathbf{X}^T\left[\left(\mathbf{X}\mathbf{X}^T\right)^+ x_h\right] \tag{4.2.6}$$

where the term inside the brackets is the n-dimensional preprocessed input vector. Since $X^T(XX^T)^+X = \mathbf{I}$, it can be concluded that the operator $(XX^T)^+$ maps the hth training vector x_h which is linearly independent from the remaining m - 1 training vectors x_k into a vector x'_h which is orthogonal to the x_k vectors and has an inner product of unity with the x_h vector. On the other hand, if one inputs a noisy version of one of the training key vectors, say x_h, then the preprocessed output x'_h will be rotated more in the direction of x_h and at the same time made more orthogonal to the remaining training vectors.

When the dimensions m and/or n are large, the direct method for solving for the generalized-inverse in equations (4.2.3) and (4.2.5) becomes impractical from a computational point of view. Furthermore, in many practical applications, the nature of the training key vectors is such that the matrix XX^T (or X^TX for the under-determined case) may be ill-conditioned, leading to numerical instabilities when computing the inverse. Therefore, it is desirable to replace the direct computation of the generalized-inverse with a more practical-to-compute stable method. This can be achieved by using gradient descent on J(**W**) in equation (4.2.2) and iteratively solving equation (4.0.2). Here, the weight matrix is incremented (starting from a zero-valued **W** matrix) according to the following equation:

$$\Delta \mathbf{W} = \mathbf{W}^{new} - \mathbf{W}^{c} = -\frac{1}{2}\rho \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}\Big|_{\mathbf{W}^c} = \rho\left[\mathbf{Y} - \mathbf{W}^c\mathbf{X}\right]\mathbf{X}^T \tag{4.2.7}$$

where the value of ρ for practical problems should be in the range $0 < \rho << 1$. We will refer to this algorithm as the *batch-mode adaptive* generalized-inverse training technique. Convergence can be speeded by initializing $\mathbf{W}^c$ as the correlation matrix $\mathbf{YX}^T$ since it can be shown [15,43] that this correlation term is the lowest-order term in a von Neumann expansion of the matrix $\mathbf{YX}^+$ given by:

$$\mathbf{Y}\mathbf{X}^+ = \alpha\mathbf{Y}\sum_{i=0}^{\infty}\left[\mathbf{I} - \alpha\,\mathbf{X}^T\mathbf{X}\right]^i\mathbf{X}^T \tag{4 2.8}$$

Equation (4.2.7) may also be modified further into *continuous-* or *local-mode* which allows for adaptive updating of the interconnection weights every time a new association $\{\mathbf{x}_k,\mathbf{y}_k\}$ is presented The following are two versions of this type of *continuous* adaptive generalized-inverse training:

$$\mathbf{W}^{new} = \mathbf{W}^{c} - \rho\left[\mathbf{y}_k - \mathbf{W}^c\mathbf{x}_k\right]\mathbf{x}_k^T \tag{4.2.9}$$

and

$$\mathbf{w}_i^{new} = \mathbf{w}_i^{c} - \rho\left[y_i^k - \mathbf{w}_i^c\mathbf{x}_k\right]\mathbf{x}_k^T \tag{4.2.10}$$

where $\mathbf{w}_i$ is the ith row of matrix $\mathbf{W}$ representing the weight vector of the ith perceptron ($i = 1, 2, ..., L$) and y_i^k is the ith bit of the kth association vector $\mathbf{y}_k$. In these equations, k is incremented after each iteration and the whole training set is cycled through multiple times until convergence is achieved. Equation (4.2.10) is used to synthesize the L memory perceptrons separately, and is known in the literature as the μ-LMS or Widrow-Hoff learning rule [44,45] (in fact, the LMS rule differs slightly from equation (4.2.10) in that it employs an additional perceptron bias bit which results in an extra weight that could be used in adjusting the threshold of the hard-clipping nonlinearity in a DAM). The choice of ρ is critical in determining stability and convergence time of the LMS algorithm. Choosing a large ρ speeds convergence, but can lead to instability. Horowitz and Senne [46] prove that the choice $0 < \rho < 1/[6\ \mathrm{tr}(\mathbf{X}^T\mathbf{X})]$ guarantees the stability of equation (4.2.10), for $\mathbf{x}_k$ patterns generated by a zero-mean Gaussian process independent over time.

A closely related recording technique to the under-determined generalized-inverse technique of equation (4.2.5), for the recording of autoassociative DAMs, is the *spectral technique* proposed by Personnaz et al. [47] and Venkatesh and Psaltis [48,49]. Here, the weight matrix is defined as follows:

$$\mathbf{W} = \mathbf{X}\,\mathbf{D}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T = \mathbf{Y}\mathbf{D}\mathbf{X}^+ \tag{4.2.11}$$

where $\mathbf{D} = \mathrm{diag}[\lambda_1, \lambda_2, ..., \lambda_k, ..., \lambda_m]$ is the m x m diagonal matrix of positive eigenvalues ($\lambda_k > 0$). Note that $\mathbf{W}$ is well defined if the inverse of $\mathbf{X}^T\mathbf{X}$ exists; i.e., if the $\mathbf{x}_k$ vectors are linearly independent. Furthermore, $\mathbf{W}$ is symmetric if $\lambda_k = \lambda$ for $k = 1, 2, ..., m$. Multiplying equation (4.2.11) by $\mathbf{X}$ gives

$$\mathbf{W}\mathbf{X} = \mathbf{X}\mathbf{D} \quad \text{or} \quad \mathbf{W}\mathbf{x}_k = \lambda_k \mathbf{x}_k, \quad k = 1, 2, \ldots, m \tag{4.2.12}$$

Assuming arbitrary positive λ_k, the above equation is a weighted minimum-norm solution where a fundamental memory (eigenvector) $\mathbf{x}_k$ having a large eigenvalue λ_k tends to have an enlarged basin of attraction compared to other memories with corresponding smaller eigenvalues; that is, more attention is shifted towards $\mathbf{x}_k$.

4.3 Ho-Kashyap Recording

Higher DAM performance can be accomplished if the perceptron activation functions are taken into account during the recording phase. An optimal recording technique employing the above feature has been proposed by Hassoun and Youssef [26,27] and Hassoun [28] for autoassociative and heteroassociative DAMs, respectively. This technique is based on the Ho-Kashyap algorithm [50] for the optimal MSE solution of a set of linear inequalities. One major difference between the Ho-Kashyap recording rule and the earlier recording techniques is that the weight vector and the activation function threshold are independently optimized for each neuron.

According to the Ho-Kashyap algorithm, the ith row of the weight matrix $\mathbf{W}$ and its corresponding threshold T_i are formulated as the weight vector $\mathbf{w}_i = [w_{i0}\ w_{i1}\ w_{i2}\ \ldots\ w_{in}]^T$, where $T_i = -\ w_{i0}$. Then, and upon the presentation of the kth association pair, the ith perceptron can be trained to classify a given training set $\{\mathbf{x}_k, \mathbf{y}_k\}$ correctly by computing the (n+1)-dimensional weight vector $\mathbf{w}_i$ satisfying the following set of m inequalities:

$$\mathbf{x}_k^T \mathbf{w} \begin{cases} > 0 & \text{if } y_i^k = 1 \\ < 0 & \text{if } y_i^k = 0 \end{cases} ; \quad \text{for } k = 1, 2, \ldots, m \tag{4.3.1}$$

where the vector $\mathbf{x}_k$ is derived from the original $\mathbf{x}_k$ by augmenting it with a bias of "1" as its x_0 component. Next, if we define a set of m new vectors $\mathbf{z}_k$ according to

$$\mathbf{z}_k = \begin{cases} +\mathbf{x}_k & \text{if } y_i^k = 1 \\ -\mathbf{x}_k & \text{if } y_i^k = 0 \end{cases} ; \quad \text{for } k = 1, 2, \ldots, m \tag{4.3.2}$$

and let

$$\mathbf{Z} = \left[\mathbf{z}_1\ \mathbf{z}_2 \ldots \mathbf{z}_m\right] \tag{4.3.3}$$

then equation (4.3.1) may be rewritten as (the subscript "i" is dropped in order to simplify notation)

$$\mathbf{Z}^T \mathbf{w} > \mathbf{0} \tag{4.3.4}$$

Now if we define an m-dimensional positive valued margin vector $\mathbf{b}$ ($\mathbf{b} > \mathbf{0}$) and use it in equation (4.3.4), we arrive at the following equivalent form of equation (4.3.1)

$$\mathbf{Z}^T \mathbf{w} = \mathbf{b} > \mathbf{0} \tag{4.3.5}$$

Thus, the training of the perceptron is now equivalent to solving equation (4.3.5) for **w**, subject to the constraint **b** > **0**. Ho and Kashyap have proposed an iterative algorithm for solving equation (4.3.5). In the Ho-Kashyap algorithm, the components of the margin vector are first initialized to small positive values and the Moore-Penrose pseudoinverse is used to generate an MSE solution for **w** (based on the initial guess for **b**) which minimizes the objective function $J(\mathbf{w},\mathbf{b}) = \|\mathbf{Z}^T\mathbf{w} - \mathbf{b}\|^2$

$$\mathbf{w} = \left(\mathbf{Z}^T\right)^+ \mathbf{b} \tag{4.3.6}$$

Next, a new estimate for the margin vector is computed by performing the constrained (**b** > **0**) descent

$$\mathbf{b}^{new} = \mathbf{b}^c + \rho\left[\varepsilon + |\varepsilon|\right]; \quad \text{with } \varepsilon = \mathbf{Z}^T\mathbf{w} - \mathbf{b} \tag{4.3.7}$$

where |.| denotes the absolute value of the components of the argument vector. A new estimate of **w** can now be computed using equation (4.3.6) and employing the updated margin vector from equation (4.3.7). This process is iterated until all the components of ε are zero (or are sufficiently small and positive), which is an indication of the linear separability of the training set, or until $\varepsilon < \mathbf{0}$, which in this case is an indication of the nonlinear separability of the training set (no solution is found). It can be shown [50,51] that the Ho-Kashyap procedure converges in a finite number of steps if the training set is linearly separable. A sufficient condition for convergence is $0 < \rho < 1$. We will refer to the above algorithm as the *direct* Ho-Kashyap (DHK) algorithm.

When the training set is under-determined (m < n+1), the Ho-Kashyap recording algorithm converges in one iteration [27]. That is, equation (4.3.6) leads to a perfect solution for **w** and no margin update is needed. This solution is identical to the LMS solution discussed in Section 4.2, if the initial margin vector was chosen to have equal positive components. Therefore, the full benefits of the Ho-Kashyap recording technique are achieved with over-determined training sets (m > n+1), which leads to optimized weights and margins. Section 5 discuses ways of extending originally under-determined training sets into over-determined ones which are well suited for harvesting the full benefits of the Ho-Kashyap recording technique.

The direct synthesis of the **w** estimate in equation (4.3.6) involves a one-time computation of the pseudoinverse of $\mathbf{Z}^T$. However, such direct computation can be time-consuming, and it requires special treatment when $\mathbf{Z}\mathbf{Z}^T$ (or $\mathbf{Z}^T\mathbf{Z}$, for the under-determined case m < n+1) is singular. An alternative algorithm that does not require the computation of $(\mathbf{Z}^T)^+$ can be derived based on gradient descent principles

Starting with the objective function $J(\mathbf{w},\mathbf{b}) = \|\mathbf{Z}^T\mathbf{w} - \mathbf{b}\|^2$, gradient descent may be performed [51] in **b** and **w** spaces so that J is minimized subject to the constraint **b** > **0**. The gradients of J with respect to **w** and **b** are given by equations (4.3.8) and (4.3.9), respectively

$$\nabla_{\mathbf{b}} J(\mathbf{w}, \mathbf{b})|_{\mathbf{w}^c, \mathbf{b}^c} = -2\left(\mathbf{Z}^T\mathbf{w}^c - \mathbf{b}^c\right); \text{ subject to } \mathbf{b} > \mathbf{0} \tag{4.3.8}$$

$$\nabla_{\mathbf{w}} J(\mathbf{w}, \mathbf{b})|_{\mathbf{w}^c, \mathbf{b}^{new}} = 2\mathbf{Z}\left(\mathbf{Z}^T\mathbf{w}^c - \mathbf{b}^{new}\right) \tag{4.3.9}$$

One analytic method for imposing the constraint **b** > **0** is to replace the gradient in (4.3.8) by

-(ε + $|\varepsilon|$) with the vector ε as defined in equation (4.3.7). This leads to the following gradient descent formulation of the Ho-Kashyap procedure:

$$\mathbf{b}^{new} = \mathbf{b}^{c} + \frac{\rho_1}{2}\left(\left|\varepsilon^{c}\right| + \varepsilon^{c}\right); \text{ with } \varepsilon^{c} = \mathbf{Z}^{T}\mathbf{w}^{c} - \mathbf{b}^{c} \tag{4.3.10}$$

$$\mathbf{w}^{new} = \mathbf{w}^{c} - \rho_2 \mathbf{Z}\left(\mathbf{Z}^{T}\mathbf{w}^{c} - \mathbf{b}^{new}\right) = \mathbf{w}^{c} + \frac{\rho_1\rho_2}{2}\mathbf{Z}\left[\left|\varepsilon^{c}\right| + \varepsilon^{c}\left(1 - \frac{2}{\rho_1}\right)\right] \tag{4.3.11}$$

with ρ_1 and ρ_2 belonging to] 0, 1]. We will refer to the above procedure in equations (4.3.10) and (4.3.11) as the *batch-mode* adaptive Ho-Kashyap (AHK) procedure, because of the requirement that all training vectors $\mathbf{z}_k$ (or $\mathbf{x}_k$) must be present and included in **Z**. A *continuous-mode* adaptive Ho-Kashyap procedure for solving equation (4.3.5) is arrived at by replacing the **Z** in the *batch-mode* procedure of equations (4.3.10) and (4.3.11) by $\mathbf{z}_k$, and thus arriving at the following continuous-mode update rule

$$b_k^{new} = b_k^{c} + \frac{\rho_1}{2}\left(\left|\varepsilon^{c}\right| + \varepsilon^{c}\right); \text{ with } \varepsilon^{c} = \mathbf{z}_k^{T}\mathbf{w}^{c} - b_k^{c} \tag{4.3.12}$$

$$\mathbf{w}^{new} = \mathbf{w}^{c} - \rho_2 \mathbf{z}_k\left[\mathbf{z}_k^{T}\mathbf{w}^{c} - b_k^{new}\right] = \mathbf{w}^{c} + \frac{\rho_1\rho_2}{2}\left[\left|\varepsilon^{c}\right| + \varepsilon^{c}\left(1 - \frac{2}{\rho_1}\right)\right]\mathbf{z}_k \tag{4.3.13}$$

where b_k represents a margin scaler associated with the $\mathbf{x}_k$ input. In all of the above Ho-Kashyap training strategies, the margin values and the perceptron weights are initialized to small positive and zero values, respectively. If full margin error correction is assumed in equations (4.3.12) and (4 1.13) (i.e, $\rho_1 = 1$), then the above AHK procedure reduces to the heuristically derived procedure reported earlier by Hassoun and Clark [53] and Hassoun [54]. An alternative way of writing equations (4.3.12) and (4.3.13) is [52]

$$\Delta b_k = \rho_1 \varepsilon^{c} \text{ and } \Delta \mathbf{w} = \rho_2\left(\rho_1 - 1\right)\varepsilon^{c}\mathbf{z}_k \quad \text{if } \varepsilon^{c} > 0 \tag{4.3.14}$$

$$\Delta b_k = 0 \text{ and } \Delta \mathbf{w} = -\rho_2 \varepsilon^{c}\mathbf{z}_k \quad \text{if } \varepsilon^{c} \le 0 \tag{4.3.15}$$

We will refer to this procedure as the AHK I learning rule.

The constraint $b_k > 0$ in equations (4.3.12) and (4.3.14) was realized by starting with a positive initial margin and restricting the change in Δb to positive real values. An alternative way to realize this constraint is to allow both positive and negative changes in Δb, except for cases where a decrease in b_k results in a negative margin. This modification results in the following alternative AHK II learning rule:

$$\Delta b_k = \rho_1 \varepsilon^{c} \text{ and } \Delta \mathbf{w} = \rho_2\left(\rho_1 - 1\right)\varepsilon^{c}\mathbf{z}_k \quad \text{if } b_k^{c} + \rho_1\varepsilon^{c} > 0 \tag{4.3.16}$$

$$\Delta b_k = 0 \text{ and } \Delta \mathbf{w} = -\rho_2 \varepsilon^{c}\mathbf{z}_k \quad \text{if } b_k^{c} + \rho_1\varepsilon^{c} \le 0 \tag{4.3.17}$$

A further modification of the above equation results in the AHK III rule [52], which is capable of fast convergence to approximate solutions in the case of nonlinearly separable mappings. However, this is beyond the scope of this chapter and is not considered further.

5. RECORDING STRATEGIES

The encoding, dimension, and number of stored patterns highly affects the performance of a DAM. The recording strategy of a given number of associations is also of critical importance in the robustness and dynamics of a DAM. For example, DAM performance can be enhanced by augmenting the training set with an additional set of specialized associations. Here, we present three examples of recording strategies: (1) training with perfect associations, (2) training with an extended set of noisy/partial associations, and (3) training with the aid of a specialized set of associations.

The first training strategy is employed when only perfect associations are present. The training set consists of the m input/target pairs $\{x_k, y_k\}$ and is shown in Figure 6(a). This represents the simplest training strategy possible, and relies on intrinsic DAM dynamics to realize the needed basins of attraction around each recorded association. This strategy works if the number of associations is relatively small compared to the smallest of the dimensions n and L, and if the degree of correlation between the m associations is relatively low. With this strategy, the training set is usually under-determined ($m < n$).

The second recording strategy is employed when the training set consists of a number of clusters with each cluster having a unique label or target. This strategy is common in pattern classification applications and is useful in defining large basins of attraction around each recorded memory, thus increasing DAM error tolerance. Figure 6(b) illustrates this case. In general, the inclusion of the noisy/partial associations increases the size of the training set and leads to an over-determined set of associations ($m > n$).

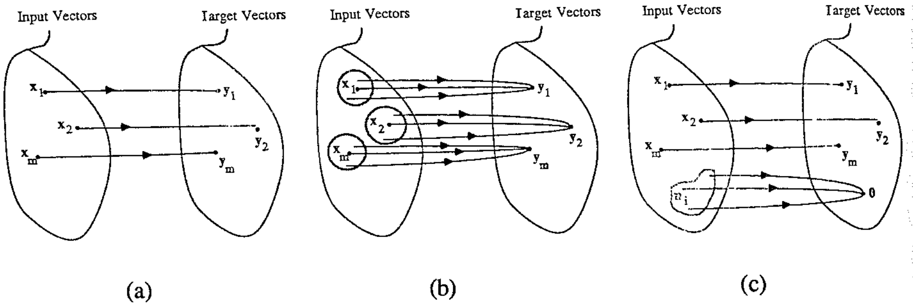


Figure 6. Examples of DAM recording strategies. (a) Simple recording, (b) training with noisy/partial associations, and (c) training with the aid of specialized associations.

The third recording strategy is employed when specific additional DAM associations must be introduced or eliminated. One possibility of employing this strategy is when a *ground state* must be introduced to act as a *no-decision* state, as depicted in Figure 6(c). For example, for associations encoded such that sparse vectors $\mathbf{n}_i$ have low information content, augmenting the original training set with associations of the form $\{\mathbf{n}_i,\mathbf{0}\}$ leads to the creation of a no-decision state $\mathbf{0}$ which attracts highly corrupted or noisy inputs and prevents them from being classified erroneously or mapped into spurious memories. Simulations illustrating the use and benefits of the above first and third recording strategies are presented in the next section, employing various recording algorithms.

6. DAM CAPACITY AND PERFORMANCE

In the following, the capacity and performance of autoassociative DAMs are discussed. Capacity is defined as a measure of the ability of a DAM to store a set of *unbiased random* binary patterns (probability that a 1/0 or +1/-1 bit equals to .5) $\{\mathbf{x}_k\}$ at given error correction and recall accuracy levels. Earlier proposed capacity measures [16,19-25,41,47,49] have defined capacity as equivalent to a tight upper bound on the pattern ratio ($r = m/n$) for which all stored memories are fixed states, with a probability approaching one. Measuring capacity as the ability to correct key pattern errors, with guaranteed convergence to the closest stored memory, has also been recently advanced.

The two most-used DAM capacity measures in the literature are the absolute capacity (C_a) and relative capacity (C_r) measures. C_a is defined as the least upper bound on the pattern ratio (m/n) such that the m stored memories are retrievable in one pass through the DAM; i.e., the DAM is capable of memorizing m patterns as equilibria points. C_r is a least upper bound on m/n such that the stored memories are approximately retrievable. For correlation-recorded autoassociative DAMs employing hard-clipping activation, the relative capacity is approximately equal to .15, for a retrieval error (between an exact input and one-pass noisy output) < 10 percent [11,14,16,20]. The requirement for error-free memory recall severely limits the capacity of correlation-recorded DAMs to $C_a = 1/[4\log(n)]$ [21-25,41], which approaches zero for large n (log is the natural logarithm).

Another, more useful, DAM capacity measure gives an upper bound on m/n in terms of attraction radius and memory size [21,41]. According to this measure, a correlation-recorded DAM has a maximum pattern ratio $m/n = (1 - 2\rho)^2/[4 \log(n)]$ which guarantees error-free fundamental memory retrieval, in the limit of large n, from key patterns lying inside the Hamming sphere of radius ρn ($\rho < .5$). This capacity result is only approximate for the limiting case $\rho = 0$, and it has been shown [24] that, when the number of stored patterns approaches $n/[4 \log(n)]$, the fundamental memories have a basin of attraction of normalized radius $\rho = 0.024$, and convergence to such memories is achieved in $O(\log \log n)$ parallel iterations. Recently, Amari [22,31] applied a statistical neurodynamics approach in the analysis of correlation-recorded DAM unbiased random error correction capability. The plot in Figure 7 depicts the single-pass error correction curves predicted for a parallel updated DAM in the limit of large n, where ρ_{in} and ρ_{out} are the probabilities of input and output noise, respectively, and r is the pattern ratio. Note how the ability of the DAM to retrieve fundamental memories from noisy inputs is reduced as r approaches the relative capacity of 0.15. These results cannot be applied to describe the second and higher passes through the DAM, since the first pass output is correlated with the $\mathbf{W}$ matrix (and hence the stored $\mathbf{x}_k$) and its noise content is not random anymore. However, Amari [22] has extended this theory to predict the complete retrieval behavior from noisy inputs.

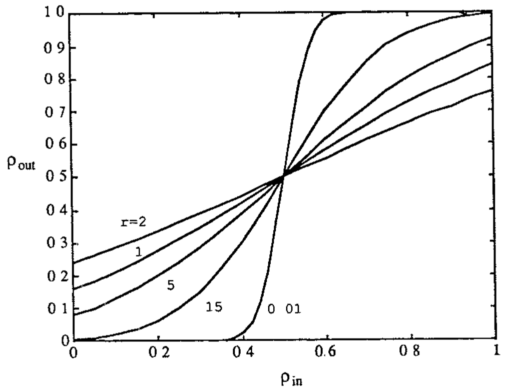


Figure 7. Single-pass error correction in a correlation-recorded DAM as a function of pattern ratio r = m/n.

Amari also extends the above capacity measures to the case of a correlation-recorded DAM with sparsely encoded memories [29], where he shows that C_a is of order $1/\log^2(n)$ which is much larger than that of non-sparsely encoded memory. In another DAM variation, Yanai and Sawada [30] derive the expression $C_a = (1+h)^2/[2 \log(n)]$ for a correlation-recorded DAM with unbiased random memories and with perceptrons employing a hysteric hard-clipping activation function with a hysteric range [-h,+h]. Other interesting extensions are reported by Marcus et al. [55], who have studied the retrieval dynamics of parallel updated continuous DAMs employing correlation and generalized-inverse recording of unbiased random bipolar memories with large n. Phase diagrams depicting various DAM dynamics (recall, oscillatory, spin glass, and ground regions) in the space of pattern ration r and activation function gain β (a $\tanh(\beta z)$ activation function was assumed) were derived for the correlation and generalized-inverse DAMs; for instance, it was shown that period-two limit cycles are eliminated from the DAM for both recording techniques when $\beta < 1/r = n/m$, and that the origin is the only attractor of the correlation DAM when $\beta < 1/[1+2 (r)^{-1/2}]$ and for the generalized-inverse DAM when $\beta < 1/(1 - r)$.

