NEURAL NETWORK TECHNIQUES
IN MANAGERIAL PATTERN RECOGNITION

By

SHOUHONG WANG

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Doctor of Philosophy

McMaster University

NEURAL NETWORK TECHNIQUES

IN MANAGERIAL PATTERN RECOGNITION

DOCTOR OF PHILOSOPHY (1990) McMASTER UNIVERSITY

(Management Science / Systems) Hamilton, Ontario

TITLE: Neural Network Techniques in Managerial Pattern Recognition

AUTHOR: Shouhong Wang, B.Eng. (Tsinghua University, China)

MBA (Tsinghua University, China)

SUPERVISOR: Dr. Norman P. Archer

NUMBER OF PAGES: xiii, 228

ABSTRACT

The management area includes a large class of pattern recognition (classification) problems. Traditionally, these problems have been solved by using statistical methods or expert systems. In practice, however, statistical assumptions about the probability distributions of the pattern variables are often not verifiable, and expertise concerning the correct classification is often not explicitly available. These obstacles may make statistical methods and expert system techniques difficult to apply. Since the early 1980s neural network techniques have been widely used in pattern recognition, especially after Rumelhart's back propagation learning algorithm was adapted to the solution of these problems. The standard neural network, using the back propagation learning algorithm, requires no statistical assumptions but uses training sample data to generate classification boundaries, allowing it to perform pattern recognition.

In this dissertation the neural network's behavior in classification boundary generation is analyzed. Based on this analysis, three models are developed. The first model improves the classification performance of neural networks in managerial pattern recognition by modifying the training algorithm through the use of monotonicity. Using simulated and real data, the developed model is tested and verified. The second model solves bias problems caused by small sample size in neural network classification results. The third model develops multi-architecture neural networks to supply

decision makers with more natural pattern recognition information, based on fuzzy theory.

ACKNOWLEDGEMENTS

This dissertation owes its existence to many people whose help I deeply appreciate and would like to acknowledge here.

I first would like to thank my supervisor, Dr. N. P. Archer for his invaluable advice and guidance throughout this research. His help has gone beyond the call of supervisory duties. I shall always be grateful to him for his patience, and his time and energy spent working with me. I also wish to thank Dr. G. W. Torrance and Dr. D. P. Taylor for serving on my supervisory committee.

More generally I would like to thank other people at McMaster University who have assisted me in completing my graduate studies, especially including Dr. W. G. Truscott, Dr. M. W. L. Chan, Dr. G. O. Wesolowsky, and Dr. M. Parlar.

Finally, my sincere appreciation and love go to my parents, my wife, and my son for their support and sacrifices while I have been apart from them during my years of graduate studies.

TABLE OF CONTENTS

NOTATION

B	discriminant function coefficient vector in Fisher method
B(X)	discriminant function of observation **X**
b_i	coefficient of x_i in discriminant function
C	set of goal classes
c	goal class, $c \epsilon C$
c^s	class to which s belongs
D	number of Fisher discriminant functions for more than two group case
d_α	desired output of neural network node α
E	error measured in learning
e	power of the test in sequential classification
F	activation function in neural network
f	distribution function
fz(X)	fuzzy function of **X**
G	number of clusters in the MF model algorithm
g	subscript
h	the number of hidden nodes in two-layer neural network
i	subscript
j	subscript
k	number of goal classes in the set of classes C
L	output of a hidden node in Kolmogorov's theorem
M	momentum
m	dimension of pattern vector
n	dimension of feature vector
net_α	net input for node α
o_α	output value of node α
$(o_\alpha)_s$	output value of node α after learning sample s
Pr	probability
R	testing data set
r	subscript
S	training data set
s	training data sample point
T	temperature
t	time
U_i	weight vector of the inputs of i-th hidden node in two layer neural network model
u	constant in Kolmogorov's theorem
v_i	weight value of the output of i-th hidden node in two layer and single output dimension neural network
W^1	weight matrix between hidden layer and output layer
W^2	weight matrix between input layer and hidden layer
w_i	weight on the ith input to perceptron
$w_{\alpha\beta}$	weight on the connection from node β to α in neural network
X	pattern vector

x_i	ith component of **X**
Y	feature vector
y_i	ith component of **Y**
α	general expression of the neural network node
β	general expression of the neural network node preceding to node α
$\Gamma(\tau)$	impurity function of τ
γ	Fisher ratio
δ	error signal in neural network learning algorithm
ϵ	constant in Kolmogorov's theorem
ζ_s	classification score (from linear discriminant analysis)
η	learning rate
θ	threshold in F
ι	one-dimensional feature vector value for **T**
κ	subscript
λ	parameter of λ-complementation in fuzzy set
μ	mean of a random variable
ν	sample size of S
Π	transfer function in Kolmogorov's theorem
ξ	small number for the training scheme model
ρ	subscript
Σ	covariance matrix of multivariate variable
σ	covariance
τ	tree node in hierarchical classifier
$\mathbf{T}_s$	desired value of the feature vector of s
Φ	cumulative normal distribution function
ϕ	mapping function from **X** to **Y**
χ	transfer function in Kolmogorov's theorem
ψ	transfer function in Kolmogorov's theorem
Ω	pattern space
ω_i	angle between pattern vector **X** and its i-th component x_i
e	error tolerance in neural network learning algorithm
ℓ	split in hierarchical classifier

FIGURES

TABLES

CHAPTER ONE

INTRODUCTION

1.1. GENERAL DESCRIPTION OF NEURAL NETWORKS

For our purposes, artificial neural networks are defined as massively parallel interconnected networks of simple (usually adaptive) elements and their hierarchical organizations, which are intended to react to information or the objects of the real world in a manner analogous to biological nervous systems [Kohonen 1987]. Neural networks or simply "neural nets" may also be referred to as connectionist models, parallel distributed processing models, and neuromorphic systems [Lippmann 1987].

Neural network architecture may be described in various ways, depending upon its desired function [Lippmann 1987, Hecht-Nielsen 1987]. The most general topology of a neural network is shown in Figure 1.1.

The neural network system carries out the information processing operation as a mathematical mapping ϕ of vector **X** to vector **Y** so that $\mathbf{Y}=\phi(\mathbf{X})$, where **X** is the vector of external inputs to the network, and **Y** is the vector of outputs. Units ("neurons") within the network may receive input signals and/or lateral feedbacks

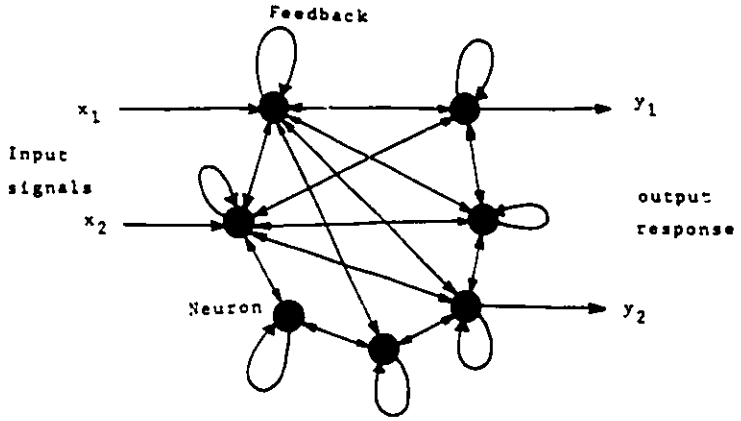


Fig. 1.1. General Neural Network Model
(modified from [Guyon 1989])

in parallel. Many parallel lines among the units may be needed to establish the input-output relationships. The detailed structure of layered neural networks will be discussed later.

McCulloch and Pitts' research in the early 1940s (e.g. [McCulloch and Pitts 1943]) is regarded as the first theoretical research which described the fundamentals of neural computing [Kohonen 1987]. This research showed how neural-like networks could be mathematically manipulated, and suggested the need for such networks to learn (cf. [Rumelhart et al. 1986]). Active research on adaptive or learning neural networks was carried out up through the 1960s. The best known work was Rosenblatt's [Rosenblatt 1962], and Minsky and Papert's [Minsky and Papert 1969] research on the perceptron, which is a simple neuron-like learning network. By the early 1970s this activity had declined substantially [Griffiths 1988], at least partially due to Minsky and Papert's pessimistic

evaluation of the capabilities of the perceptron (cf. [Rumelhart et al. 1986, p65, p111]). More interest developed in the late 1970s when multilayer nonlinear threshold neural network models were designed with much more potential for learning than the perceptron. Recently there has been an explosive rise of interest in neural networks, mainly due to the impression that general neural computers may become highly important in the marketplace [Anderson 1988] [Griffiths 1988]. Neural network modelling has usually been pursued in connection with psychological theories and neurophysiological research. However, interest is also spreading within the engineering, mathematical computing, and computer science (artificial intelligence) communities. A considerable amount of recent research in neural networks has been directed towards pattern recognition applications [Lippmann 1989].

Research in neural networks is still in its infancy, but it is expected that neural network models will be useful both as models of real brain functions and as computational devices for many applications, including pattern recognition or classification [Widrow et al. 1987, 1988] [Lippmann 1987, 1989], [Guyon et al. 1989], handling of complex searches for relevant knowledge from associative memory [Kohonen 1987], signal processing [Parkkinen et al. 1988], and optimization [Hopfield and Tank 1985]. Neural networks have been used to solve particular problems (e.g. [Bounds et al. 1988]), or to implement particular computational models [Alkley 1987]. Some neural network models have been used to mimic human behavior in problem solving and learning [McClelland et al. 1986]. Some investigations have been carried out on neural networks with particular

architectures in order to study their computational properties [Hopfield 1984]. Most of these investigations are motivated by the prospect of "neural computers" -- new generation computers with parallel processing characteristics [Anderson 1988].

In the context of machine learning [Michalski et al. 1983, 1986], neural networks initially have little task-oriented knowledge, but they learn by incrementally modifying connection strengths between individual elements. This contrasts with symbolic concept acquisition and knowledge intensive domain-specific learning which are typically supported by logical expressions, production rules (expert systems), or semantic networks.

The boundary between pattern recognition and machine learning is not clear-cut. For instance, pattern recognition by a neural network is often implemented via a kind of learning device, as will be shown in Chapter 2.

In order to give a scenario for pattern recognition and machine learning in managerial decision making, consider a simple example. The shareholders' evaluation of a stock is based on the return mean and variance (the so-called mean-variance analysis in finance). Given a stock's return mean and variance, the shareholder must judge if the stock is "good" or "bad" (Figure 1.2). The boundary separating a stock portfolio into two classes is a utility function curve [Rudd and Clasing 1982] [Keeney 1972]. Usually, the shape of the utility function curve is not regular, and expressing the curve mathematically is relatively complicated. However, on a two-dimensional plot, one can intuitively draw a curve to separate the two classes. The question is how to program a computer to solve

this kind of problem, especially in the multivariate data case where two-dimensional graphical methods are not applicable. It will be shown that the neural network approach is able to solve this type of problem by making few assumptions, and by learning from existing data.

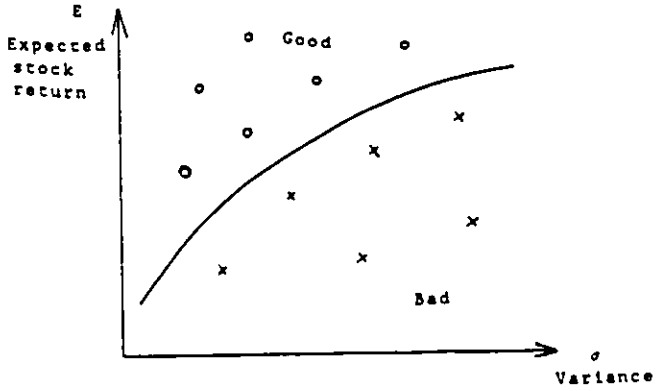


Fig. 1.2. Stock Mean-Variance Chart

1.2. BACKGROUND

1.2.1. A CLASS OF PROBLEMS IN MANAGERIAL DECISION MAKING -- PATTERN RECOGNITION / CLASSIFICATION

Pattern recognition is the prediction of the value of a goal class of any object by its description [Lbov 1982]. A pattern recognition machine or classification decision function may be divided into two parts [Nilsson 1965] [Young and Calvert 1974]: a feature extractor and a class selector, as shown in Figure 1.3. The

pattern vector **X**, with m components $(x_1, x_2 \ldots x_m)$ describes an object. The feature extractor is a transformation

$$\mathbf{Y} = \phi\ (\mathbf{X}) \qquad (1.1)$$

which transforms a pattern vector **X** into a feature vector **Y**, with n components $(y_1, y_2 \ldots y_n)$. Based on **Y**, the class selector then selects a class $c_i \epsilon C$, where $C=\{c_1, c_2 \ldots c_k\}$ is a set of k goal classes to which **X** may belong, Usually, the class selector is a very simple function [Nilsson 1965], such as $y_i = \max_j (y_j) \Longrightarrow c_i$, where $j=1 \ldots i \ldots q$, and $n=k=q$.

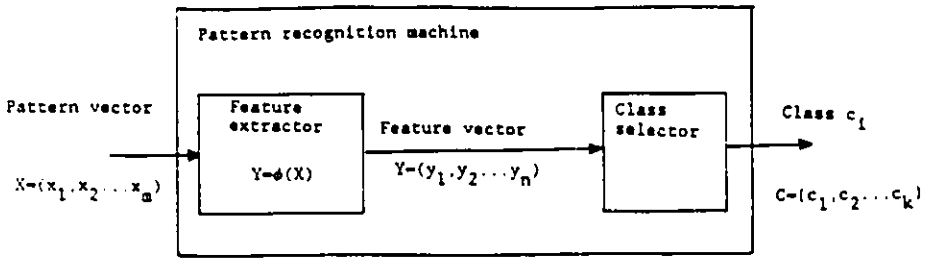


Fig. 1.3. Pattern Recognition Machine

Pattern recognition involves two stages. The first stage is the determination of the decision function ϕ from a "training sample", where the goal classes are known for a given object pattern. This is a synthesis problem [Karplus 1983]. The second stage, once ϕ has been determined, is to use ϕ to classify new objects from the same population according to their patterns, where the goal classes, of course, are not known. This is an analysis problem [Karplus 1983]. The accuracy of prediction is usually measured with a test

sample which is different from the training sample. A closely related statistical technique is discriminant analysis, which will be described later in more detail.

In the management area there is a class of managerial decisions which can be reduced to pattern recognition or classification. Before defining the characteristics of these problems, a few published examples of managerial classification problems and their solutions will be described.

Example 1. Financial ratios analysis and prediction of corporate bankruptcy.

Altman [Altman 1968] developed a linear discriminant model for company bankruptcy prediction where a sample of sixty-six firms was utilized to establish a decision function to discriminate between companies in two goal classes: those which would go bankrupt and those which would survive. The pattern vector used financial ratios, and the model predicted bankruptcy with an accuracy of 94 percent in the training sample and 95 percent in a test sample.

Example 2. Credit extension decision.

A number of studies have been published [McGrath 1960], [Greer 1968], [Myers and Forgy 1963] in which historical data on customer characteristics (e.g. age, income, etc.) were used for the pattern vector, and goal classes of "good" and "bad" credit risks were used for customers with known records. The decision function thus determined (using a linear discriminant model) is then used to classify new customers.

Example 3. Assessment of strategic planning.

Ramanujam et al. [1986] developed a set of key attributes of planning systems as a pattern vector, and employed discriminant analysis to classify planning systems as more and less effective. They claimed 81.7 percent overall classification accuracy.

These typical examples of managerial pattern recognition have certain characteristics:

(1) They assign individuals to a class on the basis of pattern data that are related to the class. No analytic algorithm is available to solve these problems; it is almost impossible to generate an objective function and find the optimal assignment solution subject to a set of known constraints using mathematical methods such as 0/1 linear programming [Hillier & Lieberman 1986]. In managerial pattern recognition cases, classification techniques have to be employed in order to achieve a low error rate when assigning an unknown observation to a class.

(2) Another common feature is that almost all managerial problems involve classification into two classes. Indeed, when a numerical approach is used in the feature vector (discussed later), there is a continuous range between extreme "no" and "yes" answers (i.e. from 0 to 1). There may also be moderate values of the feature extraction result, (e.g. 0.4-0.6, which could represent "maybe" or "indeterminate").

(3) In most managerial classification cases, the statistical distributions of the pattern variable data are not known. Usually, they are assumed to be multivariate normal distributions with

parameter estimates based on the available training sample. This assumption may or may not affect the result. Also, there is usually little explicit knowledge about why an individual sample point belongs to its class.