Let us turn our attention back to parallel updated autoassociative DAM employing hard-clipping activation and unbiased random high-dimensional binary memories. It can be shown [56] that for an under-determined training set of unbiased random binary vectors, and in the limit as n approaches infinity, the probability of linear independence of the training vectors approaches one. This makes the single-layer DAM appropriate for the realization of such training sets Little theoretical work has been done on the capacity and performance of these DAMs recorded with generalized-inverse or Ho-Kashyap techniques. Youssef and Hassoun [57] report Monte Carlo simulations for $16 < n < 128$ where the retrieval performance and capacity of various recording/learning techniques, including the generalized-inverse and Ho-Kashyap, were investigated. They propose a capacity/performance measure similar to C_a given above, with one additional DAM performance parameter: recall accuracy (RA). Here, capacity is computed under the strict requirement that all fundamental memories are perfectly retrievable and that *retrieval is not* *restricted to a single pass.* Figure 8(a) shows plots for ρ vs. m/n where ρ is an upper bound on the normalized basin radius around fundamental memories, guaranteeing a RA > 99 percent. Three curves are shown for the correlation (Hopfield), generalized-inverse (GI), and Ho-Kashyap (HK) DAMs, respectively, with n = 64. A fourth curve is shown (dotted line) for the Ho-Kashyap and generalized-inverse DAM, with n = 128. These curves clearly depict the superiority of the GI and HK recording techniques over the correlation technique. It can also be concluded that the GI- and HK-recorded DAMs have a relatively large operating region where error correction is possible. This range is defined, roughly, as 0 < m/n < .5. From the simulations in [57] it is to be noted that the synthesized weight matrix becomes a diagonal-dominant matrix in the limit as m approaches n/2 at which point all fundamental memories lose their basin of attraction. At m = .15n with large n, the GI and HK algorithms lead to DAMs capable of correcting in excess of 25 percent unbiased random noise. On the other hand, and at a loading level of m = .15n, the correlation DAM is not capable of retaining the exact fundamental memories, nor their basin of attraction. Figure 8(b) extends the results just discussed for cases of reduced recall accuracy constraints (RA > 95 percent, 85 percent) for the GI-recorded DAM. The Ho-Kashyap-recorded DAM exhibits similar characteristics.

It is interesting to note the similarity between the GI and HK DAM retrieval performance. This should not be surprising, since the training set assumed above is under-determined where, according to the discussion in Section 4.3, the HK solution is equivalent to the GI solution except for the added bias bits. The effects of the bias bits disappear when high-dimensional unbiased-random memories (associations) are used. However, when non-random memories are used, the GI and HK DAMs exhibit substantially different dynamics, as is shown next.

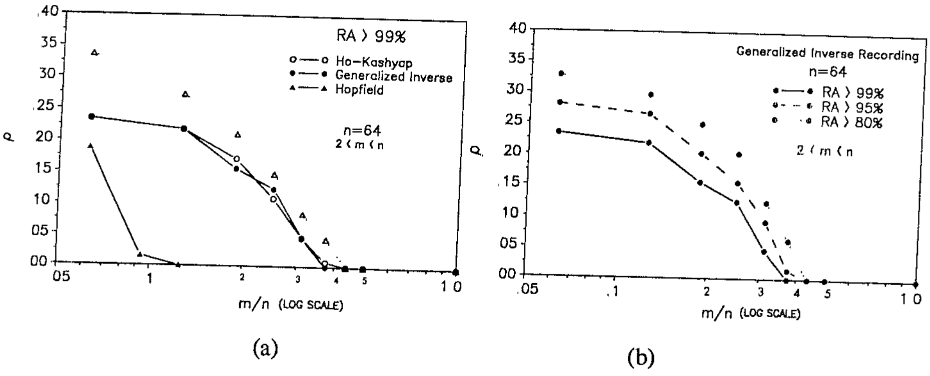
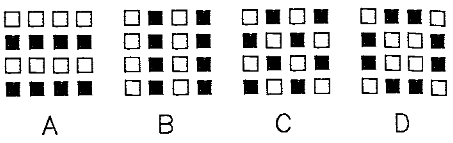


Figure 8. (a) Capacity/performance curves comparing the correlation, generalized-inverse, and Ho-Kashyap DAMs for n = 64. Dashed curve represent the cases of GI and HK DAM with n = 128. (b) Generalized-inverse performance curves for various values of recall accuracy (RA).

We conclude this chapter by presenting a limited but illustrative simulation comparing the various *direct* and *adaptive* recording techniques discussed in Section 4. We also illustrate the advantages of employing specialized recording strategies in improving the retrieval characteristics of a DAM. Four 16-dimensional binary patterns are chosen such that any two distinct patterns have a Hamming distance of 8, as shown in Figure 9(a,b).

$A = x_1 : 1111000011110000$
$B = x_2 : 1010101010101010$
$C = x_3 : 1010010110100101$
$D = x_4 : 1001011001101001$

(a)



(b)

Figure 9. (a) Four memory vectors used to train an autoassociative DAM.
(b) 2-D representation of the vectors in (a).

Two sets of simulations were performed, and the results are tabulated in Tables 1 and 2, respectively. The first simulation assumes a direct recording strategy employing the four memories ($m = 4 < n = 16$). The second set of simulations employs an over-determined training set consisting of the four memories of Figure 9 and sixteen additional heteroassociations representing the mapping of the rows of the 16-dimensional unit matrix into the 16-dimensional zero pattern. In both cases, the DHK, GI, correlation (HOP), LMS, AHK I, and AHK II recording/learning algorithms were employed in DAM synthesis.

	Recording/Learning Algorithm⁺					
Retreival Dynamics*	DHK	GI	HOP#	LMS#	AHKI	AHKII
Convergence rate to fundamental memories (%)	54 5	28 3	18 5	34 7	61 7	53 1
Convergence rate to spurious memories (%)	45 5	65 7	23 0	65 3	38 3	46 9
Number of spurious memories detected	22	51	10	79	18	16
Convergence rate to ground memories (%)	0	6 0	0 0	0 0	0 0	0 0
Convergence rate to oscillatory memories (%)	0	0	58 5	0	0	0

* DAM tested with 1000 input vectors having uniformly random binary bits
+ Training was performed with the original set of 4 memories
# Training and testing were performed assuming bipolar memories

	Recording/Learning Algorithm⁺					
Retreival Dynamics*	DHK	GI	HOP#	LMS#	AHKI	AHKII
Convergence rate to fundamental memories (%)	34 65	24 7	-----	12 7	24 8	1 1
Convergence rate to spurious memories (%)	10 2	60 4	-----	83 6	70 9	62 8
Number of spurious memories detected	7	44	---	127	38	16
Convergence rate to ground memories (%)	55 1	14 9	----	3 7	4 3	36 1
Convergence rate to oscillatory memories (%)	0 05	0	-----	0	0	0

* DAM tested with 1000 input vectors having uniformly random binary bits.
+ Training was performed employing sparse memory-to-ground mapping
# Training and testing were performed assuming bipolar memories

Table 1. Retrieval dynamics of various DAMs trained with the memories of Figure 9.

Table 2. Retrieval dynamics of DAMs with specialized associations.

The DAMs in these simulations were tested with 10,000 unbiased pseudo random vectors. Unipolar encoding was assumed for the Ho-Kashyap and GI DAMs. On the other hand, bipolar encoding was assumed for the Hopfield and LMS DAMs, for both training and retrieval phases (the LMS DAM also employed bias). In addition, all adaptive learning algorithms (LMS, AHK I, and AHK II) were initialized with zero weight matrices. The learning rates were 0.1 for LMS, 0.1 for direct HK and $\rho_1 = 1$ and $\rho_2 = 0.01$ for AHK I (with margin vector initialized to all "+1" components), and $\rho_1 = 0.1$ and $\rho_2 = 0.05$ for AHK II (with initial margin vectors of "+.1" components). Learning stopped after fifty iterations for DHK and AHK I. For LMS and AHK II, learning stopped after reducing the error function $J(\mathbf{W})$ to 0.0001. All weight matrices were normalized and rounded to integer weights in the range [-99, +99].

The three Ho-Kashyap recording algorithms have comparable performances which exceed those of generalized-inverse, Hopfield, and LMS. The first row depicts the formation of large basins of attraction around fundamental memories for the HK algorithms. Noisy inputs which do not converge to fundamental memories are attracted by spurious memories for HK recording. The GI and LMS DAMs resulted in a relatively large number of spurious memories which attracted about 65 percent of the test vectors. The GI DAM was the only DAM with a no-decision state (ground state), which attracted 6 percent of the input. As expected, the worst performance is that of the Hopfield DAM, which has a low convergence rate to fundamental memories and a high convergence rate to period-two oscillations (these oscillations can be eliminated if stochastic update is employed; however, this will result in an increased convergence rate to spurious states). It is also interesting to note that even though the $\mathbf{W}$ matrix of the HK-recorded DAM is not symmetric and that parallel updating is used, no oscillations were encountered. This phenomenon is due in part to the weight normalization and rounding employed, and to the low memory loading level ($m = 4$) [57]. The use of sparse memory-to-ground mapping enhances the performance of the DHK DAM, as depicted in the first column of Table 2. Here, the number of spurious states is reduced to seven and the convergence of highly noisy inputs is directed from spurious and fundamental memories to a ground state which attracts about half of the test inputs. In addition, only five noisy inputs (0.05 percent) converged to period-two oscillations. The GI also shows slight improvement in performance compared to the under-determined recording case in Table 1. On the other hand, this recording strategy does not seem to be adequate for the Hopfield DAM (fundamental memories were not retrievable) or for the adaptive LMS and AHK recording techniques.

Heteroassociative DAM performance and dynamics are less understood. Kosko [8,10] has analyzed the stability and capacity of correlation-recorded heteroassociative DAM (or bidirectional associative memory (BAM)) and its extension to general Hebbian learning. Hassoun [28] employs simulations in the analysis of the capacity and performance of parallel updated GI- and HK-recorded heteroassociative DAMs employing hard-clipping activations.

The dynamics and capacity of continuous or discrete updated DAMs employing multi-layer architectures have not received adequate attention in the past. Multi-layer DAMs are important, since single-layer DAMs are limited to the realization of linearly-separable mappings; in practice, many interesting problems are nonlinearly-separable. The training of DAMs also needs to be developed further, in such a way that the dynamical nature of the architecture is taken into account during the learning/recording synthesis phase, which can result in more predictable retrieval dynamics. Also, computationally efficient methods of controlling the shape and size of the basins of attraction of fundamental and spurious memories are desirable (see [58] for a recent attempt to address this problem).

REFERENCES

[1] Amari, S-I, Learning Patterns and Pattern Sequences by Self-Organizing Nets of Threshold Elements. *IEEE Trans. Computers*, C-21, pp. 1197-1206, 1972.

[2] Anderson, J. A., A Simple Neural Network Generating Interactive Memory. *Mathematical Biosciences*, 14, pp. 197-220, 1972.

[3] Nakano, K., Associatron: A Model of Associative Memory. *IEEE Trans. Sys. Man Cybern.*, SMC-2, pp. 380-388, 1972.

[4] Kohonen, T., Correlation Matrix Memories. *IEEE Trans. Computers*, C-21, pp. 353-359, 1972.

[5] Kohonen, T. and Ruohonen, M., Representation of Associated Data by Matrix Operators. *IEEE Trans. Computers*, C-22, pp. 701-702, 1973.

[6] Kohonen, T., An Adaptive Associative Memory Principle. *IEEE Trans. Computers*, C-23, pp. 444-445, 1974.

[7] Hopfield, J. J., Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proc. National Academy Sciences*, USA, 79, pp. 2445-2558, 1982.

[8] Kosko, B., Adaptive Bidirectional Associative Memories. *Applied Optics*, 26, pp. 4947-4960, 1987.

[9] Okajima, K., Tanaka, S., and Fujiwara, S., A Heteroassociative Memory Network with Feedback Connection. M. Caudill & C. Butler (Eds.), *Proc. IEEE First International Conference on Neural Networks*, San Diego, CA, pp. II: 711-718, 1987.

[10] Kosko, B., Bidirectional Associative memories. *IEEE Trans. Sys. Man Cybern.*, SMC-18, pp. 49-60, 1988.

[11] Amari, S., Neural Theory of Association and Concept-Formation. *Biological Cybernetics*, 26, pp. 175-185, 1977.

[12] Uesaka, G. and Ozeki, K., Some Properties of Associative Type Memories. *Journal of the Institute of Electrical and Communication Engineers of Japan*, 55-D, pp. 323-220, 1972.

[13] Wigstrom, H., A Neuron Model with Learning Capability and its Relation to Mechanism of Association. *Kybernetic*, 12, pp. 204-215, 1973.

[14] Hopfield, J. J., Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proc. National Academy Sciences*, USA, 79, pp 2554-2558, 1982.

[15] Kohonen, T. Self-Organization and Associative Memory (Springer-Verlag, New York, 1984).

[16] Amit, D. J., Gutfreund, and Sompolinsky, H., Storing Infinite Numbers of Patterns in a Spin-Glass Model of Neural Networks. *Physical Review Lett* , 55(14), pp. 1530-1533, 1985.

[17] Abu-Mustafa, Y. S. and St. Jacques, J. M., Information Capacity of the Hopfield Model. *IEEE Trans. Info. Theory*, IT-31, pp. 461-464, 1985.

[18] Montgomery, B. L. and Kumar, V. K., Evaluation of the use of the Hopfield Neural Network Model as a Nearest-Neighbor Algorithm. *Applied Optics*, 25(20), pp. 3759-3766, 1986.

[19] Stiles, G. S. and Denq, D-L., A Quantitative Comparison of Three Discrete Distributed Associative Memory Models. *IEEE Trans. Computers*, C-36, pp. 257-263, 1987.

[20] Meir, R. and Domany, E., Exact Solutions of a Layered Neural Network Memory. *Physical Review Letters*, 59, pp. 359-362, 1987.

[21] McEliece, R. J., Posner, E. C., Rodemich, E R., and Venkatesh, S. S., The Capacity of the Hopfield Associative Memory. *IEEE Trans Info. Theory*, IT-33, pp. 461-482, 1987.

[22] Amari, S-I, and Maginu, K., Statistical Neurodynamics of Associative Memory. *Neural Networks*, 1, pp. 63-73, 1988

[23] Newman, C., Memory Capacity in Neural Network Models: Rigorous Lower Bounds. *Neural Networks*, 3, pp. 223-239, 1988.

[24] Komlos, J. and Paturi, R., Convergence Results in an Associative Memory Model. *Neural Networks*, 3, pp. 239-250, 1988.

[25] Dembo, A., On the Capacity of the Associative Memories with Linear Threshold Functions. *IEEE Trans. Info. Theory*, IT-35, pp. 709-720, 1989.

[26] Hassoun, M. H., and Youssef, A. M., New Recording Algorithm for Hopfield Model Associative Memories In Neural Network Models for Optical Computing, R. Athale & J. Davis (Eds.), *Proc. SPIE*, 882, pp. 62-70, 1988.

[27] Hassoun, M. H., and Youssef, A. M., A High-Performance Recording Algorithm for Hopfield Model Associative Memories. *Optical Engineering*, 27, pp. 46-54, 1989.

[28] Hassoun, M. H., Dynamic Heteroassociative Neural Memories. *Neural Networks*, 2, pp. 275-287, 1989.

[29] Amari S-I, Characteristics of Sparcely Encoded Associative Memory. *Neural Networks*, 2, pp. 451-457, 1989.

[30] Yanai, H. and Sawada, Y., Associative Memory Network Composed of Neurons with Hysteric Property. *Neural Networks*, 3, pp. 223-228, 1990.

[31] Amari, S-I, Mathematical Foundations of Neurocomputing. *Proc. IEEE*, 78(9), pp. 1443- 1463, 1990.

[32] Fogelman, S. F., Contributions a' une Théorie du Calcul sur Réseaux. Thése d'Etat, Grenoble, 1985.

[33] Golden, R. M , The "Brain-State-in-a-Box" Neural Model is a Gradient Descent Algorithm. *Journal of Mathematical Psychology*, 30, pp. 73-80, 1986.

[34] Greenberg, H. J., Equilibria of the Brain-State-in-a-Box (BSB) Neural Model. *Neural Networks*, 1, pp. 323-324, 1988.

[35] Anderson, J. A., Silverstien, J. W., and Ritz, S. A. and Jones, R. S., Distinctive Features, Categorical Perception, and Probability Learning: Some Applications of Neural Model. *Psychological Review*, 84, pp. 413-451, 1977.

[36] Hopfield, J. J., Neurons with Graded Response have Collective Computational Properties like Those of Two-State Neurons. *Proc. National Academy Sciences*, USA, 81, pp. 3088-3092, 1984.

[37] Marcus, C M. and Westervelt, R. M., Dynamics of Iterated-Map Neural Networks. *Physical Review A*, 40(1), pp. 501-504, 1989.

[38] Hassoun, M. H., Two-Level Neural Network for Deterministic Logic Processing. In optical Computing and Nonlinear Materials, N. Peyghambarian (Ed.), *Proc SPIE*, 881, pp. 258-264, 1988.

[39] Rumelhart, D. E., McClelland, J. L., and the PDP Research Group, Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1 (MIT Press, Cambridge 1986).

[40] Psaltis, D. and Park, C. H., Nonlinear Discriminant Functions and Associative Memories. In Neural Networks for Computing, J. S. Denker (Ed.), *Proc. American Inst. Physics*, 151, pp. 370-375, 1986

[41] Weisbuch, G. and Fogelman S., Scaling Laws for the Attractors of Hopfield Networks. *Journal De Physique Lett* , 46(14), pp. L-623-L-630, 1985.

[42] Penrose, R., A Generalized Inverse for Matrices. *Proc Cambridge Philosophical Society*, 51, pp. 406-413, 1955.

[43] Rao, C. R. and Mitra, S. K., Generalized Inverse of Matrices and its Applications (Wiley, New York 1971).

[44] Widrow, B. and Hoff, M. E. Jr., Adaptive Switching Circuits. *IRE WESCON Convention Record*, pp 96-104, 1960.

[45] Widrow, B. and Lehr, M. A., 30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation. *Proc. IEEE*, 78(9), pp. 1415-1442, 1990.
[46] Horowitz, L. L, and Senne, K. D., Performance Advantage of Complex LMS for Controlling Narrow Band Adaptive Arrays. *IEEE Trans. Circuits Systems,* CAS-28, pp. 562-576, 1981.
[47] Personnaz, L. Guyon, I., and Dreyfus, G., Information Storage and Retrieval in Spin-Glass like Neural Networks. *Journal Physique Lett.*, 46, pp L359-L365, 1985.
[48] Venkatesh, S. S. and Psaltis , D., Efficient Strategies for Information Storage and Retrieval in Associative Neural Nets. *Workshop on Neural Networks for Computing*, Santa Barbara, CA, 1985.
[49] Venkatesh, S. S., and Psaltis, D., Linear and Logarithmic Capacities in Associative Neural Networks. *IEEE Trans. Info. Theory*, IT-35, pp. 558-568, 1989.
[50] Ho, Y. and Kashyap, R. L., An Algorithm for Linear Inequalities and its Applications. *IEEE Trans. Elec. Computers*, EC-14, pp. 683-688, 1965.
[51] Slansky, J. and Wassel, G. N., Pattern Classification and Trainable Machines (Springer-Verlag, New York 1981).
[52] Hassoun, M. H. and Song, J., Adaptive Ho-Kashyap Rules for Optimal Training of Perceptrons. To be submitted to *IEEE Trans. Neural Networks.*
[53] Hassoun, M. H. and Clark, D. W., An Adaptive Attentive Learning Algorithm for Single Layer Neural Networks. *Proc. IEEE Annual Conference on Neural Networks*, pp. 431-440, San Diego, CA 1988.
[54] Hassoun, M. H., Adaptive Dynamic Heteroassociative Neural Memories for Pattern Classification. In Optical Pattern Recognition, H-K Liu (Ed.), *Proc. SPIE,* 1053, pp. 75-83, 1989.
[55] Marcus, C. M., Waugh, F R., and Westervelt, R. M., Associative Memory in an Analog Iterated-Map Neural Network. Physical Review A, 41(6), pp. 3355-3364, 1989.
[56] Komlos, J., On the Determinant of (0,1) Matricies. *Studia Scientarum Mathematicarum Hungarica*, 2, pp. 7-21, 1967.
[57] Youssef, A. M. and Hassoun M. H., Dynamic Autoassociative Neural Memory Performance vs. Capacity. In Optical Pattern Recognition, H-K Liu (ed.), *Proc. SPIE,* 1053, pp.52-59, 1989.
[58] Venkatesh, S. S., Pancha, G., Psaltis, D., and Sirat G., Shaping Attraction Basins in Neural Networks. *Neural Networks*, 3(6), pp 613-623, 1990.

Artificial Neural Networks and Statistical Pattern Recognition
Old and New Connections
I.K. Sethi and A.K. Jain (Editors)

Optical Associative Memories

B. V. K. Vijaya Kumar and Phillip K. Wong

Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Four associative memory schemes (Hopfield associative memory, generalized inverse associative memory, spectral associative memory and Ho-Kashyap associative memory) that are suitable for optical processor implementation are described. Numerical results are presented that compare the capabilities of these 4 associative memories.

1 INTRODUCTION

Artificial Neural Networks (ANNs) have captured the imagination of many researchers because of their apparent ability to solve complex problems for which no algorithmic solutions appear to be feasible. However, a realistic evaluation of the advantages and the limitations of ANNs requires the ability to investigate the performance of ANNs with a large number of neurons. Such large scale ANNs can be implemented only with advances in optical or electronic neural networks. Another chapter in this book explores the issues in electronic neural network implementations. This chapter investigates the role of optical processing in ANNs.

ANNs are characterized by potentially full interconnectivity among a large number of simple neurons (processing elements). These features are ideally matched to the attributes of optical processors. Full interconnectivity among thousands of neurons would be virtually impossible in electronic hardware where each connection requires a wire. In optical neural computers, interconnections are represented by light beams which can freely cross each other. Also, as will be detailed later, use of volume holograms allows the packing of a large number of neurons in small volumes. Finally, the simplicity of the processing elements (usually equivalent to a summing amplifier followed by a nonlinearity) ensures that the limited dynamic range of the optical processors would not be a problem. More detailed discussion about the suitability of optical processing for neural networks can be found elsewhere [1,2,3].

Applications of ANNs so far have been mainly in three areas. The first among these are the optimization problems. Hopfield [4,5] showed in his pioneering work that a suitably configured ANN converges to the local minimum of an energy function. This can be used to advantage in solving minimization problems [6,7,8]. A second area of popular application of ANNs is pattern classification [9]. The ability of ANNs to *learn* [10] suitable discriminant functions from available training data is their major advantage in pattern recognition applications. The third application area for ANNs is as associative memories [11,12].

In conventional storage methods, each piece of information is stored in a unique slot and is retrieved by invoking the address of that slot. In contrast, associative memories store the information in a *distributed* manner throughout the entire memory. Information is usually recalled by probing the associative memory with an input that is either an incomplete or a noisy version of one of the stored pieces. The distributed nature of the information storage appears to provide more tolerance to defects and damage in the memory. Another reason for interest in associative memories is that human memory appears to be more akin to associative memories than to conventional memories.

In this chapter, we discuss the role of optical associative memories [13-28]. Since associative memories are designed to retrieve complete stored information from partial or noisy versions of it, they are useful for pattern classification also. Because of that, we will confine our attention to optical associative memories only.

The next section discusses some basic issues related to associative memories. Section 3 then provides details about four models for associative memory. So far, three main types of optical associative memories appear to have emerged. The first type utilize the ability of optical processors to perform matrix/vector multiplications, and these are discussed in Section 4. Optical processors are also very good at performing vector inner products, and associative memory implementations based on these are described in Section 5. Section 6 discusses the role of holograms in synthesizing optical associative memories. In Section 7, we outline a simple methodology that can be used for comparing the many associative memory models. Finally, Section 8 provides some concluding remarks.

2 BASICS OF ASSOCIATIVE MEMORIES

The basic concept of an associative memory can be described using Figure 1. The memory is synthesized from the set $\{(\mathbf{x}_i, \mathbf{y}_i),\ i = 1, 2, \ldots, M\}$, where M vector pairs are to be stored. Here $\mathbf{x}_i$ denotes the i-th "key" vector and $\mathbf{y}_i$ denotes the corresponding "recollection" vector. Both $\mathbf{x}_i$ and $\mathbf{y}_i$ are column vectors and can be of different sizes. For convenience, we will assume that both $\mathbf{x}_i$ and $\mathbf{y}_i$ are column vectors containing N elements. A properly designed associative memory should store the associations between the key vectors and the recollection vectors. For example, if $\mathbf{x}_3$ is the input to the associative memory in Figure 1, the output should be $\mathbf{y}_3$.