(4) The examples given above all used a linear discriminant function which generates a linear decision boundary. Because there is no prior knowledge about boundary shape, the linear boundary assumption seems to be a reasonable approximation, and linear discriminant analysis is commonly used when no knowledge other than the sample data is available.

1.2.2. PROBLEM DEFINITIONS

In this subsection we discuss definitions relevant to managerial pattern recognition.

1.2.2.1. MANAGERIAL PATTERN RECOGNITION

In Section 1.2.1. the general mathematical definition of pattern recognition was given. In fact, pattern recognition implies much more than pure mathematical matching between the observed description and the goal class. Artificial intelligence research considers pattern recognition to be the use of a human's specific knowledge and abstraction ability [Rich 1983], and attempts to solve classification problems in the "human" mode. To apply the generic theory of classification to a specific problem requires a thorough understanding of the problem, including its peculiarities and special

difficulties. For the purpose of our discussion, managerial pattern recognition is thus defined as follows:

> Managerial pattern recognition (or classification) is the use of a decision function to classify objects into two or more classes, based on historical observations from the same object population and on the generic properties of managerial problems.

The generic property of managerial pattern recognition problems most relevant to the present research is the monotonicity of the utility function. We will discuss this concept and its application to neural networks in Chapter 3. Other relevant generic properties of managerial classification problems include:

(1) rough data;

(2) very limited sample data (several to a couple of hundred points);

(3) ill-structured decision making, and

(4) subjective judgements.

1.2.2.2. CHARACTERISTICS OF CLASSIFICATION PROBLEMS

There are nine major characteristics [Young and Calvert 1974] [Lachenbruch 1975] [Green 1978] [Hand 1981] which constitute the knowledge available to build models to solve classification problems:

(1) Dimensions of the pattern vector.

In equation (1.1) each component of the m-dimensional pattern vector **X** is a variable or attribute of the object to be classified [Young and Calvert 1974]. Typical dimensionality of the pattern vector in managerial classification problems ranges from two to twenty (cf. [Green 1978] [Churchill 1979] [Zaltman and Burger 1975]). Generally, the larger the number of pattern dimensions, the more complex the system. Some techniques such as the stepwise method of discriminant analysis [Nie et al. 1975] can be used to reduce the number of pattern dimensions in order to improve understanding of the problem. However, reducing the number of pattern dimensions increases the potential risk of losing relevant information.

(2) Number of goal classes.

Given a problem, the number k of goal classes (Figure 1.3) is determined. In the sense of managerial pattern recognition, the number k in turn corresponds to future potential managerial decisions. For example, in the stock mean-variance analysis the classification of a stock as either good or bad basically corresponds to a future managerial decision "buy" or "sell". Sometimes, one may categorize the stock into several groups such as "excellent", "good", "fair", "bad", and so on. Nevertheless, if the resulting managerial decision only involves two options, the classification is still a two goal class problem, but with multiple levels within a class. Solving problems with more than two goal classes is similar to solving a two class problem; however, the approach becomes less intuitive [Lachenbruch 1975].

(3) Dimensions of the feature vector.

As shown in Figure 1.3, a pattern recognition system first transforms the pattern vector **X** into the feature vector **Y** before the goal class is selected. The dimension n of **Y** is basically dependent upon the number k of goal classes. In the two class case, **Y** may be collapsed to a scalar y. Suppose that $y<0.5$ indicates class 1, and $y\geq 0.5$ is class 2, assuming the range of y is [0,1]. An alternative is to specify **Y** with dimension 2. If the range is [0,1] for y_1 and y_2, for instance, $y_1<0.5$ and $y_2\geq 0.5$ could indicate class 1, and $y_1\geq 0.5$ and $y_2<0.5$ could indicate class 2. Other values for the y_1, y_2 pair could then indicate an indeterminate classification.

(4) A priori probabilities.

A priori probabilities $Pr(c_i)$ about the occurrence of goal classes are useful for predicting classification results [Hand 1981]. In the usual case, the $Pr(c_i)$'s are not related to the sample data themselves, but are estimated, based on previous observations of the proportion of occurrences of goal classes.

(5) Likelihood probabilities.

Knowledge about the probability distributions of the sample's pattern data for each given class $f_{c_i}(\mathbf{X})$ (i.e. $Pr(\mathbf{X}|c_i)$) is crucial to the statistical estimates involving the classification decision [Hand 1981]. Lack of knowledge about the distribution substantially weakens a statistical tool's classification power.

(6) Structure knowledge relating a pattern to its class.

In some cases, it is well known why a pattern belongs to its class rather than to others. Assigning a pattern to a particular class can be generalized into a structured tree or a set of rules [Breiman et al. 1984]. However, in many managerial problems this kind of knowledge is often not available.

(7) Decision region / boundary.

Given a problem, any pattern can be represented by a point in an m-dimensional Euclidean space Ω_X , which is called its pattern space. For example, assume the pattern dimension is 2, and the two components of X (x_1 and x_2) range from 0 to 1 respectively. Then Ω_X is square with side 1 (see Figure 1.4).

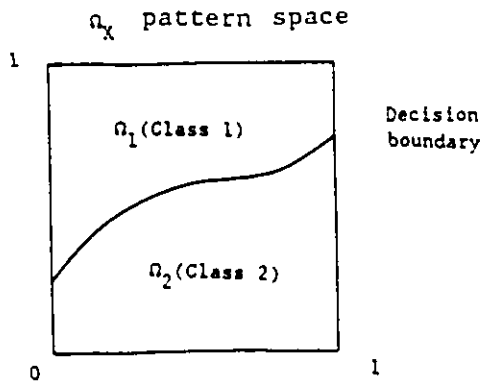


Fig. 1.4. Decision Regions And Decision Boundary

Suppose there are two classes. A pattern recognition function will divide Ω_X into two disjoint decision regions, Ω_1 and Ω_2 , each corresponding to one class. The separating line (pattern recognition

function) is called a decision boundary [Young and Calvert 1974], [Hand 1981]. When there are more than 3 pattern dimensions, the decision boundaries are hypersurfaces. If the decision boundary is a straight line, plane, or hyperplane, then it is linear; otherwise, it is nonlinear. In general, the shape of the decision boundary is not known unless the form of the multivariate random variables which generate it are known.

(8) Training Data Set vs. Testing Data Set.

Sample data are usually employed for two purposes: building the classification model, and testing the model [Toussaint 1974]. The data subset S from the sample used for modeling is called the training data set, and that used for testing is called the testing data set R. The two data subsets are not necessarily disjoint.

(9) Sample data quality.

In practice, sample data are not perfect. Sample data quality influences the classifier's performance. There are three major types of problems with sample data in classification, which may cause difficulties for particular classification techniques.

(a) Missing values.

The most obvious method of handling a missing value case is simply to omit the sample point. However, if the sample size is small, this method is less acceptable. The alternative approach is to estimate the missing value in order to fully utilize the available data [Green 1978].

(b) Correlated variables.

If the pattern variables have not been properly defined, a high degree of correlation may exist between the variables themselves. The problem not only results in redundant information, but also brings difficulties for some type of multivariate analysis such as discriminant analysis [Green 1978, p227], where multicollinearity causes reduced precision in estimating the coefficients of the classification function in manipulation of matrices.

(c) Conflict data.

Due to the incompleteness of a pattern variable set or measurement error, conflict sample data might exist. That is, two sample points with the same pattern values may belong to two different classes. In some classification techniques, it is suggested that conflict data should be removed from the sample data set [NeuroShell 1988]; however, simply deleting the conflict data will cause a loss of information.

(d) Non-representative sample.

Lack of representativeness of the underlying population in the sample may result in a large classification error rate [Hand 1981]. If the pattern data do not adequately cover the pattern space, a satisfactory classification boundary is virtually impossible to derive.

1.3. PATTERN RECOGNITION TECHNIQUES

1.3.1. REVIEW OF PATTERN RECOGNITION TECHNIQUES

Many classification techniques are available for managerial pattern recognition. In this section various commonly used classification techniques are briefly reviewed and compared.