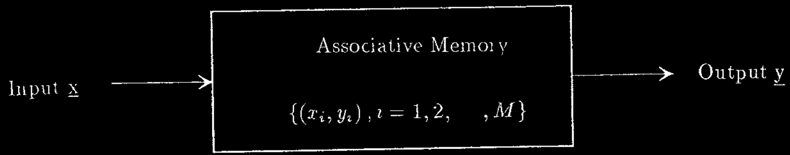


Figure 1: A block diagram description of the basic associative memory.

We are using vector notation since it allows us to utilize many results from linear algebra. Any input containing a finite number of elements can be represented using vector notation. For example, $\mathbf{x}_i$ may represent the social security number of a person and $\mathbf{y}_i$ may represent a raster scanned version of that individual's photograph. This is an example of Hetero Associative Memory (HAM), where the key vectors and recollection vectors are of different type. In contrast, an Auto Associative Memory (AAM) uses $\mathbf{y}_i = \mathbf{x}_i$, $i = 1, 2, \ldots, M$, so that the key vectors and recollection vectors are the same. From now on, we will focus our attention on AAM.

In AAM, we want $\mathbf{y}_i = \mathbf{x}_i$, $i = 1, 2, \ldots, M$. Obviously, this can be easily achieved if we represent the AAM by the $N \times N$ identity matrix $\mathbf{I}$, and we obtain output $\mathbf{y}$ by matrix/vector multiplication of input vector $\mathbf{x}$ by memory matrix $\mathbf{I}$. Such a memory matrix does produce the correct associations between the stored keys and recollections. However, it is not a desirable AAM for several reasons. First of all, it has no tolerance for errors in the input. If the input is a vector different from the M stored keys, the output will not correspond to a stored recollection. Also, if the memory matrix is damaged (e.g., some of the diagonal 1's in $\mathbf{I}$ are set to zero), the output $\mathbf{y}$ will not be the correct recollection vector. This indicates that this simple AAM is not fault tolerant. Hence, a robust AAM must exhibit the following desirable features:

- When the input $\mathbf{x} = \mathbf{x}_i$ (one of the stored keys), the output must be $\mathbf{y} = \mathbf{x}_i$.
- When the input $\mathbf{x} = \hat{\mathbf{x}}_i$ (where the cap indicates an incomplete or noisy version of $\mathbf{x}_i$), the output should be $\mathbf{x}_i$ provided $\hat{\mathbf{x}}_i$ is closer to it (using some distance measure) than to any other stored key vector.
- When the memory is damaged, we should still be able to recollect associations. This fault tolerance is not available in conventional memories without explicit coding.

Work is currently underway to make the these concepts more precise and to design AAMs capable of achieving the above objectives.

The ability of associative memories to store and retrieve information in a fault tolerant manner proves useful in many applications. Examples include content-addressable storage [11], database searching [29], and statistical pattern recognition [9]. The similiarity of associative memories to pattern classifiers is rather obvious. Both attempt to find the output in response to an input that is a noisy or an incomplete version of stored key vectors (prototypes or exemplars). In pattern classifiers, the output will be the class label whereas in associative memories, it may be a "cleaned-up" input vector (as in AAMs) or a different recollection vector (as in HAMs). The difference usually is that conventional statistical pattern recognition schemes use statistical models of the uncertainty to design optimal classifiers. In contrast, most associative memory designs are non-parametric and thus do not use explicit noise models. This may be a disadvantage (when good, realistic noise models are available) or an advantage (when input noise is unpredictable). The next section introduces four models for associative memory.

3 FOUR ASSOCIATIVE MEMORY MODELS

Several models have been proposed to store M pairs of vectors in an associative memory. All these models have two stages of operation. In the first stage, the associative memory *learns* the associations between inputs and outputs. This may be done by an explicit set of linear algebra operations as in the Hopfield memory [4] or by an iterative minimization of an error measure as in the back-propagation algorithm [10]. In the second stage, the associative memory produces a stable output vector in response to an input probe vector. This output vector may be obtained by a single operation on the input or by an iterative sequence of operations. This second stage is known as the *retrieval* process. In this section we describe briefly four different models for associative memories.

3.1 Hopfield Associative Memory [4]

Let $\mathbf{x}_1, \ldots, \mathbf{x}_M$ be the M column vectors (each with N elements) to be stored in the Hopfield memory. We will confine our attention to the case in which vector elements are either +1 or −1. These vectors are stored in an $N \times N$ memory matrix **W** defined as below:

$$\mathbf{W} = \mathbf{X}\mathbf{X}^T - M\mathbf{I} \tag{1}$$

where $\mathbf{X} = [\mathbf{x}_1\ \mathbf{x}_2\ \ldots\ \mathbf{x}_M]$ is an $N \times M$ data matrix whose columns are the vectors to be stored and where **I** is the $N \times N$ identity matrix. The subtraction of $M\mathbf{I}$ from the outer product matrix $\mathbf{X}\mathbf{X}^T$ in eq. (1) ensures that the diagonal elements of **W** are all zero.

Once the memory matrix **W** is constructed, it is used for retrieval as follows. Let $\mathbf{x}^0$ be the probe vector with N elements in it. Next, the following iterative procedure is used:

$$\mathbf{x}^{k+1} = \mathrm{Sgn}[\mathbf{W}\mathbf{x}^k - \mathbf{t}] \tag{2}$$

where **t** is an N-dimensional column vector containing the thresholds (assumed to be zero unless specified otherwise) and the superscript k indicates the iteration index. The Sgn[x] results in a vector with entries equal to +1 (if the original elements were non-negative) and to −1 (if the original elements were negative). Hopfield [4] proved that an asynchronous version of the update rule in eq. (2) will always converge to a stable vector. A stable vector is reached if $\mathbf{x}^{k+1} = \mathbf{x}^k$ for some k. After this, updates stop.

Empirical investigations revealed that only $M < 0.15N$ vectors can be stored reliably in this associative memory, which is rather inefficient. However, the vectors are stored in a *distributed* manner in **W**, so we expect to achieve good fault tolerance. Theoretical studies [30,31,32] have been carried out to characterize the storage capacity of Hopfield associative memories. In addition to its poor storage capacity, Hopfield associative memory suffers from several other weaknesses [33] when viewed as a pattern classifier, some of which are listed here:

- When we construct a storage matrix **W** from a set of vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\}$, it is quite likely that several other spurious vectors also become stored memories.

- There is no guarantee that a **W** constructed from the set $\{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$ will have these vectors as its stable vectors.

- Examples can be constructed where a probe vector converges to a vector other than its nearest neighbor (in the Hamming distance sense).

However, we must emphasize that the Hopfield associative memory stores information in a distributed manner, thus providing a degree of fault tolerance. Modifications proposed to improve the performance of Hopfield memories include: layering the networks [34], using non-zero diagonal terms [35], modifying the threshold condition [36], and using limited interconnections between neurons [37].

3.2 Generalized Inverse Memory [11]

The Generalized Inverse memory proposed by Kohonen [11] is based on the minimization of mean squared error (MSE) in the output. Let $\mathbf{x}_i$, $i = 1, 2, \ldots, M$, denote the key vectors and $\mathbf{y}_i$ denote the corresponding recollections. Then the MSE is defined by

$$\mathrm{MSE} = \frac{1}{M}\sum_{i=1}^{M} \| \mathbf{y}_i - \mathbf{W}\mathbf{x}_i \|^2 \tag{3}$$

where $\mathbf{W}$ is the memory matrix and $\| \ \|^2$ represents the squared norm of the vector. We can show that the MSE is minimized if we use

$$\mathbf{W} = \mathbf{Y}\mathbf{X}^{+} \tag{4}$$

where $\mathbf{Y}$ is an $N \times M$ matrix with $\mathbf{y}_i$ as its i-th column and $\mathbf{X}^{+}$ is the generalized inverse of $\mathbf{X}$. This generalized inverse is given by $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ if the columns of $\mathbf{X}$ are linearly independent and by $\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}$ if the rows of $\mathbf{X}$ are linearly independent. For the special case where $\mathbf{X}$ is invertible, $\mathbf{X}^{+}$ is the same as $\mathbf{X}^{-1}$ and the MSE is zero.

It is easy to see that retrieval can be accomplished in a single step. To observe this, let us compute the following assuming that the columns of $\mathbf{X}$ are linearly independent:

$$\begin{aligned} \mathbf{WX} &= (\mathbf{Y}\mathbf{X}^{+})\mathbf{X} \\ &= \mathbf{Y}(\mathbf{X}^{\mathbf{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathbf{T}}\mathbf{X} \\ &= \mathbf{Y} \end{aligned} \tag{5}$$

For the special case of AAMs, $\mathbf{Y} = \mathbf{X}$ and we get the correct recollection vectors in one shot.

Generalized Inverse memory stores the associations in a distributed manner and is thus expected to be fault tolerant. It is also attractive in the sense that there is no need for iterations and that the stored vectors are not limited to binary vectors. However, its ability to reconstruct complete information from partial inputs has not been investigated thoroughly. Theoretical analyses have been carried out [38-42] to understand the effects of input noise on the recollection ability of Generalized Inverse memories.

3.3 Spectral Neural Network [43]

One of the difficulties with the Hopfield associative memory is its limited storage capacity. This may be due in part to the construction method of the Hopfield memory matrix, in which no explicit attempts are made to control the locations of stable states in the energy space. As a result, the stored states tend to cluster which makes it difficult to distinguish one state from another. The Spectral algorithm [43] alleviates this problem by constructing the memory matrix in a manner that "spreads" out the stable states.

Suppose the threshold vector $\mathbf{t}$ in eq. (2) is an all zero vector and suppose $\mathbf{x}^k$ is an eigenvector of the matrix $\mathbf{W}$. Since $\mathbf{W}\mathbf{x}^k = \lambda\mathbf{x}^k$ where λ is the eigenvalue, then $\mathbf{x}^{k+1} = \mathbf{x}^k$ as long as $\lambda > 0$ and $\mathbf{x}^k$ is a vector containing $+1$ and -1 only. The Spectral algorithm is based on this observation. In this method, we first determine eigenvalues λ_r, $1 \leq r \leq M$, as below:

$$\lambda_r = N - \frac{1}{M-1}\sum_{s=1, s\neq r}^{M} < \mathbf{x}_r, \mathbf{x}_s > \tag{6}$$

where $< \mathbf{x}_r, \mathbf{x}_s >$ indicates the inner product between the two vectors. Let Λ denote the $M \times M$ diagonal matrix where λ_i is the i-th diagonal element. Then the memory matrix $\mathbf{W}$ is given by

$$\mathbf{W} = \mathbf{X}\Lambda(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T \tag{7}$$

It is easy to verify that $\mathbf{WX} = \mathbf{X}\Lambda$, and since all the eigenvalues are positive, $\mathrm{Sgn}[\mathbf{WX}] = \mathbf{X}$. Therefore, this matrix can recollect the stored vectors correctly. It is interesting to compare the memory matrices in eqs. (4) and (7). For AAMs, the only difference is the extra diagonal matrix Λ introduced in the Spectral algorithm.

3.4 Ho-Kashyap Associative Memory [44-47]

This procedure has roots in classical pattern recognition theory [44,45] where it is used to design linear discriminant functions. Hassoun and Youssef [46,47] modified it to construct a new associative memory matrix. To see the equivalence between the two problems, consider eq. (2) where both $\mathbf{x}^k$ and $\mathbf{x}^{k+1}$ are N-dimensional column vectors with entries +1 or −1. If we consider the i-th row of $\mathbf{W}$ as $\mathbf{w}_i$, we can rewrite eq. (2) as below:

$$x_{m,i} = \mathrm{Sgn}[\mathbf{w}_i^T\mathbf{x}_m - t_i], \quad 1 \le i \le N, \quad 1 \le m \le M \tag{8}$$

where the superscript T denotes the transpose. Since $x_{m,i}$ is either +1 or −1, each $(\mathbf{w}_i, t_i)$ represents one linear discriminant function capable of classifying M vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\}$ into 2 classes. Thus N linear discriminant functions $\mathbf{w}_i$, $i = 1, \ldots, N$, are designed to find the memory matrix $\mathbf{W}$. Once again the retrieval operation is carried out in a single step.

It has been shown [45] that this algorithm will always converge in a finite number of steps if the underlying vectors are "linearly separable". However, this is computationally demanding because N linear discriminant functions must be determined and each requires many matrix operations.

3.5 Other Associative Mappings

Several other schemes have been proposed for associative mappings. These include: higher order neural networks [48,49,50], exponential associative memories [51], and polynomial memories [52]. It is not our intention to provide a discussion of all these models. We included the above four since they appear to be more easily implemented on optical architectures than others.

4 OUTER PRODUCT ASSOCIATIVE MEMORIES

As discussed in Section 3, the memory matrix $\mathbf{W}$ is obtained from the outer products of the M vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\}$ to be stored. If we allow the diagonal elements of $\mathbf{W}$ to be nonzero, we can obtain $\mathbf{W}$ as below:

$$\mathbf{W} = \sum_{i=1}^{M} \mathbf{x}_i\mathbf{x}_i^T \tag{9}$$

Once the memory matrix W is synthesized, information can be retrieved using eq (2). Since this W is constructed from the outer products of memory vectors, this method is known as the outer product associative memory The main operation involved here is the matrix/vector multiplication In this section, we will discuss how outer product associative memories can be implemented using optical matrix/vector multipliers.

The basic optical matrix/vector multiplier [53] is shown in Figure 2 Here the vector x^k at the k-th iteration is represented by a laser diode or light emitting diode array in plane P_1. This array is then imaged horizontally and expanded vertically onto the mask in plane P_2 The mask encodes the matrix W The light leaving plane P_2 is then imaged vertically and focused horizontally onto the photodetector array in plane P_3. Hence, the vector detected by the array in plane P_3 is given by Wx^k Many other architectures for and applications of optical matrix/vector multipliers can be found elsewhere [54]

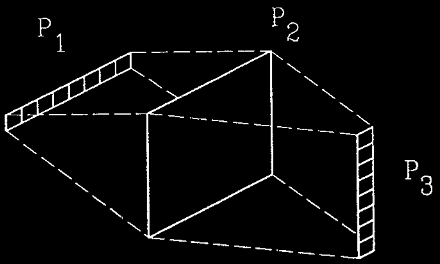


Figure 2: Basic optical matrix/vector multiplier

Once Wx^k is obtained, we can carry out the required subtraction of the threshold vector t and the hard clipping operation in electronic hardware This produces x^{k+1} which can be used as the input for the array in plane P_1 for the next iteration. The basic iterative optical matrix/vector multiplier has been the focus of much attention for several years A good summary of its applications, algorithms, and architectures can be found elsewhere [55]

Any of the associative memory schemes using eq (2) for retrieval can be implemented using the matrix/vector multiplier in Figure 2. In fact, the first optical implementation [14] of the Hopfield memory was carried out this way However, there are several limitations to this architecture The need for conversion between optics and electronics slows down the system even though the computationally intensive matrix/vector multiplications can be carried out rapidly The encoding of the mask W in plane P_2 requires that it be computed off-line Thus, every time a new input is to be stored, we need to recompute the entire W If W is going to change frequently, we need to use transmittance masks which can be externally addressed Such devices are known as Spatial Light Modulators (SLMs) What we need for plane P_2 is a large sized, rapidly addressable, 2-D SLM While these exist, they are expensive Advances in 2-D SLMs such as the magneto-optic spatial light modulator [56] and the Texas Instruments Deformable Mirror Device [57] are definitely going to help this implementation

A significant advantage of the outer product associative memories is that the vectors are stored in the entire matrix W in a distributed fashion Therefore, damage to W (e g , setting some elements of **W** to zero) degrades the retrieval gracefully Similarly, quantizing the dynamic range of **W** to a few bits may not be a big problem Since some of the available real-time 2-D SLMs can accommodate only two levels, this is certainly convenient.

5 INNER PRODUCT ASSOCIATIVE MEMORIES

In Section 4, we showed how the Hopfield associative memory can be implemented using an outer product scheme However, it can also be implemented using an inner product method. To see this, let us substitute eq (9) in eq (2):

$$\begin{aligned} \mathbf{x}^{k+1} &= \mathrm{Sgn}[\mathbf{W}\mathbf{x}^k - \mathbf{t}] \\ &= \mathrm{Sgn}[\{\sum_{i=1}^{M} \mathbf{x}_i\mathbf{x}_i^T\}\mathbf{x}^k - \mathbf{t}] \\ &= \mathrm{Sgn}[\sum_{i=1}^{M} a_i^k \mathbf{x}_i - \mathbf{t}] \end{aligned} \tag{10}$$

where the coefficient a_i^k, $i = 1, 2, \ldots, M$, is given by the inner product $(\mathbf{x}_i^T\mathbf{x}^k)$ between the i-th stored vector and the estimate of the recollection vector at the k-th iteration In this approach, we obtain $\mathbf{W}\mathbf{x}^k$ by first determining the inner products a_i^k for $i = 1, 2, \ldots, M$ and then using these in forming the linear combination $\sum_{i=1}^{M} a_i^k\mathbf{x}_i$ A basic optical implementation of this is shown in Figure 3.

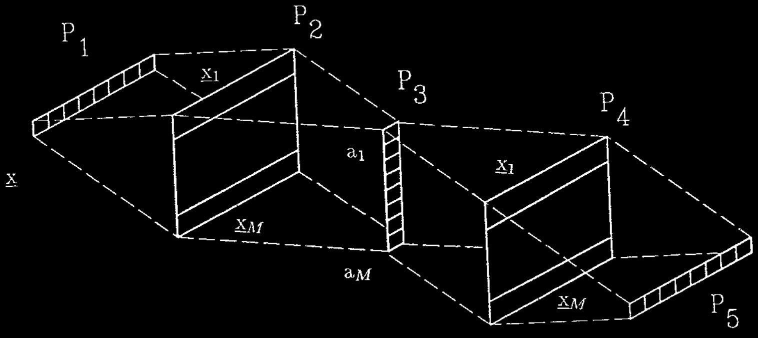


Figure 3: A basic optical implementation of inner product associative memory

The probe vector **x** is placed in the laser diode array in plane P$_1$ This light is then imaged horizontally and expanded vertically onto the mask in plane P$_2$ This mask has M rows and N columns with each row representing one of the stored memory vectors The light leaving plane P$_2$ is imaged vertically and focused horizontally onto the detector array in plane P$_3$ The vector detected by P$_3$ is given by $\mathbf{a} = [a_1 \; a_2 \; \ldots \; a_M]^T$ The inner products are then used to modulate a source array in plane P$_3$ The light from plane P$_3$ is imaged vertically and expanded horizontally onto the mask in plane P_4. The masks in planes P_2 and P_4 are identical. Finally, the light leaving plane P_4 is focused vertically and imaged horizontally, yielding $\mathbf{Wx}$ as desired. The subtraction of the threshold vector $\mathbf{t}$ and the nonlinear hard clipping operation are carried out electronically. Since the same masks are used in planes P_2 and P_4, we can use a mirror in plane P_3 to avoid needing two 2-D spatial light modulators.

Consider the case where we set the maximum among $\{a_1, a_2, \ldots, a_M\}$ to one and the others to zero. When the stored vectors and input probe are binary bipolar (e.g., elements are +1 or −1), maximizing the inner products is equivalent to minimizing the Hamming distance between the probe $\mathbf{x}$ and the stored vectors. Thus, the inner product associative memory will select the nearest neighbor from among the set of stored vectors. This observation was used [24] to suggest an associative memory scheme that is guaranteed to converge to the nearest neighbor. Selecting the maximum from the inner products is only one example of a nonlinearity. Other nonlinearities [27,58] have been proposed for use in this architecture.

One of the advantages of the inner product associative memory is that it is easy to add new vectors and delete old ones simply by adding rows and deleting rows. In contrast, we need to compute new memory matrices for outer product implementations. However, each stored vector is in a separate row of the medium and is thus *not* represented in a distributed manner. Hence, inner product associative memories do not provide good fault tolerance. The storage capacity, on the other hand, can be large compared to that of outer product memories.

Higher order associative memories [48,49,50] can provide increased storage capacity because of the increased number of degrees of freedom available. These higher order associative memories can be implemented using the inner product method in Figure 3 provided the inner products a_i, $i = 1, 2, \ldots, M$, are passed through a suitable nonlinearity. Using a square-law nonlinearity produces quadratic associative memories [59,60] and using a maximum finding nonlinearity produces nearest-neighbor finding associative memories [24]. However, as stated before, inner product associative memories are unattractive since they do not provide the fault tolerance that is characteristic of useful artificial neural networks.

6 HOLOGRAPHIC ASSOCIATIVE MEMORIES

The use of holograms as associative memories has been well known to optical processing researchers for many years. Much of the earlier use of holograms was in their ability to reconstruct 3-D images and their ability to encode complex-valued spatial functions on a medium capable of representing only positive-valued functions. This latter property was particularly useful for optical matched filtering [61]. Here our interest is in the hologram's ability to reconstruct one of two wavefronts when given the other.

Let us consider a simplified version of the hologram synthesis process. As shown in Figure 4, imagine two wavefronts incident on the medium in plane P_1. One of these is a plane wave of light (at an angle to the optical axis) indicated by $Ke^{-j\alpha x}$, where α is related to the angle. The other wavefront $S(x, y)$ contains the complex variations to be encoded on the hologram.

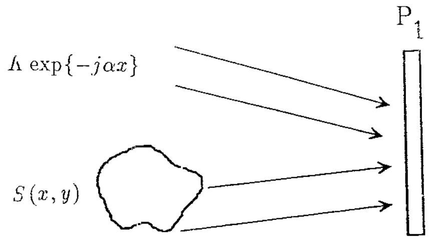


Figure 4: Hologram recording process

To be able to form a useful hologram in plane P$_1$, the two wavefronts must be coherent Roughly speaking, this means that the light is monochromatic and both wavefronts must have the same original light source Also, the two paths must have approximately the same length Otherwise, we do not get interference in plane P$_1$ When all the above conditions are met, the intensity incident on plane P$_1$ is given by:

$$\begin{aligned} \mathrm{I}(x,y) &= \mid \mathcal{S}(x,y) + K\mathrm{e}^{-j\alpha x} \mid^2 \\ &= \mid \mathcal{S}(x,y) \mid^2 \ + \ K^2 \ + \ K\mathcal{S}^*(x,y)\mathrm{e}^{-j\alpha x} \ + \ K\mathcal{S}(x,y)\mathrm{e}^{+j\alpha x} \end{aligned} \tag{11}$$

where the superscript asterisk indicates complex conjugation The intensity I(x,y) in eq (11) is positive, yet it encodes the complex wavefront $\mathcal{S}(x,y)$ The medium used in plane P$_1$ will have a transmittance t(x,y) that is proportional to I(x,y)

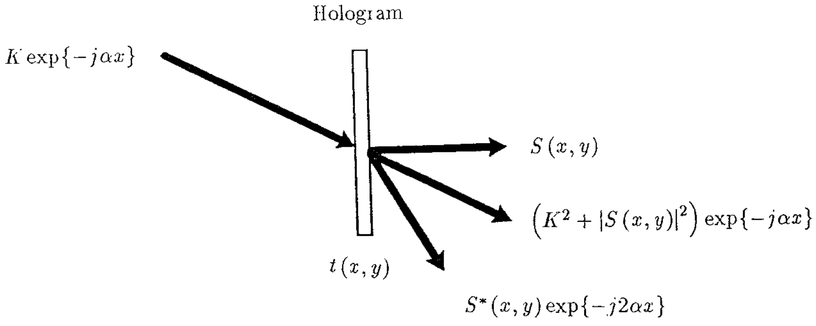

Hologram
$K\exp\{-j\alpha x\}$
$S\,(x,y)$
$\left(K^2+|S\,(x,y)|^2\right)\exp\{-j\alpha x\}$
$t\,(x,y)$
$S^*(x,y)\exp\{-j2\alpha x\}$

Figure 5: Retrieval from a hologram using the reference as the probe wavefront.

To retrieve the wavefront $\mathcal{S}(x,y)$ from the hologram t(x,y), we reilluminate it by the plane wave $K\mathrm{e}^{-j\alpha x}$ (same reference wavefront that was used in the hologram recording process) The light leaving the hologram in Figure 5 is then given by:

$$\begin{aligned} \mathrm{O}(x,y) &= \mathrm{t}(x,y)\ \ K\mathrm{e}^{-j\alpha x} \\ &= K\mathrm{I}(x,y)\mathrm{e}^{-j\alpha x} \\ &= K(K^2 + \mid \mathcal{S}(x,y) \mid^2)\mathrm{e}^{-j\alpha x} + \ K^2\mathcal{S}^*(x,y)\mathrm{e}^{-j2\alpha x} + \ K^2\mathcal{S}(x,y) \end{aligned} \tag{12}$$

where we have ignored some proportionality constants As can be seen from Figure 5, the desired complex wavefront $\mathcal{S}(x, y)$ emerges separated from the others In some sense, the hologram has recorded the association between the reference wavefront $Ke^{-j\alpha x}$ and the object wavefront $\mathcal{S}(x, y)$ Thus, we can retrieve the object $\mathcal{S}(x, y)$ by using the reference as the probe wavefront On the other hand, if $\mathcal{S}(x, y)$ is used as the probe, then the output will have three terms, one of which is $K \mid \mathcal{S}(x, y) \mid^2 e^{-j\alpha x}$ This is a distorted version of the reference wavefront Thus, we can use either of the two wavefronts to generate the other However, only plane wave probes lead to undistorted recollections These ideas were originally suggested by Gabor [62,63,64]

The above explanation of the use of holograms as associative memories treats the holograms as static However, it is essential that we use *dynamic* holograms in order to learn new associations Also, it is important to use volume holograms [65,66,67] to take advantage of the packing density in optics and to utilize the 3-D attributes of optics

The best materials for use as dynamic holograms are photorefractive crystals [68,69,70] In these crystals, the refractive index of the material changes as a function of the incident intensity. Thus, the weights to be used for associative mappings can be easily changed externally However, one serious problem is that these weights may change during the retrieval stage This can result in "forgetting" of the associations during retrieval Also, the current generation of photorefractive crystals suffer from crosstalk (noise) between stored mappings and require high light intensities for adequate speed performance Much research effort is underway to alleviate these problems

7 A COMPARISON OF FOUR ASSOCIATIVE MEMORY MODELS [71]

Associative memories are attractive because they can provide fault tolerance and exhibit the ability to retrieve complete information from incomplete or noisy versions of stored information. In this section, we present an abbreviated version of a detailed study [71] conducted to compare four associative memory models from the above consideration These models are: Hopfield associative memory, Generalized Inverse, Spectral method, and Ho-Kashyap associative memory. All these were introduced in Section 3, and here we confine our attention to comparing them using computer simulations We have not included many other important memory models in our comparison since that would make this too long Our main objective is to provide a simple methodology for comparing these associative memory networks

7.1 Details of the Simulations

The goal of the simulations was to determine the retrieval performance as a function of the number of stored vectors M, the corruption level (added noise) of the probe vector, and damage to the memory matrix The memory matrix $\mathbf{W}$ was damaged in two different ways. In the first, some matrix elements (randomly selected) were set to zero This simulated the loss of interconnections In the second, the matrix elements were quantized to various number of bits We felt that this was a more realistic simulation of the hardware storage requirements for an associative memory

The evaluation of the retrieval performance of associative memories is rather tricky Two different evaluation strategies were used In the first (called Test Program - Strict or TP-S), we examined the maximum number of bits we could corrupt in a probe vector and still achieve

100% retrieval. The results from TP-S give some idea of how each memory model would perform in situations that demand perfect retrieval. In the second test (called Test Program - Loose or TP-L), we determined the percentage of probe vectors that converged to within a specified Hamming distance of the correct output vector. TP-L is a more realistic measure for evaluating neural network associative memories.

The memory vectors to be stored were of length $N = 50$, and each bit was set randomly to $+1$ or -1 with equal probability. A total of 30 vectors were used, with an average Hamming distance of 24.79 and a standard deviation of 3.44. If two binary, bipolar vectors of length 50 are orthogonal, their Hamming distance would be 25. Therefore, the vectors represent a set of almost orthogonal vectors.

For TP-S evaluations, we selected one of the stored vectors as the probe vector and inverted k bits randomly. The value of k ranged from 0 to 30. We probed the network 25 times with the chosen probe vector. If the correct vector was retrieved in all 25 attempts, we moved on to a higher k value. For any k, even if one of the 25 trials results in a wrong recollection, then that k value indicated the maximum probe vector corruption tolerable. After all possible probe vectors were tested, we averaged the k values for each choice of M, the number of vectors stored.

In TP-L evaluations, the probe vector was corrupted just as above. However, when the network converged to a vector, its Hamming distance from the correct recall vector was determined. This was used to derive the percentage of probe vectors that fell within a specified Hamming distance of the correct stored vector.

7.2 Discussion of Results

In this subsection, we provide an abbreviated discussion of the results presented elsewhere [71]. To start with, we show in Figure 6 the maximum number of bits that can be corrupted in the probe vector and still achieve 100% retrieval. In Figure 7, we show similar results when the matrix was damaged by 50% (i.e., half the elements were set to zero). The results obtained when the matrix elements were quantized to one bit (i.e., 2 levels) are shown in Figure 8. The 50% damage and 1 bit quantization were combined and the results are shown in Figure 9.

The results in Figures 6-9 suggest that the fault tolerance to probe vector corruption decreases as we try to pack more vectors into the memory. As the memory matrix is damaged, the retrieval fidelity drops as more vectors are stored. With no quantization or damage (Figure 6), the Ho-Kashyap method seems to provide a definite fidelity and capacity advantage over the other schemes. The Generalized Inverse and Spectral method perform slightly below the Ho-Kashyap, while the Hopfield model displays a substantially lower storage capacity. As the amount of matrix damage and quantization increases, all the networks begin to exhibit a degraded response curve, with the Ho-Kashyap deteriorating most. At the one bit quantization level, the Hopfield network has a slightly better response curve than the others. However, the storage capacity is so low that small differences in retrieval fidelity have little meaning.

In Figure 10, we show the percentage of probe vectors that converged to within five units (Hamming distance) of the correct vector when the probes were corrupted by 0 bits, 6 bits, and 12 bits. We show similar results in Figure 11 (when the matrix is damaged 50%), Figure 12 (when the matrix is quantized to one bit), and Figure 13 (when the matrix is damaged 50% and quantized to one bit).

On the average, the number of vectors that converged to within five units of the correct recollection decreases as M is increased. As seen before, the Ho-Kashyap method performed well in the absence of memory damage and quantization. However, its performance degraded

Max. Probe Corruption vs. Vector Storage Capacity (N = 50)

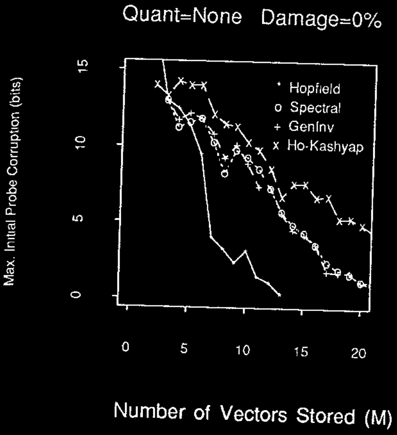


Figure 6

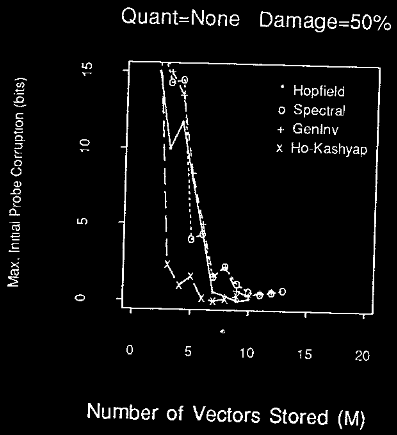


Figure 7

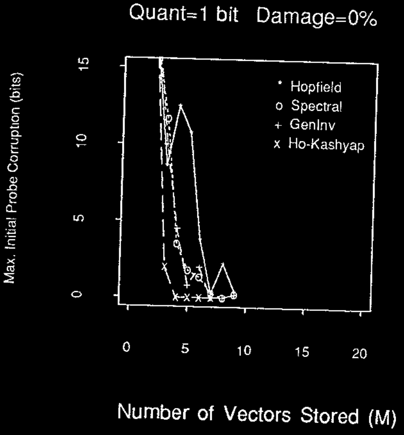


Figure 8

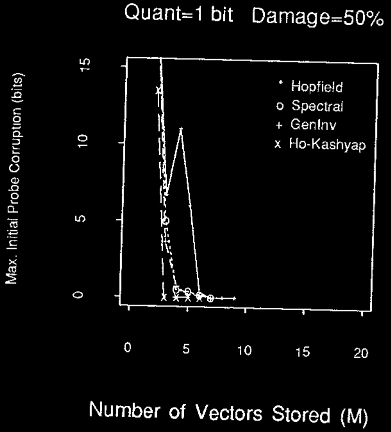
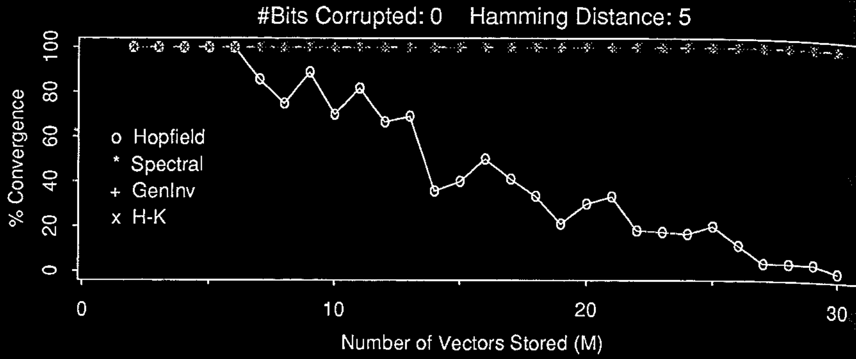
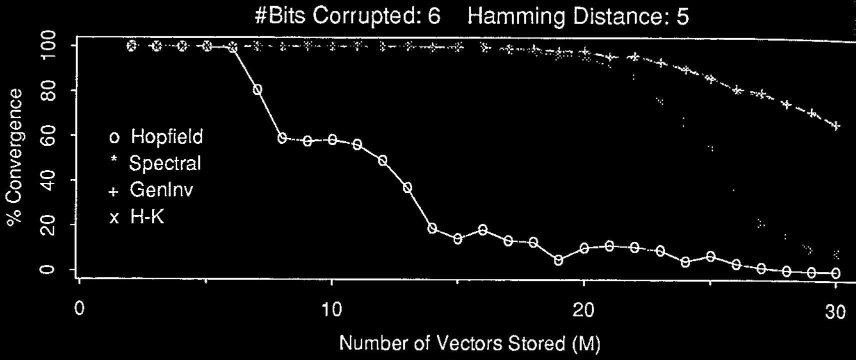


Figure 9

% Probe Convergence vs. Vector Storage Capacity
(N=50 Quant=None Damage=0%)





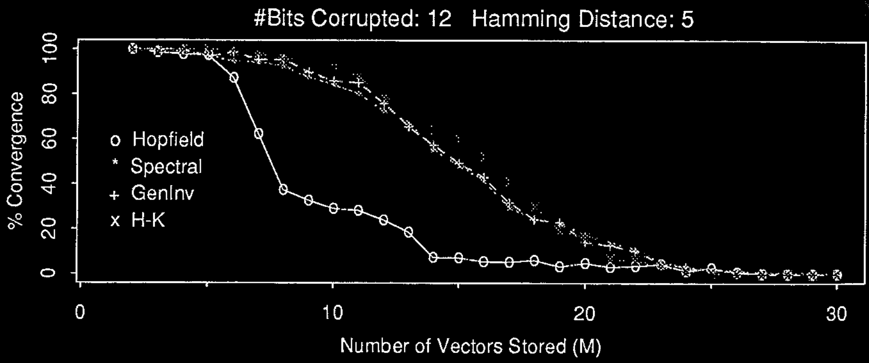
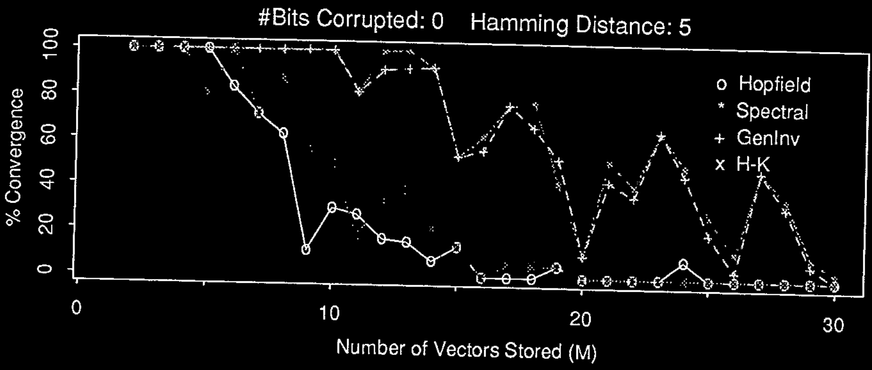
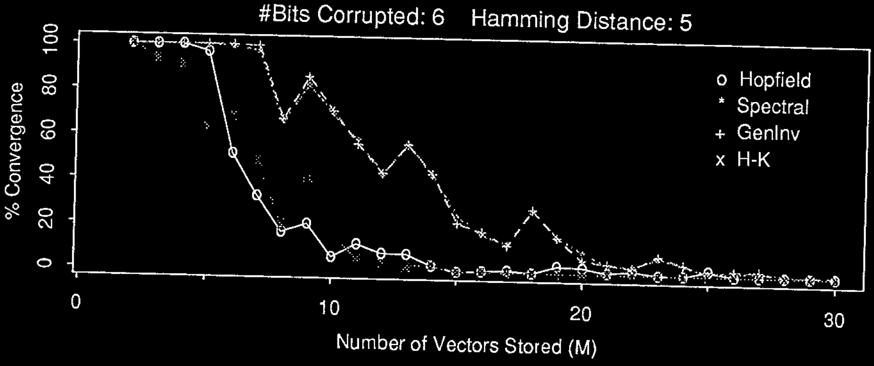


Figure 10

% Probe Convergence vs. Vector Storage Capacity
(N=50 Quant=None Damage=50%)





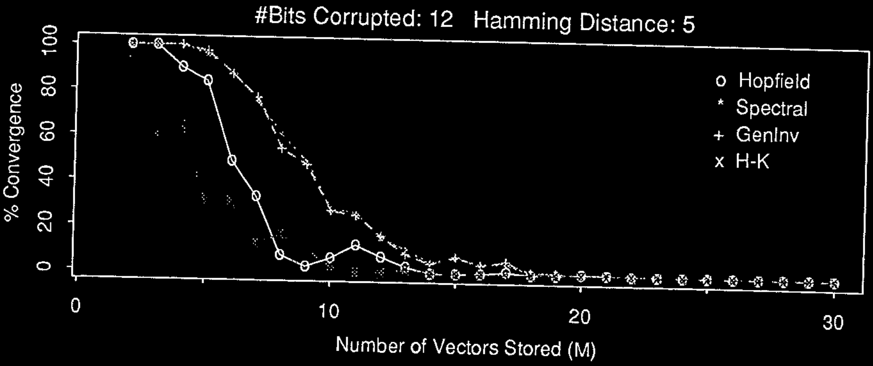
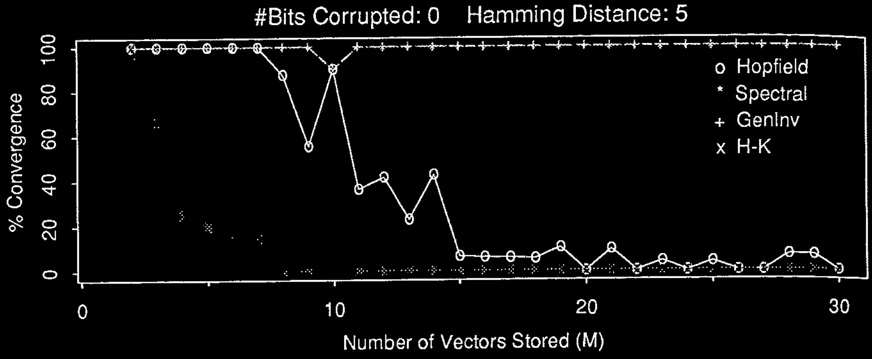
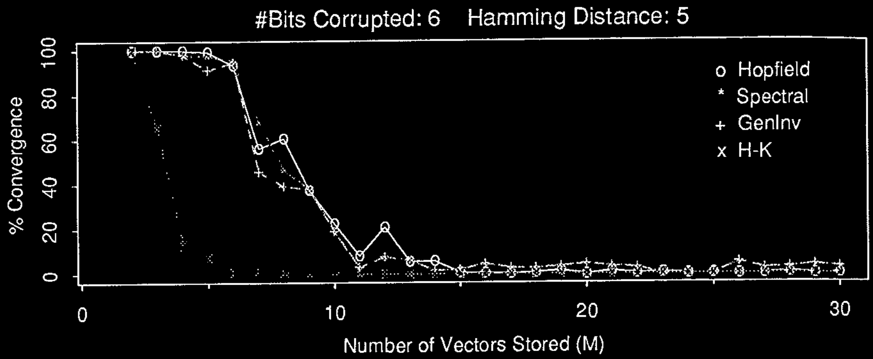


Figure 11

% Probe Convergence vs. Vector Storage Capacity
(N=50 Quant=1 bit Damage=0%)





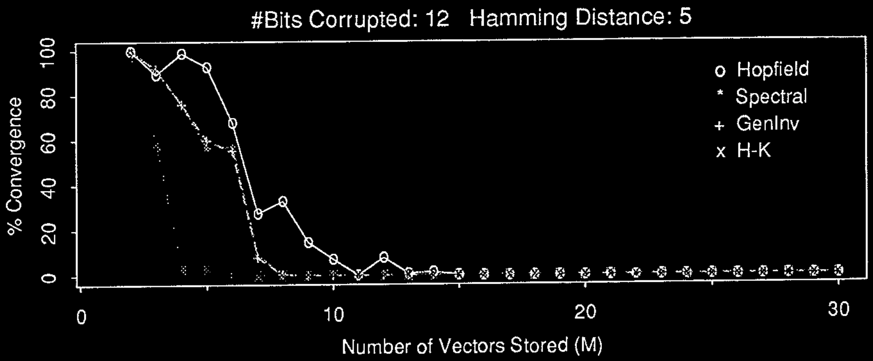
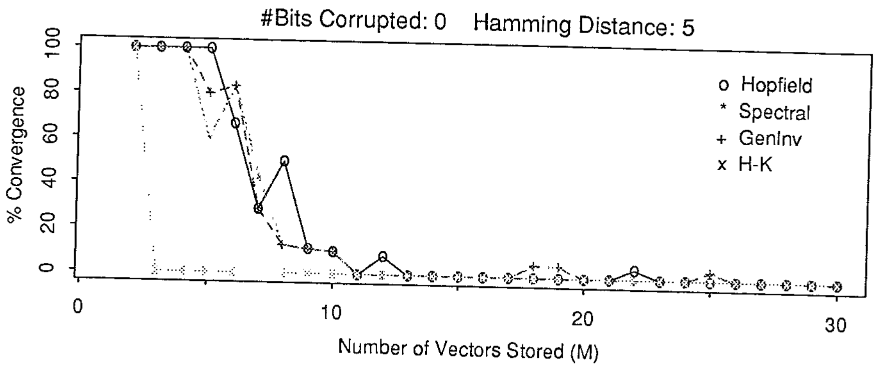
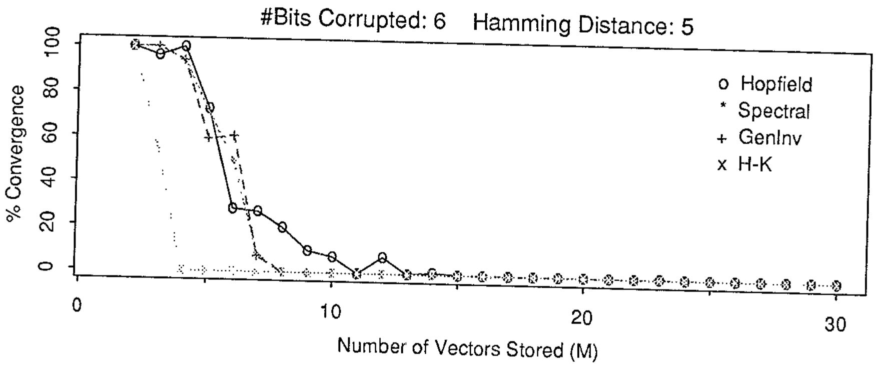


Figure 12

% Probe Convergence vs. Vector Storage Capacity
(N=50 Quant=1 bit Damage=50%)





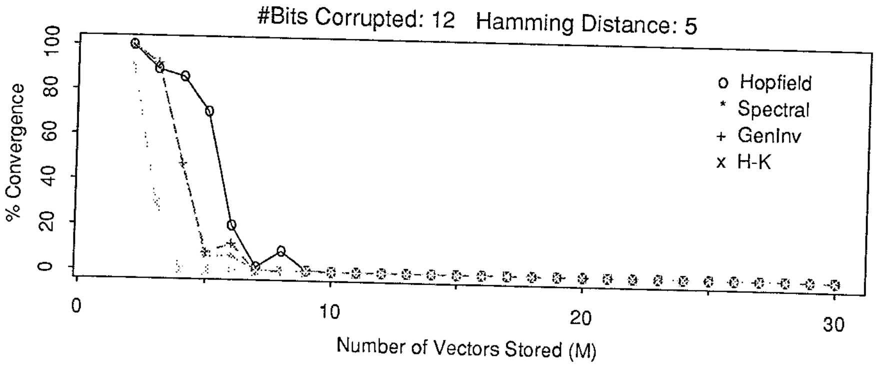


Figure 13

significantly in the presence of noise. Overall, the performance of all four methods appears to be similar in the presence of severe matrix damage and quantization. Therefore, the many associative memory models proposed appear to provide no appreciable advantages over the Hopfield method when we take into account the desirable fault tolerance attributes.

8 CONCLUSIONS

In this chapter, we presented the basic methods and motivations for the use of optical associative memories. The motivations include: possibility of large sized neural networks because of the 3-D nature of optics, full global interconnectivity, and the compatibility of the low accuracy of optical processors to the requirements of neural networks. However, there are many other issues and methods [72-79] we did not discuss here for reasons of brevity. We advise the reader to consult the extended reference list we have provided.

The success of optical neural computers will critically depend on advances in the following areas

- Improving photorefractive SLMs.
- Better understanding of the associative memory algorithms
- Improving the interface between optical and electronic hardware

Acknowledgements

The authors gratefully acknowledge partial support of this research by the Air Force Office of Scientific Research under grant number AFOSR-89-0551

REFERENCES

[1] Y. Abu-Mostafa and D. Psaltis, "Optical Neural Computers," *Scientific American*, 88-95

[2] Special Issue on Neural Networks, *Applied Optics*, vol. 23, no. 26, 1 December 1987

[3] H. J. Caulfield, J. Kinser, and S. K. Rogers, "Optical Neural Computers," *Proc. of IEEE*, vol. 77, no. 10, 1573-1583, October 1989

[4] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Natl. Acad. Sci. USA*, vol. 79, 2554-58, 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, 3088-92. 1982

[6] J. J. Hopfield and D. W. Tank, "Neural computation of decisions in optimization problems," *Biological Cybernetics*, vol. 52, 141-152, 1985

[7] M. Takeda and J. W. Goodman, "Neural networks for computation: number representations and programming complexity," *Applied Optics*, vol. 25, no. 18, 3033-3046, 1986.

[8] E. Barnard and D. P. Casasent, "Multitarget tracking with cubic energy optical neural nets," *Applied Optics*, vol. 28, no. 4, 791-798, 1989

[9] R. P. Lippman, "Pattern Classification using Neural Networks," *IEEE Communications Magazine*, 47-50, 59-64, November 1989.

[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," Chapter 8 in *Parallel Distributed Processing*, vol. 1, D. E. Rumelhart and J. L. McClelland, MIT Press, Cambridge, MA, 1986.

[11] T. Kohonen, *Self-organization and Associative Memory*, Springer-Verlag, Berlin, 1984.

[12] G. E. Hinton and J. A. Anderson, *Parallel Models of Associative Memory*, Laurence Erlbaum Associates, Hillsdale, NJ, 1981.

[13] D. Psaltis and N. H. Farhat, "Optical information processing based on an associative memory model of neural nets with thresholding and feedback," *Optics Letters*, vol. 10, 98-100, 1985.

[14] N. H. Farhat, D. Psaltis, A. Prata, and E. Paek, "Optical implementation of the Hopfield model," *Applied Optics*, vol. 24, 1469-75, 1985.

[15] A. D. Fisher and C. L. Giles, "Optical adaptive associative computer architectures," *Proc. of IEEE Compcon Spring*, 342-344, 1985.