1.3.1.1. BAYES RULE

It is well known that the Bayes classifier is optimal but that it requires full knowledge of all distributions of the prior classification probabilities in addition to the likelihood functions of the pattern variables [Krishnaiah & Karnal, 1982, p353]. The Bayes optimal principle assigns X to class c_i such that $f_{c_i}(X)*Pr(c_i)$ is maximized. For example, in the two class case:

$$Pr(c_1|X) = \frac{Pr(c_1, X)}{Pr(X)} = \frac{f_{c_1}(X)\ Pr(c_1)}{Pr(c_1)\ f_{c_1}(X) + Pr(c_2)\ f_{c_2}(X)} \qquad (1.2)$$

$$Pr(c_2|X) = \frac{Pr(c_2, X)}{Pr(X)} = \frac{f_{c_2}(X)\ Pr(c_2)}{Pr(c_1)\ f_{c_1}(X) + Pr(c_2)\ f_{c_2}(X)} \qquad (1.3)$$

We assign an observation to c_1 when $Pr(c_1|X) > Pr(c_2|X)$, that is, $f_{c_1}(X)\ Pr(c_1) > f_{c_2}(X)\ Pr(c_2)$. Otherwise, we assign it to c_2. The extension to the multiple class case is straightforward. The

classification decision boundary generated by Bayes rules is called the Bayes optimal decision boundary.

This fundamental rule minimizes the total probability of making an error [Hand 1981, p5]. Unfortunately, we rarely know the distributions $f_{c_i}(X)$ or the prior probabilities $Pr(c_i)$. To estimate these, the kernel method [Parzen 1962, Hand 1981] and k-nearest neighbour estimates method [Cover and Hart 1967] are usually used.

1.3.1.2. DISCRIMINANT FUNCTIONS

The discriminant function technique was initially developed by Fisher [1936], and is considered to be the earliest formulation of a classification technique. Later, this more intuitive approach was supplemented by probabilistic approaches involving Bayes rules [Hand 1981]. In this subsection we will discuss the linear discriminant function for the two-class problem. In Chapter 6 we will discuss problems with more than two classes.

Let b_i be the discriminant coefficient for x_i. Linear discriminant analysis develops a linear function

$$b_0 + b_1x_1 + b_2x_2 + \ldots + b_mx_m = 0$$

to separate the two classes. Then the feature vector (scalar in this case) of an object is

$$y = b_0 + b_1x_1 + b_2x_2 + \ldots + b_mx_m \quad .$$

The goal selection rule is : if y>0 classify the observed object as belonging to class c_1, otherwise, assign it to class c_2 [Lachenbruch 1975]

In Fisher's approach [Fisher 1936], using matrix notation, let μ_1 and μ_2 be the means of the pattern vectors of the two classes, respectively, and Σ be the common covariance matrix of the population, and B be the vector transpose $(b_1, b_2, \ldots b_m)'$. Fisher suggested choosing B to minimize

$$\gamma = \frac{(B'\mu_1 - B'\mu_2)^2}{B'\Sigma B} \quad . \tag{1.4}$$

Fisher's method is distribution-free in the sense that it is a reasonable criterion for constructing a linear combination. It is important to note that, in the two class case, if c_1 and c_2 are generated by bivariate normal distributions with means μ_1 and μ_2 and common covariance matrix Σ , and if the prior probabilities of c_1 and c_2 are equal, Fisher's linear discriminant function is an optimal solution from the Bayes rule point of view [Lachenbruch 1975].

There is a parallel between the linear discriminant function and multiple linear regression in the two-class case [Fisher 1936, Lachenbruch 1975, Healy 1965, Cramer 1967]. Define a predictor y_i artificially such that

$$y_s = \frac{\nu 2}{\nu 1 + \nu 2} \qquad \text{if } X_s \text{ is a member of } c_1$$

$$y_s = \frac{-\nu 1}{\nu 1 + \nu 2} \qquad \text{if } X_s \text{ is a member of } c_2$$

where $\nu 1$ and $\nu 2$ are the sample sizes of each class. The regression result is then equivalent to the linear discriminant function in this case.

If the class covariances Σ_1 and Σ_2 of the bivariate normal distributions are not equal, it can be shown that the discriminant function is quadratic, and the optimal decision boundary is quadratic [Lachenbruch 1975] [James 1985].

1.3.1.3. SEQUENTIAL CLASSIFICATION

The fundamentals of sequential classification were built from Wald's sequential analysis theory [Wald 1947] [Grometstein and Schoendorf 1982]. In the case when the data groups heavily overlap each other, the discriminatory power of the observed variables is insufficient for satisfactory assignment to c_1 or c_2 . If it is desired to avoid more than a proportion e_1 of errors in c_1 and e_2 in c_2 in the classification, and it is possible to obtain independent observations on the same object, one may use the sequential probability ratio test to assign the object to c_1 or c_2 . The classification procedure is briefly described as follows (cf.[Lachenbruch 1975]).

Determine the discriminant function B(X) from the available sample using discriminant analysis. Obtain the first observation X_1 for an individual to be classified.

If $B(X_1) \geq -\ln[e_2/(1-e_1)]$ assign to c_1 ;

If $B(\mathbf{X}_1) \leq -\ln[(1-e_2)/e_1]$ assign to c_2 ;

otherwise, take a second observation of the object. Continue the procedure i times until

$$B(\bar{\mathbf{X}}) \geq -(1/i)\ \ln[e_2/(1-e_1)] \quad \text{(assign to } c_1\text{)}$$

$$\text{or} \quad B(\bar{\mathbf{X}}) \leq -(1/i)\ \ln[(1-e_2)/e_1] \quad \text{(assign to } c_2\text{)} , \qquad (1.5)$$

where $\bar{\mathbf{X}}$ is the average $\mathbf{X}$ of the i observations.

The difficulty in obtaining replicates of $\mathbf{X}$ for the same object limits the usage of the sequential classification procedure.

There exist other methods for sequential discrimination problems. For example, instead of replicating the entire vector $\mathbf{X}$, Mallows [Mallows 1953] considered the case in which the components of $\mathbf{X}$ were observed sequentially. According to that method, observe j of the m components of $\mathbf{X}$, and

$$\text{If } \ln[f_{c_2}(x_1 \ldots x_j) / f_{c_1}(x_1 \ldots x_j)] \leq \ln[(1-e_2)/e_1]$$

assign to c_1 ;

$$\text{If } \ln[f_{c_2}(x_1 \ldots x_j) / f_{c_1}(x_1 \ldots x_j)] \geq \ln[e_2/(1-e_1)]$$

assign to c_1 ;

otherwise observe x_{j+1} . The best order in which to observe the components of $\mathbf{X}$ should be pre-defined.

The major problem in using sequential classification in the present context of managerial problems relates to assumptions about the probability distribution functions.

1.3.1.4. HIERARCHICAL CLASSIFICATION

Differing from the Bayes rule and discriminant functions, hierarchical classification methods are based on an explicit definition of properties which should largely be satisfied by members of a class, thus facilitating the subsequent activity of assigning new objects to the classes [Gordon 1981]. A hierarchical classifier uses a tree structure to implement the classification. The essential concept is shown in Figure 1.5.

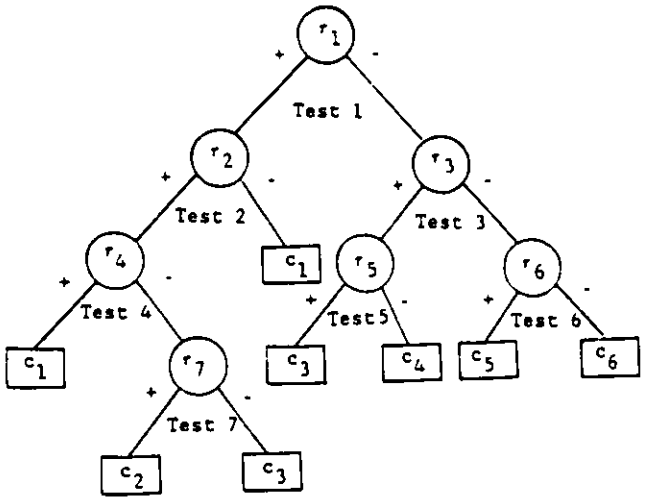


Fig. 1.5. Hierarchical Classifier

r_1 in the figure represents an object with a certain pattern to be classified. After testing by a set of questions it can be subclassified into either r_2 if the answer is positive, or r_3 if the answer is negative. This procedure continues until the tree growth is terminated. For most real problems there is no definite "yes" and

"no" answer. Breiman et al. [Breiman et al. 1984] suggested defining an impurity function Γ which is used to measure the "purity" on a set of pre-defined properties at a node. At any node τ_X , suppose that there is a candidate split ℓ of the node which divides it into τ_L and τ_R such that a proportion p_L of the cases at τ_X go into τ_L and a proportion p_R go into τ_R (Figure 1.6.).