[16] A. D. Fisher, W. L. Lippincott, and J. N. Lee, "Optical implementations of associative networks with versatile adaptive learning capabilities," *Applied Optics*, vol. 26, 5039-54, 1987.

[17] A. Yariv, S. Kwong, and K. Kyuma, "Demonstration of an all-optical associative holographic memory," *Appl. Phys. Lett.*, vol. 48, 114-116, 1986.

[18] H. J. White, N. B. Aldridge, and I. Lindsay, "Digital and analog holographic associative memories," *Optical Engineering*, vol. 27, 30-37, 1988.

[19] B. H. Soffer, G. J. Dunning, Y. Owechko, and E. Marom, "Associative holographic memory with feedback using phase-conjugate mirrors," *Optics Letters*, vol. 11, 118-120, 1986.

[20] J. M. Kinser, H. J. Caulfield, and J. Shamir, "Design for a massive all-optical bidirectional associative memory: The big BAM," *Applied Optics*, vol. 27, 3992-3999, 1988.

[21] D. Z. Anderson, "Coherent optical eigenstate memory," *Optics Letters*, vol. 11, 56-58, 1986.

[22] D. Psaltis, A. A. Yamamura, K. Hsu, S. Lin, X. Gu, and D. Brady, "Optoelectronic implementations of neural networks," *IEEE Communications Magazine*, 37-40, November 1989.

[23] M. Lemmon and B. V. K. Vijaya Kumar, "Competitively inhibited optical neural networks using two-step holographic materials," Technical Digest, OSA Topical meeting on *Optical Computing*, Salt Lake City, Utah, 36-39, 1989.

[24] B. L. Montgomery and B. V. K. Vijaya Kumar, "Nearest neighbor non-iterative error-correcting optical associative processor," *Proc. of SPIE*, vol. 638, 83-90, 1986.

[25] L. S. Lee, H. M. Stoll, and M. C. Tackitt, "Continuous-time optical neural network associative memory," *Optics Letters*, vol. 14, 162-164, 1989.

[26] E. G. Paek and D. Psaltis, "Optical associative memory using Fourier transform holograms," *Optical Engineering*, vol. 26, no. 5, 428-33, 1987.

[27] R. Athale, H. Szu, and C. Friedlander, "Optical implementation of associative memory with controlled nonlinearity in the correlation domain," *Optics Letters*, vol. 11, no. 7, 482-84, 1986.

[28] B. Macukow and H. H. Arsenault, "Optical associative memory model based on neural networks having variable interconection weights," *Applied Optics*, vol. 26, no. 5, 924-28, 1987.

[29] J. M. Char, V. Cherkassky, H. Wechsler, and G. L. Zimmerman, "Distributed and fault-tolerant computation for retrieval tasks using distributed associative memories," *IEEE Trans. Computers*, vol. 37, 484-490, 1988.

[30] Y. S. Abu-Mostafa and J. M. St. Jacques, "Information capacity of the Hopfield model," *IEEE Trans. Inf. Theory*, vol. IT-31, 461-464, 1985.

[31] R. J. McEliece, E. C. Posner, E. R. Rodemich, and S. S. Venkatesh, "The capacity of the Hopfield associative memory," *IEEE Trans. Inf. Theory*, vol. IT-33, 461-82, 1987.

[32] A. Dembo, "On the capacity of associative memories with linear threshold functions," *IEEE Trans. Inf. Theory*, vol. 35, no. 4, 709-720, 1989.

[33] B. L. Montgomery and B. V. K. Vijaya Kumar, "Evaluation of the use of the Hopfield neural net model as a nearest-neighbor algorithm," *Applied Optics*, vol. 25, 3759-66, 1986.

[34] M. J. Little and C. S. Bak, "Enhanced memory capacity of a Hopfield neural network," *Proc. SPIE*, San Diego, CA, 1986.

[35] G. R. Gindi, A. F. Gmitro, and K. Parthasarathy, "Hopfield model associative memory with nonzero diagonal terms in memory matrix," *Applied Optics*, vol. 27, no. 1, 129-134, 1988.

[36] B. Macukow and H. H. Arsenault, "Modification of the thresold condition for a content-addressable memory based on the Hopfield model," *Applied Optics*, vol. 26, no. 1, 34-36, 1987.

[37] Y. Baram, "Associative memory in fractal neural networks," *IEEE Trans. Sys. Man Cybernetics*, vol. 19, no. 5, 1133-41, 1989.

[38] K. Murakami and T. Aibara, "An improvement on the Moore-Penrose generalized inverse associative memory," *IEEE Trans. Sys. Man. Cybern.*, vol. SMC-17, no. 4, 699-707, 1987.

[39] G. S. Stiles and D. L. Denq, "A quantitative comparison of the performance of three discrete distributed associative memory models," *IEEE Trans. Computers*, vol. C-36, no. 3, 257-263, 1987.

[40] G. S. Stiles and D. L. Denq, "On the effect of noise on the Moore-Penrose generalized inverse associative memory," *IEEE Trans. Patt. Ana. Mach. Intell.*, vol. PAMI-7, no. 3, 358-360, 1985.

[41] K Murakami and T Aibara, "Least squares associative memory and a theoretical comparison of its performance," *IEEE Trans on Sys Man. Cybern*, vol 19, no 5, 1230-34, 1989.

[42] Y Kosugi, "Efficiency comparison of associative memories using BSC model," *IEEE Trans on Sys Man Cybern*, vol 19, no 5, 974-79, 1989.

[43] S S Venkatesh and D Psaltis, "Linear and logarithmic capacities in associative neural networks," *IEEE Trans Inf Theory*, vol 35, no 3, 558-568, 1989.

[44] Y C Ho and R L Kashyap, "An algorithm for linear inequalities and its application," *IEEE Trans Elec Comp*, vol EC-14, no 5, 683-88, 1965

[45] R O Duda and P E Hart, "Pattern Classification and Scene Analysis," John Wiley and Sons, New York, 1973.

[46] M H Hassoun, "Dynamic heteroassociative neural memories," *Neural Networks*, vol 2, no 4, 275-287, 1989

[47] M H Hassoun and A M Youssef, "High performance recording algorithm for Hopfield model associative memories," *Optical Engineering*, vol 28, 46-54, 1989

[48] H H Chen, Y C Lee, G Z Sun, H Y Lee, T Maxwell, and C L Giles, "High order correlational model for associative memory," AIP Conf Proc 151, *Neural Networks for Computing*, Snowbird, Utah, editor John S Denker, 86-99, 1986

[49] D Psaltis and C H Park, "Nonlinear discriminant functions and associative memories," AIP Conf Proc 151, *Neural Networks for Computing*, Snowbird, Utah, editor John S Denker, 370-375, 1986

[50] C L Giles and T Maxwell, "Learning, invariance, and genealization in high-order neural networks," *Applied Optics*, vol 26, no 23, 4972-78, 1987

[51] T D Chieuh and R M Goodman, "High-capacity exponential associative memories," *IEEE Intl Conf on Neural Networks*, vol I, 153-60, San Diego, CA, July 1988

[52] T Poggio, "On optimal nonlinear associative recall," *Biological Cybernetics*, vol 19, 201-209, 1975

[53] J W Goodman, A R Dias, and L M Woody, "Fully parallel high-speed incoherent optical method for performing discrete Fourier transforms," *Optics Letters*, vol 2, 1-3, 1978

[54] D Casasent and B V K Vijaya Kumar, "Optical linear algebra processors," Chapter 6 1 in *Optical Signal Processing*, J L Horner, editor, Academic Press, 389-407, 1987

[55] D Casasent, "Acousto-optic linear algebra processors: architectures, algorithms, and applications," *Proc of IEEE*, vol 72, no 7, 831-, 1984

[56] W Ross, D Psaltis, and R Anderson, "Two-dimensional magneto-optic spatial light modulator," *Optical Engineering*, vol 22, 485-490, 1983

[57] D A Gregory, R D Juday, J B Sampsell, R O Gale, R W Cohn, and S E Monroe, Jr , "Optical characteristics of a deformable mirror spatial light modulator," *Optics Letters*, vol 13, 10-12, 1988

[58] R Athale, C Friedlander, and B. Kushner, "Attentive associative architectures and their implementations," *Proc of SPIE*, vol 625, 179-188, 1986

[59] J S Jang, S Y. Shin, and S Y Lee, "Adaptive two-dimensional quadratic associative memory using holographics lenslet arrays," Technical Digest, OSA Topical meeting on *Optical Computing*, Salt Lake City, Utah, 40-43, 1989

[60] A Von Lehmeu, E G Paek, L C Carrion, J K Patel, and A. Marrakchi, "Optoelectronic chip implementation of a quadratic associative memory," *Optics Letters*, vol 15, no 5, 279-281, 1990

[61] A Vanderlugt, "Signal detection by complex spatial filtering," *IEEE Trans Inform Theory*, vol IT-10, 139-145, 1964

[62] D Gabor, "A new microscope principle," *Nature*, vol 161, 777-, 1948

[63] J W Goodman, *Introduction to Fourier Optics*, McGraw-Hill, New York, 1968

[64] H J Caulfield and S Lu, *The applications of holography*, Wiley Interscience, New York, 1970

[65] D Z Anderson and D M Lininger, "Dynamic optical interconnects: volume holograms as optical two-port operators," *Applied Optics*, vol 26, 5031-5038, 1987.

[66] K Wagner and D Psaltis, "Multilayer optical learning networks," *Applied Optics*, vol 26, 5061-5076, 1987

[67] D Psaltis, J Yu, X G Gu, and H Lee, "Optical neural nets implemented with volume holograms," Technical Digest, OSA Topical meeting on *Optical Computing*, Incline Village, Nevada, 129-132, 1987

[68] D Z Anderson, "Material demands for optical neural networks," *MRS Bulletin*, 30-35, August 1988.

[69] T Hall, , "The photorefractive effect - A review," *Prog Quant Electr* , vol 10, 77-, 1985

[70] J Wilde and L Hesselink, "Implementation of dynamic Hopfield-like networks using photorefractive crystals," Technical Digest, OSA Topical meeting on *Optical Computing*, Salt Lake City, Utah, 40-43, 1989

[71] P K Wong, *A Comparison of the Storage and Retrieval Performance Among Four Autoassociative Memory Neural Networks*, M S project report, Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, May 1990

[72] A Agranat, C F Neugebauer, and A Yariv, "Parallel optelectronics realization of neural network models using CID Technology," *Applied Optics*, vol. 27, 4354-4355, 1988

[73] Y Owechko, "Optoelectronic resonator neural networks," *Applied Optics*, vol 26, 5104-5111, 1987

[74] D Psaltis, J Hong, and S. Venkatesh, "Shift invaliance in optical associative memoiies," *Proc of SPIE*, vol. 625, , 1986.

[75] S H Song and S S Lee, "Pioperties of hologiaphic associative memoiy piepaied by the polaiization encoding piocess," *Applied Optics*, vol 27, 3149-54, 1988

[76] D Psaltis, D Biady, and K Wagnei, "Adaptive optical neuial netwoiks using photoiefiactive ciystals," *Applied Optics*, vol 27, 1752-1759, 1988

[77] M Oita, J Ohta, S Tai, and K Kyuma, "Optical implementation of laige-scale neuial netowiks using a time-division-multiplexing technique," *Optics Letteis*, vol 15, no 4, 227-229, 1990

[78] C Peteison, S Redfield, J K Keelei, and E. Haitman, "Optoelectionic implementation of multilayei neuial netwoiks in a single photoiefiactive ciystal," *Optical Engineering*, vol 29, no 4, 359-368, 1990

[79] D Z Andeison and M C Eiie, "Resonatoi memoiies and optical novelty filteis," *Optical Engineeiing*, vol 26, no 5, 434-444, 1987

Artificial Neural
Old and New Co
I.K. Sethi and A.
© 1991 Elsevier

ARTIFICI

Fathi M. A. Sa

Systems and C
Engineering, M

1. INTRODU

Models fo
[25] in 1943. A
the publication
works of Rume
given the field a
ers of the pote
efforts have led
of feedback ne
analytical metho
rigor [27-29, 36
forward and of
be an obstacle t
scale will be the
works would be

The crux o
methods introdu
Conversely, mod
medium, ought
quently, one sho
systems theory a
in order to deve
mimic the neural

From an en
software and/or
the mechanisms
suitably adapted,
of implementatio

In hardware
tronics or silicon
media known und
ics, and specific
integrated circuit
VLSI/LSI implem
many other intere

The interest
work of Hopfield
applicability of fe

Artificial Neural Networks and Statistical Pattern Recognition
Old and New Connections
I.K. Sethi and A.K Jain (Editors)
© 1991 Elsevier Science Publishers B.V All rights reserved

ARTIFICIAL NEURAL NETS IN MOS SILICON

Fathi M. A. Salam, Myung-Ryul Choi and Yiwen Wang

Systems and Circuits & Artificial Neural Nets Laboratories, Department of Electrical Engineering, Michigan State University, East Lansing, MI 48824

1. INTRODUCTION

Models for neural nets have been proposed since the work of McCulloch and Pitts [25] in 1943. A new turning point of modeling of neural nets came in the early 80's with the publication of the works of Hopfield and his coworkers [15, 16, 17, 18, 19] and the works of Rumelhart et al [30]. These works evolved from earlier works [1, 8], but have given the field a broad exposure and motivated (or convinced) new (engineering) researchers of the potential and feasibility of *Artificial Neural Nets* (ANNs). The cumulative efforts have led to the formulation of a basic configuration of *a class* of feedforward and of feedback neural nets [1, 5, 8, 15-19, 30] with certain interesting properties. Various analytical methods have been reported in the literature with a high degree of mathematical rigor [27-29, 36, 38, 41, 44]. These particular analyses have led to proper designs of feedforward and of feedback neural nets. Yet implementation in large scale have continued to be an obstacle to the practical utility of ANNs. It is believed that implementation in large scale will be the vehicle through which the computational powers attributed to neural networks would be revealed.

The crux of the point is that it is essential to ensure that the models and analytical methods introduced for neural networks are suitable for the medium of implementation. Conversely, models which lend themselves naturally to implementation via some chosen medium, ought to have sound analytical support for their dynamic properties. Consequently, one should merge and adopt discoveries and tools from neuro-biology, dynamical systems theory and mathematics, computer science and engineering, physics, and the likes, in order to develop information/image processing machines inspired by neurobiology that mimic the neural net capability in *real time*.

From an engineering prospective, implementation may be viewed in two ways: via software and/or via hardware. In our view hardware implementation is superior because the mechanisms driving (models of) neural networks would then be directly realized, and suitably adapted, into parallel large scale integrated hardware. Consequently, the medium of implementation would be exploited to its optimum limit.

In hardware implementations, there are now two obvious media: (i) optics, (ii) electronics or silicon. In addition, one may consider any appropriate combination of the two media known under the name electro-optics. Our implementation focus here is on electronics, and specifically on the well-developed and standard technology of MOS silicon integrated circuits We shall restrict our discussions to particular digital and/or analog VLSI/LSI implementations. In doing so, we realize that we will not, and cannot cover many other interesting and promising implementations.

The interest in hardware implementation has been revived, to a large degree, by the work of Hopfield and his coworkers [15-19]. In [19], Hopfield and Tank proposed the applicability of feedback ANNs to the design of A/D converters, signal decision circuits, and linear programming circuits. The ANN is realized as a number of operational amplifiers connected via variable (linear) resistors. The input of each operational amplifier is connected via a resistor to the non-inverting (inverting) output of another operational amplifier if the latter amplifier is excitatory (inhibitory). Consequently, for each neuron, there are non-inverting and inverting amplifiers. Also, in general, the number of variable (programmable) resistors (referred to as the synaptic weights) is equal to the square of the number of neurons. That is, for an n–neuron interconnected ANN there are $2n$ amplifiers and n^2 variable resistors or potentiometers.

It has been emphasized that large number of neurons are needed in an ANN to produce *emergent* useful computations in the neural network sense. Since then, many workers have sought to implement the feedback (and the feedforward) ANNs using VLSI/LSI silicon electronics. The input-output function of a neuron, which is usually a (differentiable) monotone sigmoid function, can be easily implemented in VLSI/LSI by an amplifier or by two logical inverters in series. A major obstacle, however, has been the VLSI/LSI implementation of the variable linear resistors (or artificial synapses).

ANN hardware based on the silicon MOS VLSI/LSI technology is actively being pursued by several research groups. Hubbard *et al.* [20] demonstrated a thin film synaptic array in submicron feature size fabricated by e-beam lithography. Selectively deposited amorphous silicon resistive elements at the nodes provide the resistive array synaptic connections which will be useful as an associative read-only memory. Sivilotti *et al.* [48] had fabricated a programmable neural network chip with twenty-two neurons and +1, 0, and -1 synapses. This circuit is designed to perform on-chip learning by employing a truncated version of Hebb's rule. Pursuing a different approach, Mead *et al.* [26] have integrated sensor arrays and processing elements to emulate some of the spatial and temporal properties of neural networks in the retina of the (human) eye. Researchers at Bell Labs [11] have fabricated a 54-neuron CMOS chip with programmable +1, 0, and -1 synapses. Such an implementation uses static RAM cells as prespecified memories. The desired data are stored as the synaptic weights of the rows of the interconnect (matrix). The circuit will perform the inner product of the incoming data and the rows of the synaptic weight matrix. The resulting inner products are passed through respective sigmoidal nonlinearities. The resulting largest output indicates that the incoming data is closest to the desired vector stored in the corresponding row of the synaptic weights. (This is a winner-take-all circuit.) Sage *et al.* [28] at MIT Lincoln Laboratory developed chips based on Metal-Nitride-Oxide-Semiconductor (MNOS) and Charge-Coupled Devices (CCDs) technology. Such an implementation would achieve analog synaptic weights via variable charge storage. Recently, one VLSI chip from Intel [14] was implemented with fully analog circuitry operating in a deterministic manner. Since digital circuits have proven themselves in standard VLSI, there is another chip [58] which was implemented with fully digital circuitry operating in a stochastic manner. Learning, however, is delegated to off-chip algorithms in the last two chips.

Here we describe our prospective regarding some research activities on the silicon implementation of ANNs. Our prospective has transpired from our extensive review of reports, papers, and products pertaining to aspects of neural theory, analysis, implementation, and developments. Based on our extensive review of the literature as well as our own activities, we delineate some observations in the following remarks:

(1) There are still some problems pertinent to silicon VLSI implementation of ANNs. For instance, in order to retain robustness to imperfections, to noise, and to temperature effects, the (analyzed) neural models must be faithfully mapped onto the VLSI electronic chips. If models are not (or cannot) be mapped faithfully, on the other hand, then it is essential that the introduced modifications are consistent with the formulation and the "spirit" of the analytically verified neural models. Such consistency would ensure the robustness and preservation of the functionality and the properties attributed to neural networks.

(2) Most of the existing implementations have not faithfully mapped the neural models into the domain of implementation, be it digital, analog, or hybrid silicon electronic circuits. Some researchers also appear to minimize or totally ignore the serious destabilizing and damaging effects of some introduced modifications. More often than not, the final designed product *does not* and *cannot* exhibit neural-like behaviors.
This, we believe, will have negative consequences. It would generate wrong and sweeping conclusions about the whole area of neural nets and its implementations Many, who may be unfamiliar with the dynamics of neural nets, will attribute the limitations and the shortcomings of the (unjustified and/or unfaithful) implementations to limitations and shortcomings of the theory of neural nets.

(3) The above two remarks are of a serious nature when one considers dynamic (nonlinear) neural net models (and all the silicon VLSI implementations are dynamic due to the ever-present parasitic capacitances). The statements are even more serious when one includes learning algorithms into these implementations.

(4) It is appealing, but also necessary, to build neural modules which will be compatible and may be integrated into a functioning system. However, because of the presence of feedback (which occurs in the structure in feedback neural nets and via the learning in feedforward neural nets), systems issues of stability, convergence, and robustness will be dominant.

(5) One may suggest models without any learning. We believe that such suggestion would not lead us too far. We believe, neural network research is unique because, among other things, it introduced dynamic changes to data or because it incorporates learning in the process. Learning thus is an essential attribute of the area of neural net which ought to be emphasized. It is the one subject that is needed the most, yet it is understood the least.

Our own research has focused on developing basic cells in all-MOS analog subcircuits as building blocks for the analog silicon implementation of ANNs. Some of our published works has adopted a systems-level approach whereby modules are built to be integrated into a neural network system, e.g., see [59]. We have focused on all-MOS design of vector-vector multipliers as basic cells, see e.g. [35] and Section 3. We have designed layouts for all-MOS feedforward ANNs with learning capability based on the vector analog multiplier cell [43]

The chapter is organized as follows. Section 2 presents a digital CMOS VLSI implementation of a 4-neuron circuit. We use the static RAM cells to store the synaptic weights and off-chip learning rules on a host (Personal) computer. This section describes our first experiments with implementation which enhanced our understanding and suggested the necessity of the *analog* pads as opposed to the conventional *digital* pads common in digital circuits. Section 3 presents our implementation of ANNs employing the basic cell of the all-MOS analog vector-vector multiplier. We have successfully demonstrated the implementation of our modified learning rule [44, 47] which is suitable for the silicon implementation. Section 4 presents our focus on the natural building block of the CMOS silicon technology, namely, the MOS Transistor. We use the (*nonlinear*) four-terminal transistor as the building block or the basic analog cell for constructing artificial neural circuits. The main neural circuit was proposed and constructed based on solid analytical base [33, 37, 42] while other implementations are demonstrated via SPICE simulations of prototype circuits [59]. Due to space limitations, we focus on the latter implementations and refer the reader to the literature and Section 4 for the former implementations. Section 5 summarizes our conclusions.

2. A DIGITAL CMOS VLSI IMPLEMENTATION OF A 4-NEURON CIRCUIT

A Hopfield-type neural network has been implemented and fabricated using standard CMOS VLSI technology with MOSIS 2.0 μm, n-well process in a 40-pin TINYCHIP. This chip is implemented with the mixture of the analog and digital VLSI technology. A neuron is implemented with two CMOS push-pull inverters in series and a connection is implemented with a CMOS pass-transistor, which is controlled by its own T_{ij} cell. T_{ij} cells are implemented by using static RAMs (SRAM). Learning is performed by a host computer. After learning, synapse weights are loaded through input pins and stored at T_{ij} cells with -1, 0, and +1 correspond to the inhibitory, open, and excitatory operations, respectively. We have used various learning rules [45], including the Hebbian rule.

2.1. Artificial Electronic Neural Circuit

A silicon implementation of the neural network proposed by Hopfield is described. An artificial neural circuit has been implemented using standard CMOS VLSI technology. A block diagram of an artificial neural circuit is shown in Figure 2.1. This contains 4 subblocks: control circuits and a column decoder, amplifiers, T_{ij} cells, and a row decoder. The control circuits and a column decoder subblock is used to store a synapse weight in a T_{ij} cell, to initialize an input of a neuron, and to read the output of a neuron after convergence. This subblock is very crucial to implement the neural circuit since the large number of neurons in a typical neural circuit usually exceeds the number of pins on a standard chip.

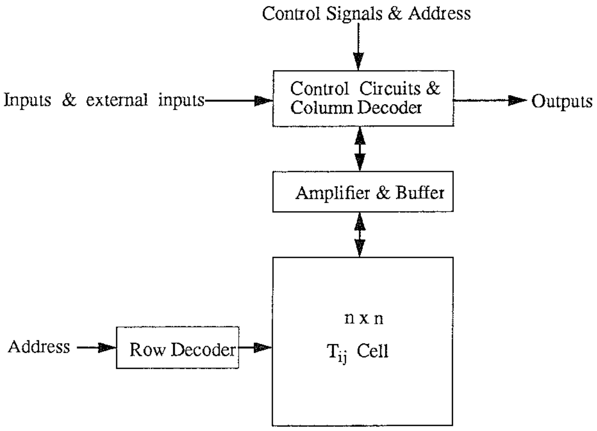


Figure 2.1 The block diagram of a neural circuit

For example, if a neural circuit is designed with 32 neurons, then more than 20 pins can be saved by using 4 I/O pins with one 3x8 column decoder instead of 32 I/O pins. The T_{ij} cell subblock consists of CMOS analog pass transistors and $2n^2$ SRAM cells for an n-neuron circuit. The row decoder part has k address inputs and one k x 2^k row decoder for n-neuron circuit, where $2^k \geq 2n$ and the decoder is used to select the corresponding row T_{ij}-cell. An amplifier is used to function as a cell body and is implemented with two CMOS inverters in series. In the next subsections, the implementation of a 4-neuron circuit is discussed in detail.