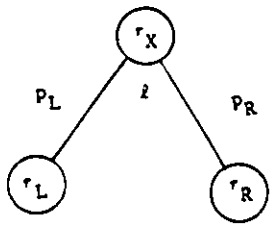


Fig. 1.6. A Split of The Classification Tree

Then the goodness of the split is defined to be the decrease in impurity

$$\Delta\Gamma(\psi,\tau) = \Gamma(\tau) - p_L\,\Gamma(\tau_L) - p_R\,\Gamma(\tau_R) \quad . \tag{1.6}$$

Actually, the split is not necessarily binary and may be, say, ternary. The additional branch would indicate "undecided" and a need for sequential sampling.

Usable information regarding the structure of the data is obtained through the hierarchical method. However, a set of test questions as well as the criteria for subclassification must be developed, implying a certain amount of knowledge about relationships among subclasses.

1.3.1.5. EXPERT SYSTEMS

An expert system is a computer-based system that uses knowledge, facts, and reasoning techniques to solve problems that normally require the abilities of human experts [Martin & Oxman 1988]. A major difference between expert systems and traditional data processing systems is that, in expert systems, the knowledge base is separated from the inference engine [Hayes-Roth et al. 1983]. For pattern recognition applications, the knowledge base representation schemes used for expert systems are based on either production rules or frames, as follows.

1.3.1.5.1. PRODUCTION RULE SYSTEM FOR PATTERN RECOGNITION

Classification knowledge in production rule systems is represented in the "IF-THEN" form. For instance, in an expert system for judging company credit status, two production rules could be :

IF the financial ratios are extremely low,
THEN the company is "Bad".
IF the financial ratios are extremely high,
THEN the company is "Good".

Expert knowledge can also be represented in uncertainty form. For example,

IF the cash turnover rate is lower than 10,
THEN the company is probably (90%) "Bad".

Production rules are accessed and reasoning is accomplished by the inference engine, with forward chaining and backward chaining

as two commonly used inference strategies. In general, backward chaining is applied when a goal or a hypothesis is chosen as the starting point for the problem, and forward chaining is applied when data is to be used as the starting point for problem solving. A production rule system is diagrammed in Figure 1.7.

Examples where production rules have been applied in classification include [Wolfgram et al. 1987] and [Balachandra 1988].

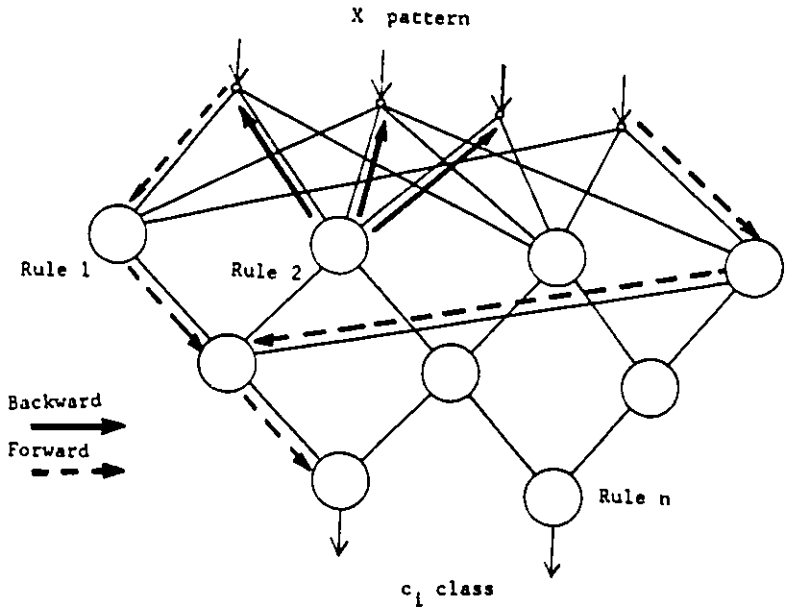


Fig. 1.7. Production Rule System for Pattern Recognition

1.3.1.5.2. FRAME-BASED SYSTEMS FOR PATTERN RECOGNITION

It is widely believed that knowledge is organized in structured chunks which have their own sub-structure. The data-structure of knowledge representations is a frame or a template which represents a class [Minsky 1975]. A frame has terminals or "slots" that must be filled by specific instances or data. For example, an object pattern "Good-company" would have slots representing the attributes of "Good-repay-ability", "High-revenue", and others. A collection of related frames is linked together into a frame system (see Figure 1.8).

Frame-based expert systems use inference to perform assertion and retrieval operations. Commonly used inference methods include inheritance, and value class and cardinality reasoning [Fikes and Kehler 1985].

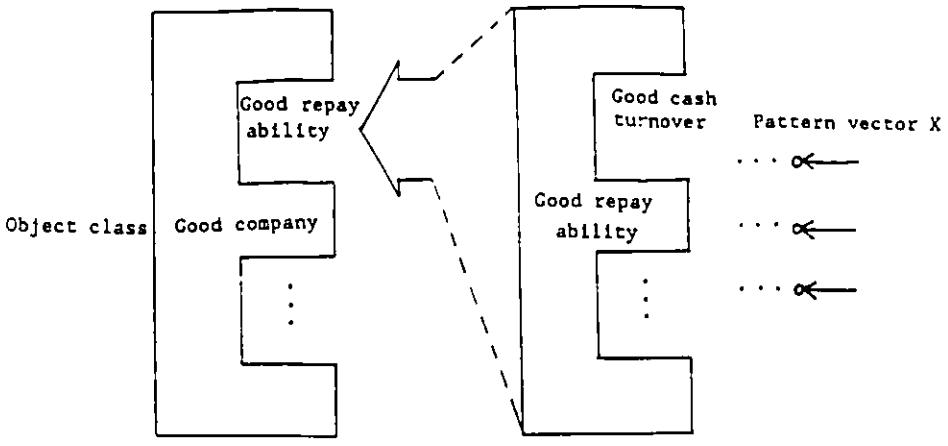


Fig. 1.8. Frame Representation of Pattern Recognition

1.3.1.5.3. PROPERTIES OF EXPERT SYSTEMS IN PATTERN RECOGNITION

An expert system is usually efficient for solving pattern recognition problems if the required knowledge is available. However, the potential number of rules increases factorially with the number of pattern dimensions. If the pattern vectors are expressed as real numbers, the rule set is theoretically unlimited, although categories could be used to limit the number of rules required. More importantly, explicit knowledge is often not available to cover all possibilities for managerial pattern recognition. Finally, sometimes uncertainty must be reflected in the knowledge base, but there is little agreement in how to support this [Hayes-Roth et al. 1983] [Hutchison 1987].

1.3.1.6. NEURAL NETWORK TECHNIQUES

A layered neural network [Nilsson 1965] [Rosenblatt 1962] [Rumelhart et al. 1986] (Figure 1.9) is a special version of the general neural network model described in Figure 1.1. It is commonly used in classification applications [Lippmann 1987].

In this section we briefly discuss the general properties of the layered neural network (neural network for short) to prepare for a comparison of various classification techniques. Chapter 2 will give a more detailed description of layered neural network mechanisms.

The simplest neural network which contains just one node, except for input nodes, is a perceptron [Nilsson 1965] [Minsky &

Papert 1969]. A perceptron (Figure 1.10) receives an object pattern vector **X**, and assigns a weight w_i to each component x_i.

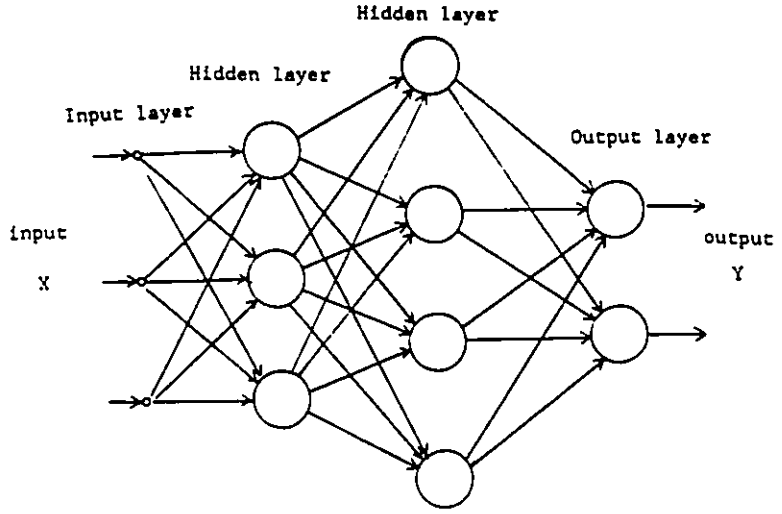


Fig. 1.9. Layered Neural Network Model

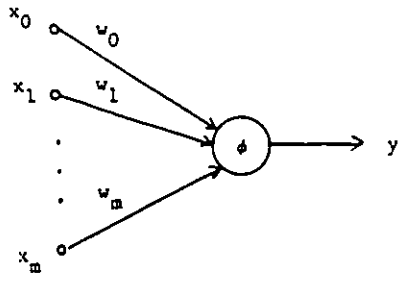


Fig. 1.10. Perceptron Model

The weighted components are processed by a ϕ-processor giving an output value of y. ϕ may be represented by a variety of functions. The simple "standard" form of the ϕ-processor is a linear combination of the $w_i x_i$. It is not difficult to show that, if

$y = w_0x_0 + w_1x_1 + \ldots + w_mx_m$, the perceptron is equivalent to a linear discriminant function classifier. However, the determination of the weight w_i in a perceptron is by a learning technique (discussed in Chapter 2) which is of course different from Fisher's method. Thus, the functions generated by the two approaches will usually not be the same. When operating as a linear discriminant classifier, the perceptron generates a linear boundary. Figure 1.11 shows a linear decision boundary generated from a case with two classes, and with a two-dimensional pattern vector.

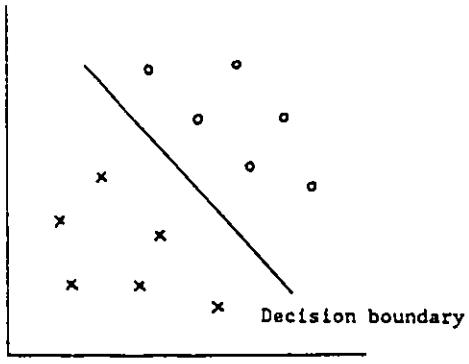


Fig. 1.11. Linear Decision Boundary

A standard perceptron cannot solve more complicated pattern recognition problems such as the XOR (exclusive or) problem [Rumelhart et al. 1986] (Figure 1.12), which has a non-linear decision boundary. However, a multi-layer neural network can generate arbitrarily complex decision regions (Figure 1.13) [Lippmann 1987]. Such a network includes one or more "hidden" layers of nodes between the input and output nodes.

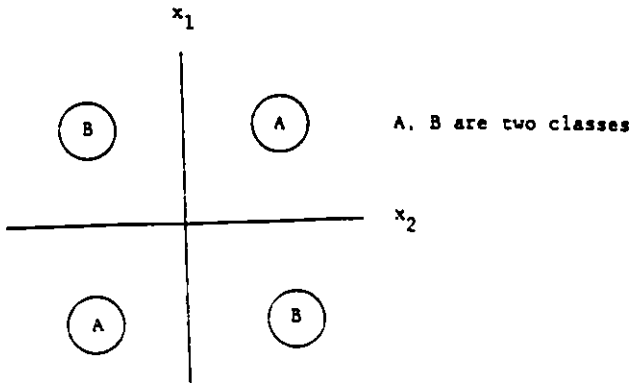


Fig. 1.12. XOR (Exclusive OR) Problem

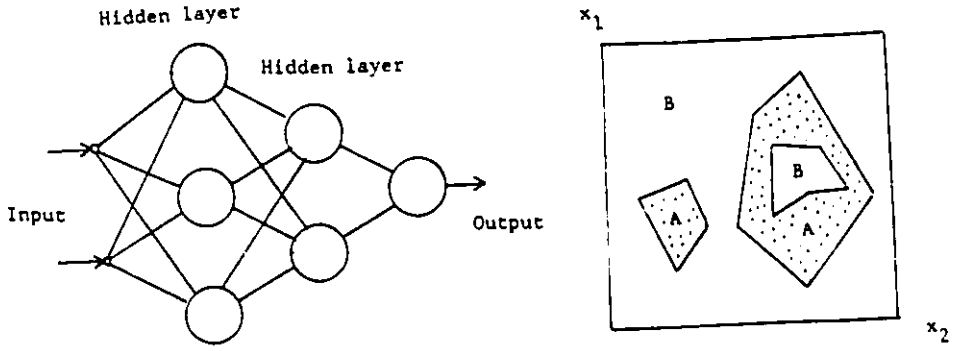


Fig. 1.13. A Multi-Layer Neural Network Can Generate Arbitrarily Complex Decision Regions (adapted from [Lippmann 1987])

When the number of hidden layers is greater than 3, the neural network tends to become unstable [Soulie et al. 1987]; that is, the network component weights are difficult to establish during the learning process. On the other hand, Kolmogorov's mapping neural

network existence theorem [Hecht-Nielsen 1987] asserts that, given any continuous function $Y=\phi(X)$, ϕ can be implemented exactly by a one-hidden-layer neural network having m input elements, (2m+1) processing elements in the hidden layer, and n processing elements in the output layer, if the processing elements implement one of a specific class of transformation functions. Such a one-hidden-layer neural network can therefore handle any level of difficulty in classification, and there is no need in principle to consider more hidden layers for pattern recognition problems (* see Note at the end of this chapter).

How to determine the weights for node inter-connections, and what network architecture to use (i.e. numbers of layers and nodes) depends upon the specific problem to be solved. It has been shown [Minsky & Papert 1969] [Rumelhart et al. 1986]) that if a specific architecture of neural network has been selected, (say, three-layer with 10 nodes in the first layer, 20 nodes in the second layer, and so on,) then the weight of each connection can be determined from training sample data. Generally, these weights can be adjusted in such a way that the neural network is "optimally" improved in terms of matching the training samples' input and output (see Chapter 2).

Literature is rare on the application of neural networks in managerial classification decision making, but this is partly due to secrecy surrounding this type of research [O'Reilly 1989]. However, there have been many reports of pattern recognition applications in other fields. For example, Widrow et al. [1988] built a neural network called ADALINES. After learning a training set consisting of 36 features, each arranged on a 5 X 5 grid, the system made zero

errors in classifying the training data set. Lippmann and Gold [Lippmann and Gold 1987] used a three-layer neural network for isolated-word speech recognition. The neural network was trained with fewer than 1000 examples. The result was better than 99% accuracy in classifying speech pattern from a large speech data base with 35 difficult words, 9 talkers, and a total of 4095 test tokens. Guyon et al. [Guyon et al. 1989] used a neural network to classify handwritten digits, and claimed an accuracy of 98 percent for the test data set.

Three important points should be noted: First, classification techniques other than the neural network technique are not applicable to many pattern recognition problems due to the lack of knowledge about distribution and classification data structure, the large number of goal classes, and, most important, the potentially complex classification region boundaries. Second, a neural network model almost guarantees good separation in training data sets [Rumelhart et al. 1986], but has mixed results for classification accuracy of test data sets [Lehar and Weaver 1987] [O'Reilly 1989]. Finally, in order to improve its performance, the neural network is often used in combination with other techniques which are more specific to the problem domain. For example, a handwritten digit classification neural network is usually supported by a pre-processor for pattern extraction [Guyon et al. 1989]. This suggests that a neural network performs better if it is combined with other techniques which use domain-specific knowledge. Therefore, one of the current research tasks will be to investigate the generic characteristics of managerial classification problems,

and to develop methodologies to aid the neural network mechanism in achieving improved classification results.

A summary of the classification techniques discussed above is shown in Table 1.1.