2.2. Implementation of a 4-Neuron Circuit

An artificial electronic neural circuit with 4 neurons is designed with MAGIC (University of California at Berkeley) VLSI tools and is fabricated by MOSIS (University of Southern California) 2 0 microns, n-well process in a 40-pin TINYCHIP padframe 34 pins out of 40 pins are used in order to test this chip more conveniently after fabrication; It should be mentioned that 23 pins suffice for our circuit design, however. Figure 2.2 shows the 40-pin TINYCHIP padframe including the design project; there are approximately 630 transistors in this project

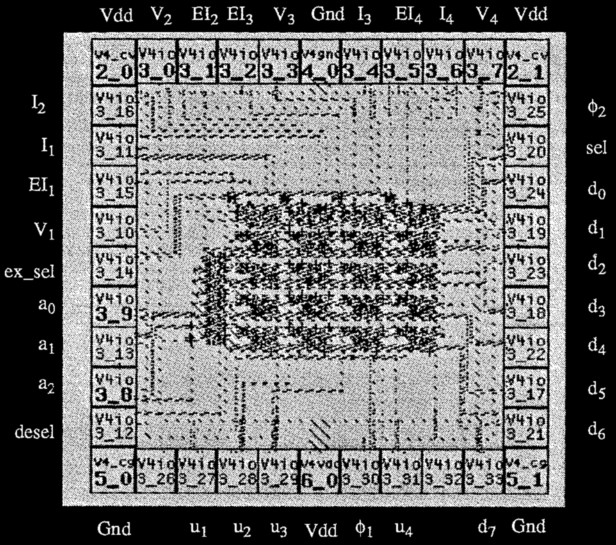


Figure 2 2 The 4-neuron circuit on a 40-pin TINYCHIP padframe

2.3. The Architecture of the 4-Neuron Circuit

The architecture of the 4-neuron circuit contains control circuits, 4 amplifiers, one 3x8 row decoder, and 16 programmable T_{ij} cells. Each amplifier consists of two push-pull inverters connected in series, which are used for excitatory and inhibitory processing and each T_{ij} cell is composed of 2 static RAMs (SRAMs) and CMOS analog pass transistors. A 3x8 row decoder is used to select one column of 4 T_{ij} cell columns and each T_{ij} cell column consists of two columns, one for excitatory connection storage and the other for inhibitory one (See Table 2.1). For example, when $(a_2\ a_1\ a_0) = (1\ 0\ 0)$, the input vector $(I_1\ I_2\ I_3\ I_4)$ is stored in the T_{ij} cell column, $(T_{13}\ T_{23}\ T_{33}\ T_{43})$.

Learning is performed by a host computer. After learning, a host computer stores the modified synapse weights into the corresponding T_{ij} cells with three different interconnection weights (-1, 0, +1), where -1 corresponds to inhibitory processing, +1 to excitatory processing and 0 to an open connection. Through these T_{ij} cells, the output of an amplifier can be connected to the input of any amplifier in the circuit. Figure 2.3 shows the connection between the output of the amplifier j and the input of the amplifier i with a pass transistor and SRAMs. Here the stored datum in each T_{ij} cell is applied to the gate of the pass transistor to allow proper connection between the output of the amplifier j and the input of the amplifier i, where the pass transistor works as a resistive device between

Table 2.1 Selection of 3x8 Row Decoder

a_2	a_1	a_0	I_1	I_2	I_3	I_4
0	0	0	T_{11}	T_{21}	T_{31}	T_{41}
0	0	1	$-T_{11}$	$-T_{21}$	$-T_{31}$	$-T_{41}$
0	1	0	T_{12}	T_{22}	T_{32}	T_{42}
0	1	1	$-T_{12}$	$-T_{22}$	$-T_{32}$	$-T_{42}$
1	0	0	T_{13}	T_{23}	T_{33}	T_{43}
1	0	1	$-T_{13}$	$-T_{23}$	$-T_{33}$	$-T_{43}$
1	1	0	T_{14}	T_{24}	T_{34}	T_{44}
1	1	1	$-T_{14}$	$-T_{24}$	$-T_{34}$	$-T_{44}$

two amplifiers. In case of excitatory connection, $T_{ij} = 1$, the noninverting output of the amplifier j, V_j, is connected to the input of the amplifier i, u_i, for inhibitory processing, $T_{ij} = -1$, the inverting output of the amplifier j, $-V_j$, is connected to the input of the amplifier i, u_i, and for open connection, $T_{ij} = 0$, the output of the amplifier j is not connected to the input of the amplifier i.

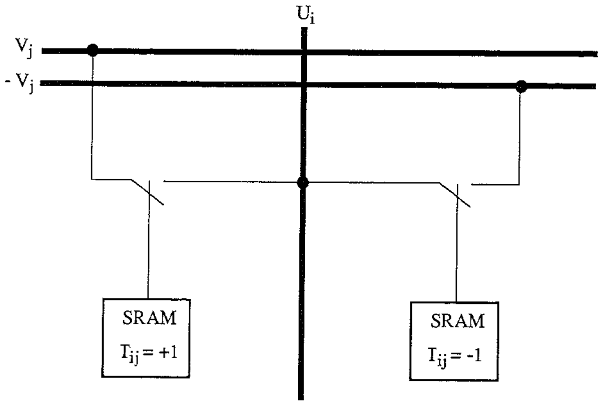


Figure 2.3 A SRAM cell interconnect (T_{ij})

The voltage of an input line of an amplifier is determined by the sum of the currents flowing into the line, which is zero here since the input of each amplifier circuit is a gate of a transistor.

A threshold value is adjusted by choosing an aspect ratio (W/L) of the nMOS and the pMOS transistors of the inverters since the input/output characteristic of an inverter is a function of the aspect ratio of MOS transistors. Here 3.0/3.0 and 4.0/2.0 are chosen for the aspect ratio of nMOS transistors and the pMOS transistors, respectively.

2.4. SPICE Simulation and Chip Test

SPICE is used to simulate this 4-neuron circuit and the results for 2-neuron, 3-neuron, and 4-neuron circuits are shown in Table 2.2. Table 2.2 shows that these neural circuits act like majority-vote when the T_{ij} cells are set to 1 for all the off-diagonal terms and 0 for the diagonal terms. In this simulation, this 4-neuron circuit converges to logical value (0000), when it is initialized with two 5 volts and two zero volts. This is due to the threshold voltage of the double inverter, which is 2.65 volts in this design. If a different aspect ratio is chosen, then a different threshold value will be obtained since this threshold voltage is a function of the aspect ratio. For example, if the threshold value is chosen to be less than 2.5 volts, then the circuit will converge to logical value (1111) when initialized with only two logical high.

The fabricated chip has been tested with the TOPAZ VLSI digital testing system and its results were compared to the SPICE simulation. First, this chip is tested with each sub-block (Figure 2.1): the amplifier part, the external input part, and the 3x8 decoder part. The test of the amplifier part results in 0 output for 0 input and 1 output for 1 input. This shows that the amplifier part works correctly. The test of the external input part is done by measuring the input of a neuron by applying external inputs. This part works correctly since the expected output values are measured. The 3 x 8 decoder part has been tested by measuring the outputs of the decoder by applying all the possible addresses This part works correctly also since the corresponding output of the decoder is 1 and all the other outputs are 0.

Finally, the whole circuit is tested by applying test vectors. Table 2.2 shows the results of the chip test and those of the SPICE simulation. This shows that the expected outputs can not be measured by applying test vectors. There are three possible situations where this chip might generate errors. One possible source of error is that the SRAM may not function properly by either not storing the synapse weight or failing to supply the weight to turn on the pass-transistor connection. However, the SPICE simulations, based on the parameters provided by MOSIS, verify that the well-understood SRAM operation cannot be the source of errors. Eventually, after numerous tests, we concluded that the source of error may be due to the use of conventional digital pads for the fabrication of, in reality, an analog neural circuit. We note that digital pads consist of protection and buffer circuitry. When initialized (then disconnected), the buffer circuitry retains the initial voltages and interacts with the neural circuit to generate transient dynamics of the coupling which may differ from the transient dynamics of the neural circuit alone. This finding had prompted us to use analog pads in all of our subsequent designs.

2.5. Discussion

A 4-neuron circuit has been simulated, implemented, and tested, in order to understand how a neural circuit works, what kind of relationship exists between computing elements and conductance elements, and what sizes of transistors are more suitable. This understanding has encouraged us to pursue and propose larger designs which would include analog pads.

3. ANNs USING ANALOG VECTOR MULTIPLIER BASIC CELLS

A crucial element in implementing the available models of ANNs in silicon hardware is the implementation of the programmable (*linear*) synaptic weights. In addition to implementing synaptic weights, vector-vector multipliers are also useful in realizing components of certain learning algorithms. We describe a successful implementation based on analog vector-vector multipliers. We first describe the design of the vector-vector multiplier as a basic cell. Then we incorporate the basic cell multiplier in the implementation of feedforward ANNs with learning.

Table 2.2 Chip Test using TOPAZ and Circuit Simulation using SPICE (unit: hex number and 0(7)0(7) means that there are two possible states, i.e., 00 or 77.)

a) 2-Neuron circuit with $T = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$

Initial Condition	TOPAZ		SPICE	
$i_1 i_0$	$u_1 u_0$	$v_1 v_0$	$u_1 u_0$	$v_1 v_0$
0	0	0	0	0
1	0	0	1	1
2	0	0	2	2
3	0	0	0	0

b) 3-Neuron circuit with $T = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$

Initial Condition	TOPAZ		SPICE	
$i_2 i_1 i_0$	$u_2 u_1 u_0$	$v_2 v_1 v_0$	$u_2 u_1 u_0$	$v_2 v_1 v_0$
0	0	0	0	0
1	0(7)	0(7)	0	0
2	0	0	0	0
3	7(0)	7(0)	7	7
4	0(7)	0(7)	0	0
5	0(7)	0(7)	7	7
6	0(7)	0(7)	7	7
7	7	7	7	7

c) 4-Neuron circuit with $T = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$

Initial Condition	TOPAZ		SPICE	
$i_3 i_2 i_1 i_0$	$u_3 u_2 u_1 u_0$	$v_3 v_2 v_1 v_0$	$u_3 u_2 u_1 u_0$	$v_3 v_2 v_1 v_0$
0	0(F)	0(F)	0	0
1	0(F)	0(F)	0	0
2	0	0	0	0
3	F(0)	F(0)	0	0
4	F(0)	F(0)	0	0
5	F	F	0	0
6	0	0	0	0
7	F	F	F	F
8	0	0	0	0
9	0(F)	0(F)	0	0
A	F(0)	F(0)	0	0
B	F	F	F	F
C	F(0)	F(0)	0	0
D	F	F	F	F
E	F	F	F	F
F	F	F	F	F

3.1. A Simple Tunable Analog All-MOS Vector Multiplier

A simple tunable analog all-MOS vector-vector multiplier is implemented [32, 35] for an artificial neural network with learning [43]. This multiplier circuit is suitable to implement a synapse in the neural nets since it is simple and can execute a linear multiplication with small error. The multiplier is depicted in Figure 3.1 and its output is represented by [35, 43]

$$V_m = \frac{(W/L)_i}{(W/L)_r(V_{c_1} - V_{c_2})} \sum_{i=1}^{n} (W_1^i - W_2^i)(X_1^i - X_2^i), \tag{3.1}$$

with the following operating constraints

$$\begin{aligned} &X_i{}^k - W_j{}^k \geq V_T > 0 \qquad \text{for } all\ i,\ j = 1,2, \text{ and } k = 1,2,\ldots,n \\ &V_{c_i} - V_T \geq V_m \qquad \text{for } all\ i = 1,2. \end{aligned} \tag{3.2}$$

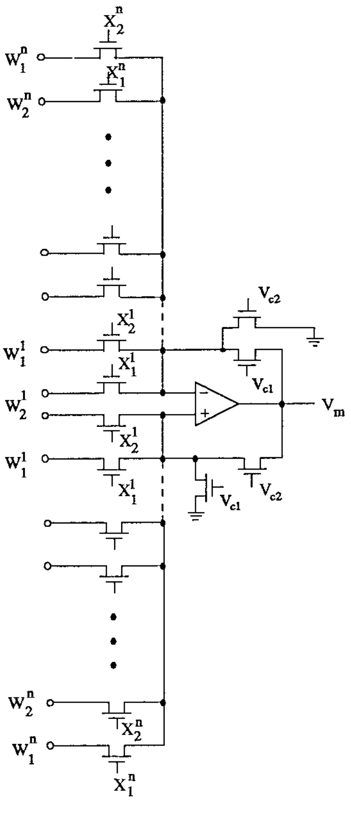


Figure 3.1 All-MOS n-D analog multiplier

To satisfy the operating constraints (3.2), the output range of the multiplier would not match the operating input range. We have designed the multiplier to operate in a small operating range in order to reduce errors caused by the variation of the mobility. The operating range of the multiplier is required to be compatible with the operating range of a neuron in order to use the multiplier as an artificial synapse.

There are two problems to be considered for this multiplier in order to implement it as a synapse in ANNs: One is the multiplier's driving input to the drain of a floating MOS transistor, whose input impedance is low; the other is the operating range of the multiplier and its input-output compatibility. The first problem is solved by employing a voltage follower. In order to make the operating input range match the output range of the multiplier and/or increase the operating input range of the multiplier, voltage shifters/attenuators are employed.

The vector multiplier basic cell, consequently, is designed with additional circuitry that include voltage followers and voltage shifters/attenuators, which are depicted in Figure 3.2.

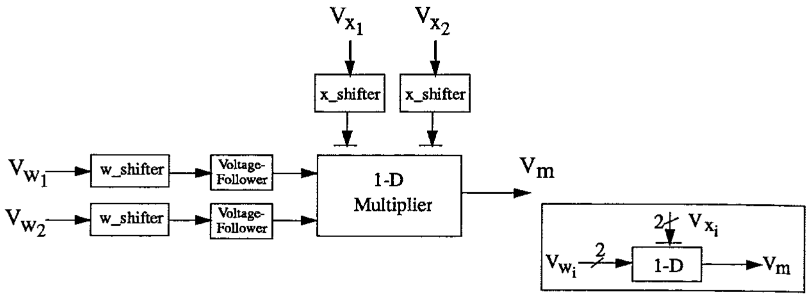


Figure 3.2 A 1-D multiplier cell

Two simple level shifter/attenuator circuits are designed: one is for the w_i input, named w_shifter, and the other is for the x_i input, named x_shifter. The w_shifter is designed using four MOS transistors with one bias voltage and the x_shifter is designed using four MOS transistors. These circuits are simulated using the PSPICE circuit simulator and the results show that these level shifters/attenuators have sufficiently small errors.

3.2. MOS Programmable Feedforward ANN Circuits

3.2.1. A Modified Learning Rule for Feedforward ANNs

A modified learning rule [44], which removes all of the sigmoidal derivative functions, is realized with circuits using MOS multiplier basic cells. The circuit is simulated using the PSPICE circuit simulator for a prototype two-layer feedforward ANN with learning using standard CMOS VLSI/LSI technology.

Two learning circuits are introduced for the implementation of the modified learning rule. Each circuit is described in a block diagram and its PSPICE simulation results are summarized. In order to save time and render executable simulations, PSPICE simple models as well as some detailed models are employed for some of the amplifiers and for some of the level shifters. The complexity and size of the overall ANN circuit model rendered the PSPICE simulator inefficient to execute when the complete detailed models of all components are included.

The governing static equation of each neuron in a feedforward ANN is expressed as [30]:

$$\bar{y}_j = S_j(\sum_{i=1}^{N} \bar{w}_{ji} \underline{y}_i + \bar{\theta}_j). \tag{3.3}$$

Here $\bar{y}_j$ is the output of the j-th neuron in the present layer, $\bar{w}_{ji}$ is the synaptic weight connecting the output $\underline{y}_i$ of the i-th neuron from the previous layer to the input of the j-th neuron in the present layer, $\bar{\theta}_j$ is the bias weight connected to the j-th neuron in the present layer, and S_j is the sigmoidal function.

The error function is defined as usual as

$$E_p = \frac{1}{2} \sum_{j=1}^{n} (t_{pj} - y_{pj})^2, \tag{3.4}$$

where t_{pj} is the value of the desired target p for the j-th output component and y_{pj} is the j-th output of the final layer when the external input associated with the target p is applied.

The modified learning rule, given in [43, 44, 47], is expressed for any weight w_{ji} as:

$$\dot{w}_{ji} = -\eta \, (\frac{\partial S_j}{\partial u_j})^{-1} \sum_p \frac{\partial E_p}{\partial w_{ji}} - \mu_{ji} w_{ji} = \eta \sum_p e_{pj} \, \underline{y}_{pi} - \mu_{ji} w_{ji}, \tag{3.5i}$$

where if the neuron j is in the output layer, then

$$e_{pj} = t_{pj} - \bar{y}_{pj} \tag{3.5ii}$$

and if the neuron j is in any hidden layer, then

$$e_{pj} = \sum_k \frac{d\bar{S}_k}{d\bar{u}_k} e_{pk} \, \bar{w}_{kj}, \tag{3.5iii}$$

where k is the index for the elements in the immediate subsequent layer, $\bar{S}_k$ is the nondecreasing differentiable monotone sigmoid function of neuron k in the immediately subsequent layer. It is also shown that the derivative terms ($d\bar{S}_k/d\bar{u}_k$ in (3.5iii)) may be removed without loss of stability and convergence of the update (learning) law (3.5).

3.2.2. The Sequential-Learning Circuit for Feedforward ANNs

Two different sequential-learning circuit implementations are introduced for two-layer feedforward ANNs. We discuss these circuits focusing on PSPICE simulation results of prototype ANN circuits.

The Sequential-Learning ANN Circuit #1

The sequential-learning ANN circuit #1 is discussed in the context of a two-layer prototype, see Figure 3.3. The realized prototype circuit includes a feedforward static model and a dynamic learning circuitry implementing the modified learning rule. The circuit implementation of the static model results in the governing equations

$$\underline{Y}_{p1} = S_n(k_2(\sum_j \underline{w}_{1j} x_{pj})) \tag{3.6i}$$

$$\underline{Y}_{p2} = S_n(k_2(\sum_j \underline{w}_{2j} x_{pj})) \tag{3.6ii}$$

$$\bar{Y}_p = S_n(k_2(\sum_j \bar{w}_{1j} \underline{y}_{pj})), \tag{3.6iii}$$

where $S_n()$ is a sigmoid function of a neuron and k_l is a constant associated with an l-dimensional vector-multiplier basic cell. In (3.6), $\underline{y}_{pj}$ and x_{pj} are defined as follows:

$$\underline{y}_{pj} := \underline{Y}_{pj} - 2.5$$

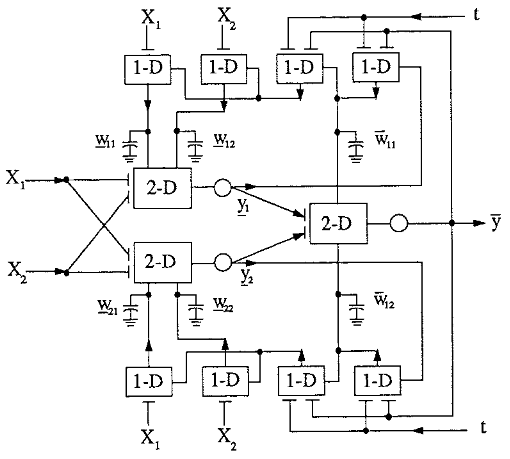


Figure 3.3 The feedforward ANN with the sequential learning circuit #1

$$x_{pj} := X_{pj} - 2.5,$$

where $\underline{Y}_{pj}$ is an actual output of the hidden layer, $\overline{Y}_p$ is an actual output of the output layer, and X_{pj} is an actual input of the input layer for pattern p. This notation is used throughout this chapter.

The implemented learning circuitry of the feedforward ANN is a realization of the dynamic update equations. The circuitry is governed by:

$$RC\dot{\overline{w}}_{11} = k_1 \, [\, (t_{11} - \overline{y}_{11}) \, \underline{y}_{11}] - \overline{w}_{11} \tag{3.7i}$$

$$RC\dot{\overline{w}}_{12} = k_1 \, [\, (t_{11} - \overline{y}_{11}) \, \underline{y}_{12}] - \overline{w}_{12} \tag{3.7ii}$$

$$RC\dot{\underline{w}}_{11} = k_1 \, \overline{w}_{11} \, [\, (t_{11} - \overline{y}_{11}) \, x_{11}] - \underline{w}_{11} \tag{3.8i}$$

$$RC\dot{\underline{w}}_{12} = k_1 \, \overline{w}_{11} \, [\, (t_{11} - \overline{y}_{11}) \, x_{12}] - \underline{w}_{12} \tag{3.8ii}$$

$$RC\dot{\underline{w}}_{21} = k_1 \, \overline{w}_{12} \, [\, (t_{11} - \overline{y}_{11}) \, x_{11}] - \underline{w}_{21} \tag{3.8iii}$$

$$RC\dot{\underline{w}}_{22} = k_1 \, \overline{w}_{12} \, [\, (t_{11} - \overline{y}_{11}) \, x_{12}] - \underline{w}_{22}. \tag{3.8iv}$$

This learning feedforward ANN circuit is simulated using PSPICE. Four learning tasks using four distinct input-output patterns are executed separately following this procedure:

1. For all the executed PSPICE simulation tasks, initialize the dynamic ANN circuit using the same initial condition. In the case reported here, we set ($\underline{w}_{11}$ $\underline{w}_{12}$ $\underline{w}_{21}$ $\underline{w}_{22}$ $\overline{w}_{11}$ $\overline{w}_{12}$) = (-0.5 0.6 -0.5 0.5 0.7 0.5).
2. Each input data and its desired target (i.e., the input-output pattern) are applied and PSPICE transient analysis is then executed. We then measure the steady state weight values and the output error, namely, $\tau - \overline{y}$.
3. PSPICE dc analysis is executed by setting the weights to the steady state weight values. Then, we again measure the output error $\tau - \overline{y}$.

The results, summarized in Table 3.1, show that this ANN circuit succeeds in learning each applied desired input-output pattern. When the four input-output pairs are applied sequentially, we found that the overall ANN circuit did not retain all prior input-output patterns which it learned. We also performed sequential PSPICE runs as follows: we initialize the ANN circuit for an input-output pattern which it would learn successfully by converging to a set of (equilibrium) weights. We then use the attained (equilibrium) weights as initial weights for the next input-output pattern. Continuing this process, until we employ all the input-output patterns and reapply the first pattern again. In this process, we found that the ANN circuit would learn each pattern that is presently applied, but it does not necessarily retain other input-output patterns that it has previously learned. The ANN circuit is also simulated using the PSPICE transient analysis by applying voltage waveforms for the external inputs X_1, X_2, and the desired τ, whose result concurs with the result in Table 3.1.

Table 3.1 The PSPICE transient analysis results of the feedforward ANN with the sequential learning circuit #1. The initial and steady state weights are tabulated for each pattern (units are in volts).

X_1 / X_2	0.5 / 0.5		0.5 / 4.5		4.5 / 0.5		4.5 / 4.5	
τ	1.0		4.5		4.5		1.0	
$\bar{y}$	1.81		5.0		5.0		1.81	
$\tau - \bar{y}$	-0.81		-0.5		-0.5		-0.81	
weights	init.	s.s.	init.	s.s.	init.	s.s.	init.	s.s.
$\underline{w}_{11}$	-0.5	0.053	-0.5	0.02	-0.5	-0.019	-0.5	-0.052
$\underline{w}_{12}$	0.6	0.053	0.6	-0.019	0.6	0.02	0.6	-0.052
$\underline{w}_{21}$	-0.5	0.053	-0.5	0.02	-0.5	-0.019	-0.5	-0.052
$\underline{w}_{22}$	0.5	0.053	0.5	-0.019	0.5	0.02	0.5	-0.052
$\bar{w}_{11}$	0.7	0.276	0.7	0.168	0.7	0.168	0.7	0.276
$\bar{w}_{12}$	0.5	0.276	0.5	0.168	0.5	0.168	0.5	0.276

s.s means steady state and init. means initial value.