Model Type	Technique	Assumption/Condition	Internal model	Implementation	Classification Criteria	Restrictions	References
Statistical	Bayes rule	Prior probabilities $Pr(c_i)$ and density functions $f_{c_i}(X)$ are known		Kernel method K nn method	Maximize $f_{c_i}(X)Pr(c_i)$	Estimates of $f_{c_i}(X)$, $Pr(c_i)$ needed	Hand 1981 Parzen 1962 Krishnaiah & Kanal 1982 James 1985
	Discriminant analysis	Decision function form is known		Linear/ quadratic functions	Minimize (between-group variance / within-group variance)	Decision boundary is assumed	Fisher 1936 Lachenbruch 1975 Young 1972
	Sequential classification	Probability functions are known	Discriminant functions	Sequentially sampling	Sequential probability ratio falls into desired ranges	Estimates of probability distribution functions needed	Wald 1947 Mallows 1953
	Hierarchical classifier	A set of questions for split and impurity function are required	Tree rep question structure	Binary tree split	Minimize the value of impurity function	Impurity functions and a set of questions are required	Gordon 1981 Breiman et al. 1984
Expert system	Production rule	A set of rules is required Uncertainty factors may be estimated	Tree/ network rep. relationship between rules	Forward/ backward reasoning	Production rules reasoning	1. Complete rule set is needed 2. Poor at dealing with numerical pattern data 3. Fuzzy logic difficult to implement	Martin and Oxman 1988 Hays-Roth et al. 1981 Hutchison1981 Wolfgram et al. 1987
	Frame-based system	Frames (knowledge structure) are known	Tree/ semantic net rep knowledge structure	Inheritance method, value class and cardinality reasoning	Frames inferencing	1. Complete frames needed 2. Poor at dealing with numerical pattern data 3. Uncertainty problem	Martin and Oxman 1988 Minsky 1975 Fikes and Kehler 1985
Neural network	Neural network		Network of function nodes, with feedback	Back-propagation learning	Training samples fall into desired classes	Predictability, repeatability problems	Rosenblatt1962 Nilsson 1965 Minsky and Papert 1969 Rumelhart 1986 Kohonen 1987 Lippmann 1987

Table 1.1. Comparison of Classification Techniques

1.3.2. COMPARISON OF CLASSIFICATION TECHNIQUES

A classification problem is a synthesis problem; that is, given a system's input and feature output, a model having the desired input / output relationship is to be designed or realized (cf. [Karplus 1983]). The various classification techniques discussed above are all means of modeling the classifier that best reflects the relationship between pattern input and feature output in a certain context. In Figure 1.14, input and output are specified but the contents of the system are unknown; hence it is "black". In modeling the system, there are constraints which represent the available knowledge about the system. If more knowledge is available about the system's internal structure, then the system is more "transparent" in terms of developing a suitable model of its performance.

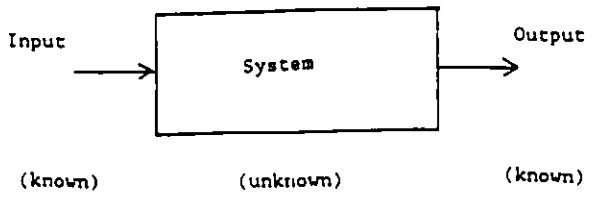


Fig. 1.14. Synthesis Problem

As discussed above, individual classification techniques may require more or less knowledge about the system to be modeled. Consequently, these classification techniques cover a range in a spectrum which exhibits the "transparency" of the system, as shown in Figure 1.15. As depicted, working from the black box end of the spectrum to the glass box end, a neural network classifies an object

using only training data. Discriminant analysis needs an additional assumption of the form of the function which separates the classes. Bayes rules require prior probabilities and likelihood functions as an underlying optimization condition for the discriminant analysis method. Bayes and discriminant analysis classifiers are both parametric classifiers in that parameters belonging to known distribution functions are being estimated [Kohonen, 1977, p98]. Hierarchical classification explores the detailed nature of why an object belongs to a class, and implicitly defines the pattern's distribution. Expert systems explicitly define all knowledge about the classification system in the form of production rules or frames. Consequently, the methods closer to the glass box end explicitly or implicitly cumulate the knowledge required by the methods closer to the black box end, and more clearly provide insight and understanding of why an observation should belong to its class.

Knowledge about the system initially comes from input/output observations. No matter how detailed and reliable the observations, the validity of the knowledge is always open to question [Karplus 1983]. In managerial pattern recognition problems, patterns are generated by multi-dimensional random variables, the pattern structures are unpredictable, and the forms of the multivariate distribution functions are unknown. Discriminant analysis in combination with neural networks can be applied in solving synthesis problems of this nature, as discussed below.

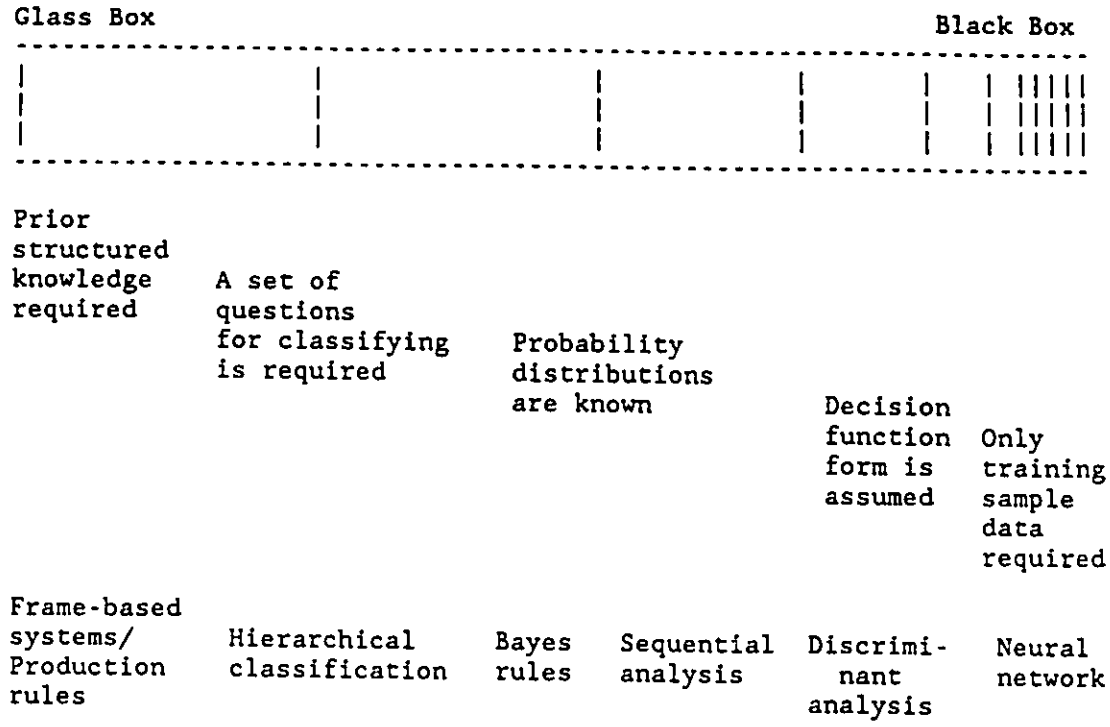


Fig. 1.15. The Position of Various Classification Techniques in the "Glass Box - Black Box" Spectrum

1.3.3. PRIMARY COMPARISON OF DISCRIMINANT ANALYSIS AND NEURAL NETWORKS

In this subsection we give a primary comparison of discriminant analysis and neural networks to show how a neural network can be used to solve classification problems without the assumptions inherent in the use of discriminant analysis.

1.3.3.1. LIMITATIONS OF DISCRIMINANT ANALYSIS

Certain problems have been described [Eisenbeis 1977, 1978] [Morrison 1969] [Frank et al. 1965] [Joy and Tollefson 1975] in

applying discriminant analysis to managerial decisions involving classification. The most serious concerns about discriminant analysis arise from assumptions about the decision function form.

As described in Section 1.3.1.2, linear discriminant functions assume that the decision boundary is linear. However, the linear decision boundary is optimal only if the pattern vector is multivariate normally distributed, and the covariance matrices are equal across all classes. Similarly, quadratic discriminant functions assume that the decision boundary is quadratic, and it is optimal only if the pattern variables are multivariate normal distributions; however they can have different covariance matrices. Though some studies (e.g. the examples cited in Section 1.2) obtained good results using discriminant analysis methods, most of them did not verify the distribution conditions. Good results obtained could be due to the inherent normality of the distributions, robustness of the technique to deviations from normality, or chance.

If the normality hypothesis is rejected, one is then faced with a virtually impossible task of first determining the form of the actual function, and secondly deriving an appropriate alternative solution technique based on statistical theory. Most researchers adopt the standard discriminant procedure and proceed as if the normality assumption holds, as long as it yields a reasonably accurate result [Eisenbeis 1977].

On the other hand, the boundary classification function for managerial problems is often much more complicated than linear or quadratic (see, e.g., [Keeney 1972]). Existing discriminant analysis techniques cannot deal effectively with this problem.