The Sequential-Learning Circuit #2

The block diagram of the sequential-learning circuit #2 is depicted in Figure 3.4. This circuit incorporates bias weights and nonlinear terms in the weight equations, in addition to the circuitry in the sequential-learning circuit #1. The circuit implementation is governed by the following equations:

$$RC\ \dot{\bar{T}}_{11} = k_1\ (t_{11}-\bar{Y}_1)\ \underline{y}_{11} - \bar{T}_{11} \tag{3.9i}$$

$$RC\ \dot{\bar{T}}_{12} = k_1\ (t_{11}-\bar{Y}_1)\ \underline{y}_{12} - \bar{T}_{11} \tag{3.9ii}$$

$$RC\ \dot{\underline{T}}_{11} = k_1\ \bar{w}_{11}\ (t_{11}-\bar{Y}_1)x_{11} - \underline{T}_{11} \tag{3.10i}$$

$$RC\ \dot{\underline{T}}_{12} = k_1\ \bar{w}_{11}\ (t_{11}-\bar{Y}_1)x_{12} - \underline{T}_{12} \tag{3.10ii}$$

$$RC\ \dot{\underline{T}}_{21} = k_1\ \bar{w}_{12}\ (t_{11}-\bar{Y}_1)x_{11} - \underline{T}_{21} \tag{3.10iii}$$

$$RC\ \dot{\underline{T}}_{22} = k_1\ \bar{w}_{12}\ (t_{11}-\bar{Y}_1)x_{12} - \underline{T}_{22} \tag{3.10iv}$$

and

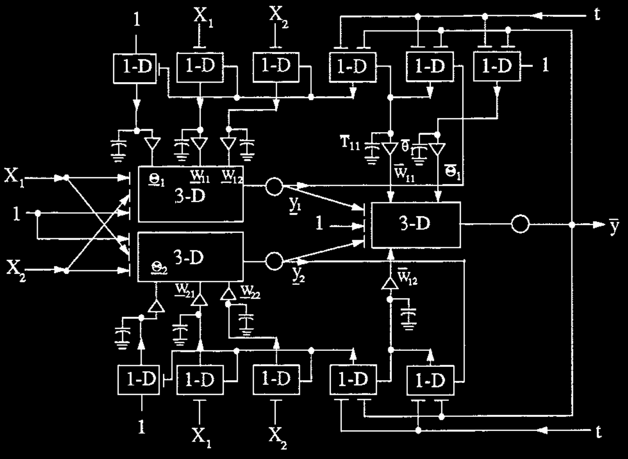


Figure 3.4 The feedforward ANN with the sequential learning circuit #2

$$w_{ij} = S(T_{ij}).$$

The (incorporated) modified learning circuits for (bias) weights are governed by:

$$\underline{\theta}_1 + RC\,\dot{\underline{\theta}}_1 = k_1\,\overline{w}_{11}\,(t_p - \overline{Y}_p)\,1 \qquad (3.11\text{i})$$

$$\underline{\theta}_2 + RC\,\dot{\underline{\theta}}_2 = k_1\,\overline{w}_{12}\,(t_p - \overline{Y}_p)\,1 \qquad (3.11\text{ii})$$

$$\overline{\theta}_1 + RC\,\dot{\overline{\theta}}_1 = k_1\,(t_p - \overline{Y}_p)\,1 \qquad (3.11\text{iii})$$

and

$$\Theta_i = S(\theta_i).$$

The circuit implementation of the governing equations results in

$$\underline{Y}_{p1} = S_n(\,k_3(\sum_j \underline{w}_{1j}x_{pj} + \underline{\Theta}_1)) \qquad (3.12\text{i})$$

$$\underline{Y}_{p2} = S_n(\,k_3(\sum_j \underline{w}_{2j}x_{pj} + \underline{\Theta}_2)) \qquad (3.12\text{ii})$$

$$\overline{Y}_p = S_n(\,k_3(\sum_j \overline{w}_{1j}\underline{y}_{pj} + \overline{\Theta}_1)), \qquad (3.12\text{iii})$$

where $S_n(.)$ represents a nonlinearity for Θ_i and w_{ij}.

This learning circuit is tested using the PSPICE circuit simulator. Learning tasks are again executed sequentially as follows (Tables 3.2 and 3.3 summarize the results):

1. Initialize the overall ANN circuit with the initial condition given at the top of each table.
2. The first input-output pattern is applied and the PSPICE transient analysis is executed. Then, measure the steady state (equilibrium) weight values.
3. Use this last steady state weight values as initial conditions and run the PSPICE transient analysis for the next input-output pattern.

Table 3.2 (a) The PSPICE transient analysis results of the feedforward ANN with the sequential learning circuit #2 (w/o amplification) (units are in volts). τ is a target and $\bar{y}$ is an actual output. The initial conditions are given by

$(\underline{T}_{11}(0)\ \underline{T}_{12}(0)\ \underline{T}_{21}(0)\ \underline{T}_{22}(0)\ \bar{T}_{11}(0)\ \bar{T}_{12}(0)\ \underline{\theta}_1(0)\ \underline{\theta}_2(0)\ \bar{\theta}_1(0)) = (0.5\ -0.5\ -0.5\ 0\ 5\ 0.5\ 0.5\ -0\ 5\ -0\ 5\ 0.5)$

$(\underline{w}_{11}(0)\ \underline{w}_{12}(0)\ \underline{w}_{21}(0)\ \underline{w}_{22}(0)\ \bar{w}_{11}(0)\ \bar{w}_{12}(0)\ \underline{\Theta}_1(0)\ \underline{\Theta}_2(0)\ \bar{\Theta}_1(0)) = (1\ 0\ -1\ 0\ -1\ 0\ 1\ 0\ 1.0\ 1\ 0\ -1\ 0\ -1.0\ 1.0).$

	Column #1	Column #2	Column #3	Column #4	Column #5
x_1	4 5	1.0	1.0	4.5	4.5
x_2	4 5	1.0	4.5	1.0	4.5
τ	0.5	0.5	4.5	4.5	0.5
$\bar{y}$	0.5002	0.4999	4.501	4.501	0.5002
$\tau-\bar{y}$	-0.002	0.001	-0.001	-0.001	-0.002
$\underline{T}_{11}$	4 210E-04	-2.511E-04	-2.901E-04	3 817E-04	4 565E-04
$\underline{T}_{12}$	4.210E-04	-2.511E-04	3.817E-04	-2 901E-04	4 565E-04
$\underline{T}_{21}$	4 210E-04	-2.511E-04	-2.901E-04	3.817E-04	4 565E-04
$\underline{T}_{22}$	4 210E-04	-2.511E-04	3.817E-04	-2.901E-04	4.565E-04
$\bar{T}_{11}$	3 410E-03	3.411E-03	3.410E-03	3.410E-03	3 410E-03
$\bar{T}_{12}$	3.410E-03	3 411E-03	3.410E-03	3.410E-03	3.410E-03
$\underline{\theta}_1$	4.210E-04	3 304E-04	3.817E-04	3.817E-04	4 565E-04
$\underline{\theta}_2$	4.210E-04	3 304E-04	3 817E-04	3 817E-04	4.565E-04
$\bar{\theta}_1$	-3.269E-03	2.739E-03	-2.723E-03	-2.723E-03	-3.771E-03
$\underline{w}_{11}$	-1	-1	-1	-1	-1
$\underline{w}_{12}$	-1	-1	-1	-1	-1
$\underline{w}_{21}$	-1	-1	-1	-1	-1
$\underline{w}_{22}$	-1	-1	-1	-1	-1
$\bar{w}_{11}$	-0.1393	0.1396	-0.1663	-0.1663	-0.1393
$\bar{w}_{12}$	-0.1393	0 1396	-0.1663	-0.1663	-0 1393
$\underline{\Theta}_1$	-1	-1	-1	-1	-1
$\underline{\Theta}_2$	-1	-1	-1	-1	-1
$\bar{\Theta}_1$	-1	-1	-1	-1	-1

Table 3.2(b) The PSPICE DC analysis results of the feedforward ANN circuit with the convergent weights in Table 3.2(a). τ is a target, y_i is an output of the hidden layer, and $\bar{y}$ is an output of the output layer.

			Column #1,5				Column #2				Column #3,4			
x_1	x_2	τ	y_1	y_2	$\bar{y}$	$\tau-\bar{y}$	y_1	y_2	$\bar{y}$	$\tau-\bar{y}$	y_1	y_2	$\bar{y}$	$\tau-\bar{y}$
1.0	1.0	0 5	5.0	5.0	0 0	0.5	5.0	5.0	0.4995	0.005	5.0	5 0	0.0	0.5
1 0	4.5	4 5	0 0	0 0	0 5002	3.998	0.0	0 0	0.0	4.5	0.0	0 0	4 501	-0.001
4 5	1 0	4 5	0.0	0.0	0.5002	3 998	0 0	0 0	0 0	4 5	0.0	0 0	4 501	-0 001
4 5	4.5	0.5	0 0	0 0	0.5002	-0.002	0.0	0 0	0.0	0.5	0.0	0.0	4 501	-4 001

Table 3.3(a) The PSPICE transient analysis results of the feedforward ANN with the sequential learning circuit #2 (w/o amplification) (units are in volts) τ is a target and $\bar{y}$ is an actual output. The initial conditions are given by

($\underline{T}_{11}(0)$ $\underline{T}_{12}(0)$ $\underline{T}_{21}(0)$ $\underline{T}_{22}(0)$ $\bar{T}_{11}(0)$ $\bar{T}_{12}(0)$ $\underline{\theta}_1(0)$ $\underline{\theta}_2(0)$ $\bar{\theta}_1(0)$) = (0 0 0.0 0 0 0.0 0 0 0 0 0.0 0 0 0 0)

($\underline{w}_{11}(0)$ $\underline{w}_{12}(0)$ $\underline{w}_{21}(0)$ $\underline{w}_{22}(0)$ $\bar{w}_{11}(0)$ $\bar{w}_{12}(0)$ $\underline{\Theta}_1(0)$ $\underline{\Theta}_2(0)$ $\bar{\Theta}_1(0)$) = (1 0 1.0 1.0 1.0 1 0 1 0 1.0 1 0 1 0)

	Column #1	Column #2	Column #3	Column #4	Column #5
x_1	4 5	1 0	1 0	4.5	4.5
x_2	4.5	1 0	4 5	1.0	4 5
τ	0.5	0.5	4.5	4.5	0.5
$\bar{y}$	0 5099	0 4928	4.51	4.51	0.509
$\tau-\bar{y}$	-0.099	0.0072	-0.01	-0.01	-0.099
$\underline{T}_{11}$	4 607E-04	-2 507E-04	-2.85E-04	3 284E-04	4.535E-04
$\underline{T}_{12}$	4 607E-04	-2 507E-04	3 751E-04	-2 906E-04	4 535E-04
$\underline{T}_{21}$	4 607E-04	-2 507E-04	-2 85E-04	3 284E-04	4 535E-04
$\underline{T}_{22}$	4 607E-04	-2.507E-04	3.751E-04	-2 906E-04	4 535E-04
$\bar{T}_{11}$	3 410E-03	3.411E-03	3.410E-03	3 410E-03	3 410E-03
$\bar{T}_{12}$	3 410E-03	3.411E-03	3 410E-03	3 410E-03	3.410E-03
$\underline{\theta}_1$	4 607E-04	3.300E-04	3 751E-04	3.824E-04	4.535E-04
$\underline{\theta}_2$	4.607E-04	3 300E-04	3 751E-04	3.824E-04	4.535E-04
$\bar{\theta}_1$	-3.804E-03	2.748E-03	-2.723E-03	-2.723E-03	-3.739E-03
$\underline{w}_{11}$	-1	-1	-1	-1	-1
$\underline{w}_{12}$	-1	-1	-1	-1	-1
$\underline{w}_{21}$	-1	-1	-1	-1	-1
$\underline{w}_{22}$	-1	-1	-1	-1	-1
$\bar{w}_{11}$	-0.1396	0.1394	-0 1663	-0.1663	-0 1396
$\bar{w}_{12}$	-0 1396	0.1394	-0 1663	-0.1663	-0.1396
$\underline{\Theta}_1$	-1	-1	-1	-1	-1
$\underline{\Theta}_2$	-1	-1	-1	-1	-1
$\bar{\Theta}_1$	-1	-1	-1	-1	-1

Table 3 3(b) The PSPICE DC analysis results of the feedforward ANN circuit with the convergent weights in Table 3 3(a). τ is a target, $\underline{y}_i$ is an output of the hidden layer, and $\bar{y}$ is an output of the output layer

			Column #1,5				Column #2				Column #3,4			
x_1	x_2	τ	$\underline{y}_1$	$\underline{y}_2$	$\bar{y}$	$\tau-\bar{y}$	$\underline{y}_1$	$\underline{y}_2$	$\bar{y}$	$\tau-\bar{y}$	$\underline{y}_1$	$\underline{y}_2$	$\bar{y}$	$\tau-\bar{y}$
1.0	1 0	0 5	5 0	5 0	0 0	0 5	5.0	5.0	0.4931	0.0069	5.0	5.0	0.0	0.5
1.0	4 5	4 5	0 0	0.0	0 5107	3.9893	0.0	0.0	0.0	4.5	0.0	0 0	4.511	-0.011
4 5	1 0	4 5	0.0	0.0	0.5107	3 9893	0 0	0.0	0.0	4.5	0.0	0.0	4.511	-0 011
4.5	4 5	0 5	0 0	0 0	0 5107	-0 0107	0.0	0 0	0 0	0.5	0 0	0.0	4.511	-4.011

4. After executing the learning tasks for all the input-output patterns, the PSPICE dc analysis is executed by setting the weights to the steady state weight values. We then measure their error $\tau - \bar{y}$.

Tables 3.2 and 3.3 summarize the results. The simulation results indicate that this ANN circuit does not learn all the desired targets concurrently.

For example, let's consider Table 3.2. Table 3.2(a) contains the set of (equilibrium) connection weights of the feedforward ANN with sequential learning after executing the PSPICE transient analysis With these weights, the PSPICE dc analysis is executed and their results are summarized in Table 3.2(b). Note that in Table 3.2(b), column #1,5 means that the dc analysis is performed using the data of column #1 and the data of column #5 of Table 3.2(a), separately; the identical dc analysis results are summarized in the same column. In column #1,5, the pattern $(X_1\ X_2\ \tau)$=(4.5 4.5 0.5) is learned with an error equals -0.002 and $(X_1\ X_2\ \tau)$=(1.0 1.0 0.5) is learned with an error equal to 0.5. In column #3,4, the patterns $(X_1\ X_2\ \tau)$= (1.0 4.5 4.5) and (4.5 1.0 4.5) are learned with an error equals to -0.001, and $(X_1\ X_2\ \tau)$=(1.0 1.0 0.5) is learned with an error equals to 0.5; note, however, that the pattern $(X_1\ X_2\ \tau)$ = (4.5 4.5 0.5) is not simultaneously learned.

3.2.3. Silicon MOS Implementation of Feedforward ANNs with Learning

A module chip is designed to implement feedforward ANNs with learning. This module chip can be used as a building block which may be expanded into a large scale feedforward ANNs with learning. To increase the number of neurons in a layer (or the number of the layers), a number of these module chips are connected together vertically (respectively, horizontally).

The sequential-learning circuit #2 is employed to build this module chip. Figure 3.5 depicts the block diagram of an n x m module chip, where n is the number of inputs and m is the number of outputs. Its block representation is shown in the box of Figure 3.5, where x is an n-dimensional input vector, y is an m-dimensional output vector, e is an m-dimensional error vector from the next higher layer, and $\bar{e}$ is an n-dimensional back-propagated error vector to the previous lower layer.

This module chip consists of two sub-circuits: a feedforward sub-circuit and an error-back-propagation sub-circuit. The feedforward sub-circuit generates m outputs, indexed as y_i. These outputs are applied as the inputs to the next higher layer or, in the event they are in the final layer, are used to modify the connection weights, w_{ji} and the bias weights, θ_i, in the error-back-propagation sub-circuit. These modified weights are applied to generate the back-propagated error signals for the previous lower layer and also applied to the feedforward sub-circuit.

Figure 3.6 illustrates a simple example of how this module chip can be used to expand vertically and horizontally. Three $n \times m$ module chips are used to implement a $n \times 2m \times m$ two-layer feedforward ANN with learning. Similarly, when $n = km$, a $n \times km \times m$ two-layer feedforward ANN with learning can be implemented using $k \times n \times m$ module chips.

The input layer consists of just the input voltage nodes, x. The module chips in the middle form (a) hidden layer(s). All the input vectors in the hidden layer are fed directly from the external input voltage nodes. The outputs of the hidden layer are applied to the inputs of the output layer. In the output layer, the desired target vector is applied through the error vector, e. Then a back-propagated error vector is generated and applied to the error vector node, e, of the module chips in the hidden layer.

The design of this module chip is critically dependent upon the number of pins of the MOSIS standard chip package and the design project area. Table 3.4(a) lists the four MOSIS standard chip sizes for a 2 μm process. Table 3.4(b) summarizes the $n \times m$ module chips when the only limitation is given by the number of pins. The module chip has been

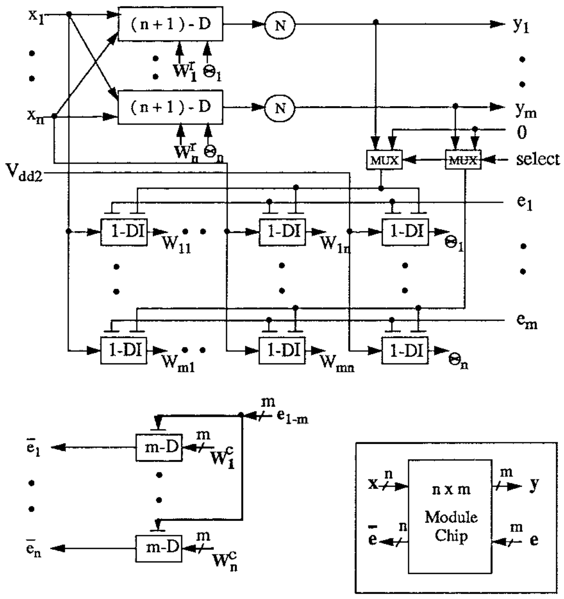


Figure 3.5 The block diagram of the n x m module chip

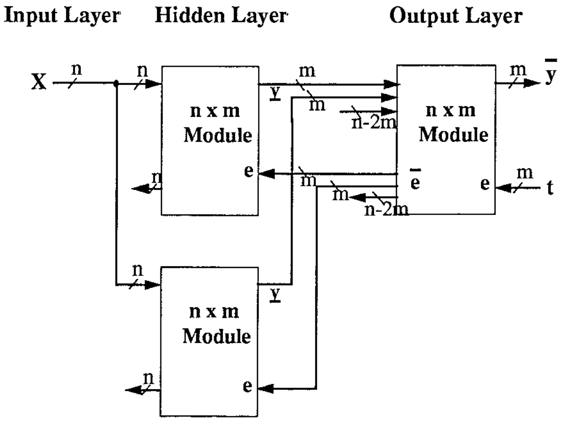


Figure 3.6 nx 2m x m two-layer feedforward ANNs with learning

designed with the restriction that n is at least three times larger than m.

3.2.4. Remarks and Conclusions

The modified learning rule [44] is employed to implement feedforward ANNs with learning. Two sequential-learning ANN circuits are implemented and simulated using the

Table 3.4(a) The four MOSIS standard chip sizes for a 2 μm process

Chip	Die Size(μm)	Max. Project Size(mm)	Area Normalization	Price (Quantities)
Tiny	2540 x 2667	2.22 x 2.25	1	\$500 (4)
Small	4826 x 7112	4.6 x 6.8	6.26	\$2,500 (12)
Medium	7112 x 7112	6.9 x 6.8	9.39	\$5,400 (24)
Large	8218 x 9779	7.9 x 9.2	14.55	\$10,900 (32)

Table 3.4(b) n x m Module Chip vs. the number of pins of MOSIS standard chip

Chip	40 Tiny	40	64	84	108	132
n x m	4 x 1	4 x 2	12 x 4	21 x 6	30 x 10	40 x 10

PSPICE circuit simulator. The PSPICE results infer that the sequential-learning circuits do not enable the ANN circuits to learn all the desired input-output patterns concurrently.

We now remark on the possible reasons for the observed inability of the sequential-learning to concurrently learn multiple input-output patterns. Consider the total squared error function:

$$E = \frac{1}{2} \sum_{p} E_p, \tag{3.13i}$$

where each E_p denotes the squared error function for the desired target p given as

$$E_p = \frac{1}{2} \sum_{j} (t_{pj} - y_{pj})^2 \tag{3.13ii}$$

The sequential-learning ANN circuits are a realization of the error function (3.13ii) for the particular pattern p considered at that instant. As the pattern p is changed, the resulting error function E_p *may* or *may not* retain previous patterns as minima. Consequently, the weight values that the ANN circuit would converge to are only expected to minimize the error due to the latest supplied pattern p. This explains the results of the PSPICE simulations where the sequential-learning ANN circuit may not retain all previously learned patterns. It also justifies the necessity of the simultaneous-learning ANN circuits in the analog implementation of feedforward ANNs. However, simultaneous-learning ANN circuits may diminish the possibility of implementing such networks in large scale. Consequently, the choice between the sequential-learning and simultaneous-learning ANN circuits becomes dependent on a trade-off among basically subjective factors.

4. ANALOG VLSI IMPLEMENTATION OF SYNAPTIC WEIGHTS VIA SIMPLE MOSFETs

We have used a single nMOSFET as a programmable *nonlinear* synaptic weight in the VLSI/LSI MOS implementation of a certain class of artificial neural networks [33, 34, 35]. This *nonlinear* programmable synaptic weight can be utilized in both feedback and feedforward neural nets. Particularly, it was first introduced for a new feedback neural architecture which was proposed in [33, 42]; the proposed ANN circuits have been shown to exhibit the dynamics of continuous-time gradient feedback neural systems.

4.1. Implementation of feedforward neural nets via simple MOSFETs as synaptic weights

To implement an all-MOS feedforward (ANNs), a single nMOS transistor is employed for the connection (synapse) element [40]. Note that the synapse is nonlinear as intended. A one-layer feedforward ANN circuit is depicted in Figure 4.1, where X_i denotes the output of the previous layer, V_i denotes a connection weight voltage, and Y denotes the output of a neuron of the present layer.

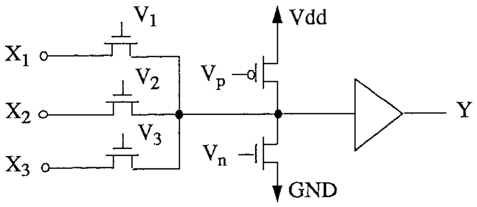


Figure 4.1 A one-layer feedforward ANN with nonlinear elements

A single nMOS transistor is used as a nonlinear, controllable resistive element by controlling its gate voltage. *Its substrate voltage is connected to the lowest (global) voltage level, V_{ss}*. Consequently, depending on their voltage magnitudes, the roles of the source and the drain of floating nMOS transistor may be interchanged due to the MOS transistor symmetric structure.

One pMOS and one nMOS transistors emulate the bias (weight) circuit; the two MOS transistors are connected to the input of each neuron. The source of the pMOS transistor is connected to the highest (global) voltage, V_{dd}, and the source of the nMOS transistor is connected to the lowest voltage, V_{ss}. To generate a more positive bias, $V_p = V_n$ is set to low; and to generate a more negative bias, $V_p = V_n$ is set to high.

A synaptic weight is taken to be the value of the gate voltage of the MOSFET connections. This value may be stored in an analog storage device in order to supply its voltage to the gate of the corresponding floating nMOS transistor. (Few popular analog storage devices include: (1) a large capacitance connected to a very large impedance or (2) the so-called floating-gate device.) The learning process may be executed on a host computer using a suitable learning algorithm. Then the updated weights may be stored in digital form in memory and loaded to a digital-to-analog converter by its memory controller. The analog gate voltages are then stored in local analog memories (such as capacitors) and applied to the gate of each floating nMOS transistor.

This simple "floating" nMOS transistor is suitable for analog VLSI implementation. We have decoupled the design of the architecture of the network from the development of a learning scheme that would obtain a set of control gate voltages which would achieve a certain mapping. Consequently, the learning scheme may be developed off-line or on-line, via analog/digital VLSI or using a host computer whichever the case may be.

The capability of this feedforward ANN is illustrated by solving the XOR problem. Two different architectures are used for this problem using two-layer feedforward ANNs (Figure 4.2). The architecture of these two examples are analogous to the ones described by Rumelhart et al. for the linear weight feedforward ANNs [30].

Figure 4.2 shows two different architectures for solving the XOR problem using two-layer feedforward ANNs and also includes all the gate voltages for the proper connection weights. The gate voltage values of the MOS transistors are properly chosen for the corresponding weights. This two-layer feedforward ANN consists of one input layer, one hidden layer, and one output layer. The input layer can produce inverting and noninverting outputs of the external inputs by using double inverters. The hidden layer has one neuron for one architecture and two neurons for the other. A bias is applied to the hidden layer and the output layer in order to supply proper threshold value of each neuron.

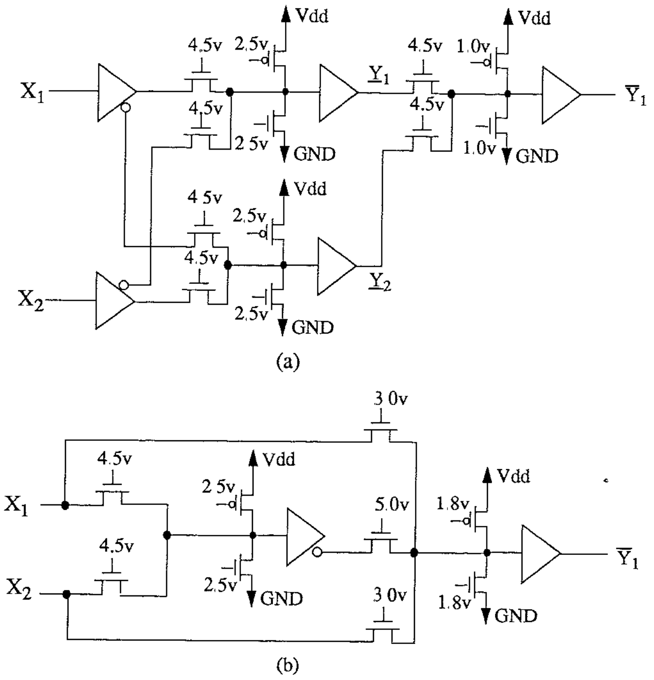


Figure 4.2 Two different architectures for the XOR problem using two-layer feedforward ANNs

PSPICE simulation is executed to show that these two two-layer feedforward ANNs can solve the XOR problem correctly. Table 4.1 summarizes the results of the PSPICE simulation.

Table 4.1 The output of the circuit in Figure 4.2(a) and the expected output of the XOR problem (unit: volts)

		The output of the circuit			The expected output
X_2	X_1	$\underline{Y}_1$	$\underline{Y}_2$	$\overline{Y}_1$	Y_1
0 0	0.0	0.45	0.45	0.0	0 0
0.0	5 0	0.67	5.0	5.0	5.0
5.0	0	5.0	0 67	5.0	5.0
5.0	5.0	0.45	0.45	0.0	0.0

4.2. Implementation of feedback neural nets via simple MOSFETs as synaptic weights: A Hopfield-type neural net

Simple nMOSFETs used as programmable synaptic weights in the VLSI implementation of a certain class of ANN has been described in [33, 34, 42]. Here we describe our use of this (nonlinear) programmable synaptic weights for the VLSI/LSI implementation of feedback ANNs of the Hopfield model. We remark on the outset that there are yet no theoretical justifications that preserve the gradient dynamic feature of these new implementations. As of yet, one can not theoretically rule out the possibility of oscillations or chaotic dynamics when gate voltages are not constrained by symmetry. One can ensure the presence of stable equilibria, however. Each synaptic weight in the Hopfield neural network is implemented by a simple nMOSFET conductance element instead of a pure resistor. In addition, the neuron is simply realized by CMOS double inverters. The synaptic weight can now be adjusted via the gate control voltage of the (nonlinear) nMOSFET conductance element. The negative synaptic weights can be implemented via connecting the complement of the outputs of the neurons instead. Figure 4.3 depicts a prototype 3-neuron Hopfield neural network when the connections are nMOSFETs.

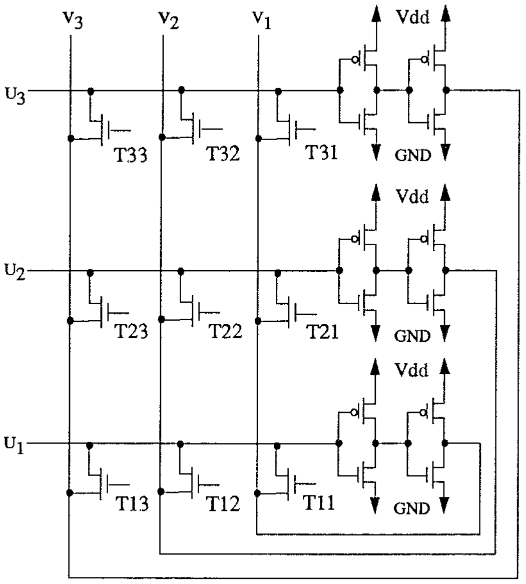


Figure 4.3 A prototype 3-neuron Hopfield ANN using simple MOSFETs

4.2.1. SPICE Simulations

SPICE simulations have been conducted for 2-neuron Hopfield ANNs with nMOSFET interconnects. The power supplies of this neural network circuit, V_{DD} and V_{SS}, were set at 2.5v and -2.5v, respectively. The parameters chosen for this simulation are set at

$$V_G = \begin{bmatrix} -2.5 & 2.5 \\ 2.5 & -2.5 \end{bmatrix},$$

where $V_{G_{ij}}$ denotes the gate voltage of the interconnect nMOSFET in Figure 4.3 and

$C_i = 1.0$ microfarads (i=1,2). This circuit preserves the same qualitative dynamic behavior of the circuit with analog multipliers; and all initial conditions eventually converge to one of two stable equilibrium points with the logical representations (+1,+1) and (-1,-1). Table 4.2 summarizes the SPICE simulation results.

Table 4.2 The SPICE simulation results of 2 neuron Hopfield ANN with nMOSFET interconnects

V_G matrix	Initial States			
	(1.9V,1.2V)	(-1.9V,1.2V)	(1.9V,-1.2V)	(-1.9V,-1.2V)
	Steady States (measured at input nodes)			
$\begin{bmatrix} -2.5 & 2.5 \\ 2.5 & -2.5 \end{bmatrix}$	(1.85,1.17)	(-2.50,-2.50)	(1.85,1.17)	(-2.50,-2.50)
$\begin{bmatrix} 2.5 & -2.5 \\ -2.5 & 2.5 \end{bmatrix}$	(1.85,1.17)	(-2.50,1.17)	(1.85,-2.5)	(-2.5,-2.5)

A 3-neuron prototype of the Hopfield neural network with nMOSFET interconnects has also been simulated using SPICE. By adjusting the gate voltages of the interconnect nMOSFETs, we have been able to obtain different number of stable equilibrium points, from the minimum of two distinct stable equilibria to the maximum of eight distinct stable equilibria. In addition, the circuit also exhibits the dynamic behavior discussed in [36]: there are two distinct stable equilibria co-existing within a single quadrant of the state space.

4.2.2. Experiments using discrete components

In order to verify the SPICE simulation, the discrete-component realization of a 3-neuron prototype of the Hopfield ANNs with nMOSFET interconnects was built in our laboratory. Every CMOS inverter is one inverter of CD4069UBE chips, which is single stage and is not buffered. Every n-channel MOSFET conductance element is realized via a 2N4351 transistor. Table 4.3 summarizes the experimental results. In Table 4.3 we use binary representation for both outputs of the CMOS inverter for all possible initial conditions. Then we interpret any analog voltage greater than 2.5 volts as binary value 1 and less than 2.5 volts as binary value 0.

The experimental results are consistent with the SPICE simulations qualitatively. They show that this ANN circuit has multiple stable equilibria and the number of stable equilibria can be changed by adjusting the gate voltages of the interconnect MOSFETs.

5. CONCLUSIONS

We have presented the implementation of a 4-neuron circuit using digital CMOS circuits. SRAM cells are used to store the synaptic weights, a double inverter is used as a neuron cell, and a CMOS switch is used as a resistive element. This circuit has been designed using the MAGIC VLSI editor and fabricated via MOSIS. The 4-neuron chip has been tested using the TOPAZ VLSI digital testing system and its results are compared with the SPICE simulation results.

We have presented implementable realizations of ANN circuits in MOS silicon VLSI/LSI. We developed a building block comprised of an analog vector-vector multiplier which is viewed as a basic cell for ANN MOS silicon implementation. We describe a learning rule for updating the weights particularly suitable for MOS silicon circuit

Table 4.3 The laboratory test result for a 3-neuron prototype of the Hopfield architecture with nMOSFET interconnects.

T matrix	Number of Stable Equilibria	Initial States							
		000	001	010	011	100	101	110	111
		Steady States							
$\begin{bmatrix} 0.0 & 2.5 & 2.5 \\ 2.5 & 0.0 & 0.0 \\ 2.5 & 0.0 & 0.0 \end{bmatrix}$	1	111	111	111	111	111	111	111	111
$\begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 5.0 \\ 0.0 & 5.0 & 0.0 \end{bmatrix}$	2	001	001	001	001	001	001	111	111
$\begin{bmatrix} 5.0 & 0.0 & 5.0 \\ 0.0 & 5.0 & 5.0 \\ 0.0 & 5.0 & 5.0 \end{bmatrix}$	3	000	000	000	000	000	000	110	111
$\begin{bmatrix} 5.0 & 5.0 & 0.0 \\ 0.0 & 5.0 & 5.0 \\ 0.0 & 0.0 & 5.0 \end{bmatrix}$	4	000	000	000	000	100	100	110	111
$\begin{bmatrix} 5.0 & 0.0 & 0.0 \\ 5.0 & 5.0 & 5.0 \\ 0.0 & 0.0 & 5.0 \end{bmatrix}$	5	000	001	000	001	100	101	100	111
$\begin{bmatrix} 5.0 & 0.0 & 0.0 \\ 0.0 & 5.0 & 2.4 \\ 0.0 & 0.0 & 5.0 \end{bmatrix}$	6	000	001	000	001	100	101	110	111
$\begin{bmatrix} 5.0 & 4.9 & 0.0 \\ 4.85 & 5.0 & 0.0 \\ 0.0 & 0.0 & 5.0 \end{bmatrix}$	7	000	001	000	011	100	101	110	111
$\begin{bmatrix} 5.0 & 0.0 & 0.0 \\ 0.0 & 5.0 & 0.0 \\ 0.0 & 0.0 & 5.0 \end{bmatrix}$	8	000	001	010	011	100	101	110	111

implementation. The new rule removes all derivatives of the sigmoidal function which are present in the usual form of the back-propagation update rule. Yet, the rule guarantees convergence of the update law to minima. It should be mentioned that the implementation of the derivative of the sigmoidal function can not be guaranteed to be the exact analytical derivative of the implemented sigmoidal function employed. It should also be stressed that no analytical support is available for such imperfection except unjustifiable hope that imperfections in the implementation is likely to retain stability and convergence of the dynamic weight update law.

We also describe the radically different view of implementing the *nonlinear* MOS element to emulate the functionality of the synapse No existing theory supports this type of nonlinear synapse for the conventional feedback ANN of the Hopfield type or the conventional feedforward multilayer ANN. Our motivation arises from our use of the nonlinear synapse in the proposed new model [33, 34, 37, 42] where analytical justification is provided. The use of the MOS transistor as a nonlinear synapse here does not necessarily retain the gradient-like characteristics of the conventional model. The use, however, preserves the programmable mapping and/or dynamic system features which are the essential aspect of ANNs [36].

Due to space limitations, we were not able to include the material on our proposed novel ANN architecture; the references [33, 34, 37, 42] provide a description and analysis of this new ANN model, however.

For the novel ANN architecture [33], we have designed a layout for a Tiny-Chip and had it fabricated via MOSIS (in May 1989). The architecture of the TinyChip network adheres to the rigorous mathematical analysis of its model [37, 42] and it appears to emulate the connectivity of the Horizontal and Bipolar cells in the retina. The TinyChip is capable of [34, 59]:

(1) preprocessing data using nonlinear averaging to restore/enhance images when it is configured in the hexagonal/retinal architecture, and

(2) storing (and retrieving) distinct multiple analog data of any number between 0 and 2^n, where n is the number of neurons.

We have extensively tested the TinyChip over the course of a year and found that it performs the two tasks ((1) and (2) above). We have also developed learning algorithms that accurately ensure the storing (and retrieving) of a given set of data.

Recently, we have designed a layout of our all-MOS analog VLSI neural circuit which will contain 50 neurons and will have on-chip learning for the (nonlinear) weights. The small standard chip will be able to process 7x7 sub-images or 50-d feature-space data vectors. This small chip is a direct expansion of our earlier TinyChip.

Finally, we conclude that standard MOS technology will in fact enable us to produce *working* integrated neural network chips in the very near future. These chips will be tailored for specific tasks, will include in the order of 100 neurons with possible digital on-chip learning, and will operate in the order of micro seconds. For the long term, new material and new technologies may be needed to meet the requirements of more complex and versatile tasks.

ACKNOWLEDGMENT

This research was supported in part by ONR Grant N00014-89-J-1833 and in part by the State of Michigan Research Excellence Fund (REF).

REFERENCES

[1] J. A. Anderson and E. Rosenfeld, editors, Neurocomputing, Cambridge, MA: The MIT Press, 1988.

[2] J. N. Babanezhad and G. C. Temes, "A 20-V Four-Quadrant CMOS Analog Multiplier," IEEE J. Solid-State Circuits, SC-20, Dec. 1985, pp. 1158-1168.

[3] M. Banu and Y. Tsividis, "Fully-integrated active-RC filters in MOS technology," IEEE J. Solid-State Circuits, SC-18, Dec. 1983, pp. 644-651.

[4] T. Borgstrom et al., "A Neural Network Integrated Circuit Utilizing Programmable Threshold Voltage Devices," IEEE Proc. of ISCAS' 89, 1989, pp. 1227-1230.

[5] M. A. Cohen and S. Grossberg, "Absolute Stability of Global Pattern Formation and Parallel Memory Storage by Competitive Neural Networks," IEEE Trans. on Systems, Man, and Cybernetics, vol. SMC-13, No. 5, September/October 1983, pp. 815-825.

[6] Z. Czarnul, "Novel MOS Resistive Circuit for Synthesis of Fully Integrated Continuous-Time Filters," IEEE Trans. on Circuits and Systems, vol. CAS-33, Nov. 1986, pp. 718-721.

[7] Z. Czarnul, "Modification of the Banu-Tsividis continuous-time integrator structure," IEEE Trans. on Circuits and Systems, vol. CAS-33, July 1986, pp. 714-716.

[8] DARPA Neural Network Study, October 1987 - February 1988, Published by AFCEA International Press, November 1988.

[9] S. Eberhardt, T. Duong and A. Thakoor, "Design of parallel hardware neural network systems from custom analog VLSI 'building block' chips," Proceedings of International Joint Conference on Neural Networks, Washington D.C., June 1989, pp. 183-190.

[10] B. Furman and A. A. Abidi, "An Analog CMOS Backward Error-Propagation LSI," Asilomar Conf. on Signal, Control, and Systems, 1988, pp. 645-648.

[11] H.P. Graf and P. Vegvar, "A CMOS Implementation of a Neural Network Model," Proc. Stanford Conf. Advanced Research in VLSI, MIT Press, 1987, pp. 351-367.

[12] P. R. Gray and R. G. Meyer, "MOS operational amplifier design - A tutorial overview," IEEE J. Solid-State Circuits, SC-17, Dec. 1982, pp. 969-982.

[13] B. Gilbert, "A high-performance monolithic multiplier using active feedback," IEEE J. Solid-State Circuits, SC-9, Dec. 1974, pp. 364-373.

[14] M. Holler, S. Tam, H. Castro and R. Benson, "An electrical trainable artificial neural network (ETANN) with 10240 'Float gate' synapses," Int. Joint Conf. Neural Networks, vol. 2, June 1989, pp. 191-196.

[15] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," Proc. Natl. Acad. Sci. U.S.A., vol. 79, 1982, pp. 2554-2558.

[16] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," Proc. Natl. Acad. Sci. U.S.A., vol. 81, 1984, pp. 3088-3092.

[17] J. J. Hopfield and D. W. Tank, "Neural computation of decisions optimization problems," Biological Cybern., vol. 52, 1985, pp. 141-152.

[18] J. J. Hopfield and D. W. Tank, "Collective computation with continuous variables," in Disordered Systems and Biological Organization, E. Bienenstock, F. Fogelman, and G. Weisbush, eds., Springer-Verlag, 1985.

[19] J. J. Hopfield and D. W. Tank, "Simple neural optimization networks: an A/D converter, a signal decision circuit, and a linear programming circuit," IEEE Trans. on Circuits and Systems, vol. CAS-33, no 5, May 1986, pp. 533-541.

[20] W. Hubbard, J. S. Denker, H P. Graf, R. E. Howard, L.D. Jackel, B. Straughn, and D. Schwartz, "Electronic Neural Networks," AIP Neural Networks for Computing, 1986, pp. 227-234.

[21] M. Ismail, "Four-transistor continuous-time MOS transconductor," Elec. Lett., vol. 23, no. 20, Sep. 1987, pp. 1099-1100.

[22] N. I. Khachab and M. Ismail," Novel Continuous-Time All MOS Four Quadrant Multipliers," Proc. of IEEE Int. Symp. Circuits and Systems, May 1987, pp. 762-765.

[23] F. Kub, K. Moon, I. Mack and F. M. Long, "Programmable Analog Vector-Matrix Multipliers," IEEE J. Solid-State Circuits, SC-25, Feb. 1990, pp. 207-214.

[24] B. Lee, J. Lee and B. Sheu, "VLSI Image Processors Using Analog Programmable Synapses and Neurons," Proceedings of International Joint Conference on Neural Networks, San Diego, California, June 1990, pp. II-575-580.

[25] W. S. McCulloch and W. Pitts, Bull. Math Biophys., 5, 1943, pp 115-133.

[26] C. A. Mead, Analog VLSI and Neural Systems, Addison-Wesley, 1989.

[27] J. H. Li, A. N. Michel and W. Porod, "Qualitative Analysis and Synthesis of a Class of Neural Networks," IEEE Trans. on Circuits and Systems, vol. 35, no. 8, August

1988, pp. 976-986.

[28] A. N. Michel, J. A. Farrell and W Porod, "Qualitative Theory of Neural Networks," IEEE Trans. on Circuits and Systems, vol. 36, no. 2, Feb. 1989, pp. 229-243.

[29] A. N. Michel and J. A. Farrell, "Associative Memories via Artificial Neural Networks," IEEE Control Systems Magazine, vol. 10, no. 3, April 1990, pp. 6-17.

[30] D. E Rumelhart, J. L. McClelland, and the PDP Research Group Eds., "Parallel distributed processing-Explorations in the microstructure of cognition," vol. 1, Foundations. Cambridge, MA: MIT Press, 1986.

[31] J. P. Sage et al., "An Artificial Neural Network Integrated Circuit Based on MNOS/CCD Principle," AIP Neural Networks for computing, 1986, pp. 381-385.

[32] F. M. A. Salam, N. Khachab, M. Ismail and Y. Wang, "An Analog MOS Implementation of the Synaptic Weights for Feedback Neural Nets," 1989 IEEE International Symposium on Circuits and Systems (ISCAS), Portland, Oregon, CA, May 9-11, 1989, pp. 1223-1226.

[33] F. M. A. Salam, "A Model of Neural circuits for Programmable VLSI Implementation of the Synaptic Weights for Feedback Neural Nets," 1989 IEEE International Symposium on Circuits and Systems (ISCAS), Portland, Oregon, May 1989, pp. 849-851

[34] F. M. A. Salam and Y. Wang, "Neural Circuits for Programmable Analog MOS VLSI Implementation," Proc. of 32nd Midwest Symposium on Circuits and Systems, Champaign, Illinois, Aug. 1989, pp. 240-243.

[35] F. M. A. Salam and M. R. Choi, "Analog MOS Vector Multipliers For The Implementation of Synapses In Artificial Neural Networks," The Journal of Circuits, Systems, and Computers, 1991 (to appear).

[36] F M. A. Salam and Y. Wang, "Some Properties of Dynamic Feedback Neural Nets," Proc. 27th IEEE Conference on Decision and Control, December 1988, pp. 337-342.

[37] F. M. A. Salam, "Global convergence properties of a new feedback model for neuro-engineering," Memorandum No. MSU/EE/S 89/01, Department of Electrical Engineering, Michigan State University, East Lansing, MI 48824-1226, 25 January 1989.

[38] F. M. A. Salam, Y. Wang and M. R. Choi, "On The Analysis of Dynamic Feedback Neural Nets," IEEE trans. on Circuits and Systems, vol. 38, February 1991.

[39] F. M. A. Salam, M. R. Choi, Y. Wang and B. Javidi, "On The Design of Neural Nets," 25th Annual Allerton Conference on Communication, Control, and Computing, Sep. 30 - Oct. 2, 1987, pp. 1142-1151.

[40] F. M. A. Salam and M. R. Choi, "A Feedforward Neural Network For CMOS VLSI Implementation," IEEE Midwest Symposium on Circuits and Systems, Aug. 1989, pp. 489-492.

[41] F. M. A. Salam, "A Formulation For The Design of Neural Processors," Proc. of the 2nd IEEE Annual International Conference on Neural Networks (ICNN), San Diego, CA, July 24-27, 1988, pp. I-173-180.

[42] F. M. A. Salam, "New Artificial Neural Models: Basic Theory and Characteristics," 1990 IEEE International Symposium on Circuits and Systems (ISCAS), New Orleans, Louisiana, May 1990, pp. 200-203.

[43] F. M. A. Salam and M. R. Choi, "An All-MOS Analog Feedforward Neural Circuit With Learning," 1990 IEEE International Symposium on Circuits and Systems (ISCAS), May 1990, pp. 2508-2511.

[44] F. M. A. Salam, "A Modified Learning Rule for Feedforward Artificial Neural Nets For Analog Implementation," Memorandum No. MSU/EE/S 90/02, Department of

Electrical Engineering, Michigan State University, East Lansing, MI 48824-1226, 26 January 1990.

[45] F. M. A. Salam, S. Bai and J. Hou, "Dynamic of Feedback Neural Nets with Unsupervised Learning," 1990 IEEE International Joint Conference on Neural Networks (IJCNN), San Diego, CA, June 17-21, 1990, pp. II-239-244.

[46] F. M. A. Salam and S. Bai, "A New Feedback Neural Network with Supervised Learning," IEEE Trans. on Neural Networks, January 1991.

[47] F. M. A. Salam, "Learning in Feedforward Artificial Neural Nets For Continuous-time Implementations," Interface' 90 proceedings, East Lansing, MI, May 15-16, 1990 (Proceedings to appear in 1991).

[48] D Schwartz et al., "A Programmable Analog Neural Network Chip," IEEE J. of Solid-State Circuits, vol. 24, No. 2, Feb 1989, pp 313-319

[49] R. Shimabukkuro et al., "Dual-Polarity Nonvolatile MOS Analog Memory Cell for Neural-type Circuitry," Elec. Lett., vol. 24, no. 19, 1988, pp. 1231-1232.

[50] R. Shimabukkuro et al., "Circuitry for Artificial Neural Networks with Non-Volatile Analog Memory," Proc. of ISCAS' 89, 1989, pp. 1217-1220.

[51] M. A. Sivilotti et al., "VLSI Architectures for Implementation of Neural Networks," AIP neural networks for computing, 1986, pp. 408-413.

[52] M. Sivilotti, M. Emerling and C. Mead, "A Novel Associative Memory Implemented Using Collective Computation," 1985 Chapel Hill Conference on VLSI, pp. 329-342.

[53] B.-S. Song, "CMOS RF Circuits for Data Communications Applications," IEEE J. Solid-State Circuits, SC-21, no. 2, April 1986, pp 310-317.

[54] D. Soo and R. G. Meyer, "A four-quadrant NMOS analog multiplier," IEEE J. Solid-State Circuits, SC-17, Dec. 1982, pp. 1174-1178.

[55] A. P. Thakoor, A. Moopen, J. Lambe and S. K. Khanna, "Electronic hardware implementations of neural networks," Applied Optics, vol 26, 1987, pp. 5085-5092.

[56] Y. Tsividis, M. Banu and J. Khoury, "Continuous-time MOSFET-C Filters in VLSI," IEEE Trans. on Circuits and Systems, vol. CAS-33, Feb. 1986, pp. 125-140.

[57] Y. Tsividis et al., "Analog Circuits for Variable-Synapse Electronic Neural Networks," Elec. Lett., vol. 23, no. 24, Nov. 1987, pp 1313-1314.

[58] D. E. Van Den Bout and T. K. Miller III, "A digital architecture employing stochasticism for the simulation of Hopfield neural nets," IEEE Trans. on Circuits and Systems, vol. 36, no 5, pp. 732-738, May 1989.

[59] Y. Wang and F. M. A. Salam, "Design of Neural Network Systems from Custom Analog VLSI Chips," 1990 IEEE International Symposium on Circuits and Systems (ISCAS), New Orleans, Louisiana, May 1990, pp. 240-243.

[60] M. White et al., "Electrical Modifiable Nonvolatile Synapses for Neural Network," Proc. of ISCAS' 89, 1989, pp. 1213-1216.

AUTHOR INDEX