1.3.3.2. AN EXAMPLE FOR COMPARISON OF DISCRIMINANT ANALYSIS AND NEURAL NETWORK TECHNIQUES

In order to show that a neural network could be appropriate for solving managerial pattern recognition problems, an experiment was conducted. The experiment demonstrated that a neural network using the BPLMS algorithm (discussed in Chapter 2) was able to obtain more accurate classification results than discriminant analysis, particularly when sample distribution normality was violated. For normally distributed pattern data, discriminant analysis obtains the Bayes optimal boundary (see Section 1.3.1.2.). However, any sample data with a nonlinear and non-quadratic classification boundary, such as exponential, logarithmic, trigonometric etc., cannot be classified optimally by discriminant analysis.

For the experiment, a training data set was generated from a pattern with a bivariate normal distribution and the same covariance matrix [Law and Kelton 1982] [Scheuer and Stoller 1962]. This population was then shifted into two classes with different means on both variates. Thus far the Bayes optimal decision boundary between the two classes is a straight line as discussed earlier, with the form

$$b_0 + b_1 x_1 + b_2 x_2 = 0 \quad .$$

To change the shape of the Bayes optimal decision boundary we simply transform one or both of the coordinates as desired. For instance, if an exponential boundary is desired,

let $x_1 = e^{x_1'}$; then the optimal discriminant function becomes

$$b_0 + b_1 e^{x_1'} + b_2 x_2 = 0 \quad .$$

After this transformation (see Figure 1.16) the population distribution does not fit the normality requirements of discriminant analysis. A full explanation of the sampling procedure is exhibited in Appendix I.

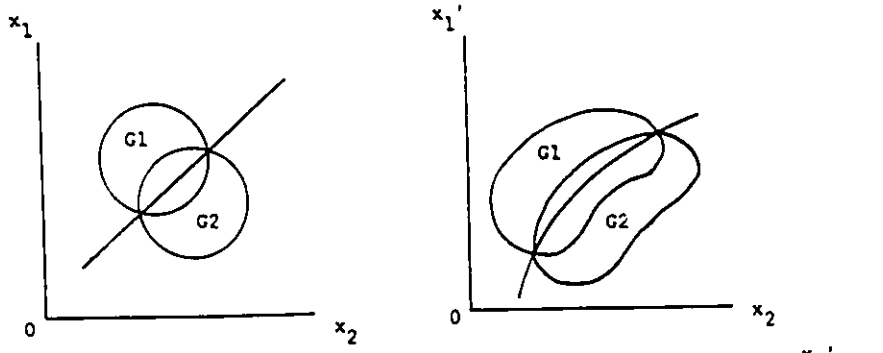


Fig. 1.16. Coordinate Transformation

The size of the training set was 40 (20 in each class; see the plotted points in Figure 1.17).

Linear discriminant analysis (LDA) of the transformed data gives a classification accuracy of 37/40 (92.5%); that is, three points are misclassified (see Table 1.2).

Linear Discriminant Analysis

Group	0	1
Count	20	20

N = 40 N Correct = 37 Prop. Correct = 0.925

Linear Discriminant Function for Group

	0	1
Constant	-2.999	-3.308
x1	-7.738	18.000
x2	22.224	-11.145

Table 1.2. Discriminant Analysis for the Exponential Boundary Example

For the same data set, the neural network with 5 hidden nodes obtained an accuracy of 39/40 (97.5%) after 2450 learning sweeps. The boundary line obtained (NNa in Figure 1.17) has an exponential shape (compare it with the true boundary). After 37655 learning sweeps it completely separated the two classes, with significant boundary distortion (NNb in Figure 1.17). Note that this leads to difficulties in implementing the standard BPLMS neural network approach where there is a great deal of overlap between two or more data classes. The technique must therefore be adapted before it can be used to routinely solve this type of problem, and this is the main topic of the remainder of the thesis.

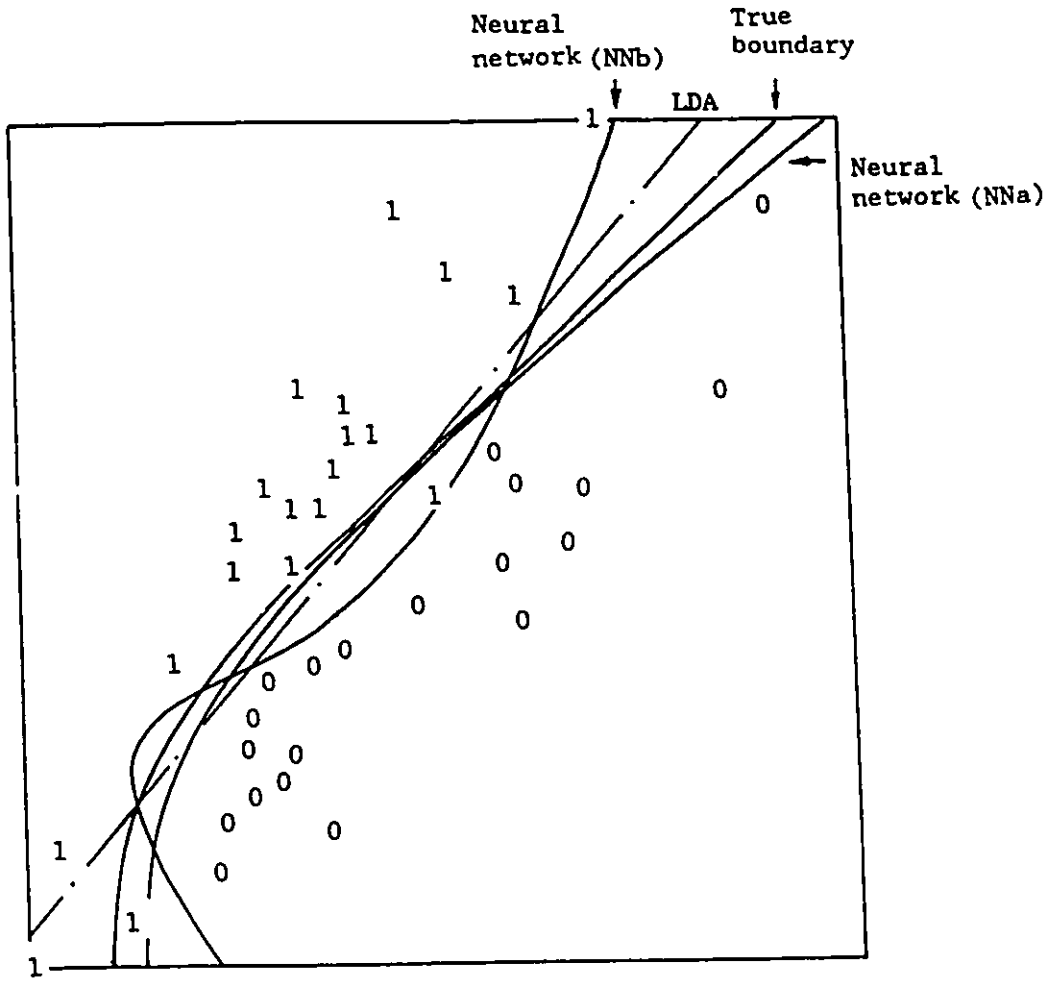


Fig. 1.17. Neural Network Results for the Exponential Boundary Example

*) **NOTE** (See Section 1.3.1.6)

This note explains the applicability of Kolmogorov's mapping theorem [Hecht-Nielsen 1987] [Sprecher 1965] in pattern recognition in the sense that Rumelhart's backpropagation neural network ([Rumelhart et al. 1986], see Chapter 2) may carry out an approximation of a mathematical mapping function $\mathbf{Y}=\phi(\mathbf{X})$. A more complete mathematical proof, which demonstrated that continuous nonlinear mapping can be closely approximated using sigmoidal nonlinearities and layered neural network with only one hidden layer, is found in [Cybenko 1989].

From [Hecht-Nielsen 1987] [Sprecher 1965], the second layer (hidden layer) of a neural network implements the transfer function

$$L_i = \sum_{j=1}^{m} u^i \psi (x_j + \epsilon i) + i$$

where m is the dimensionality of $\mathbf{X}$, $0 \le i \le 2m$, the real constant u and the continuous real monotonic increasing function ψ are independent of ϕ, and ϵ is an arbitrarily chosen positive constant.

The L_i, which are independent of ϕ, can be approximately represented by a sigmoid function (Chapter 2) when a proper parameter temperature (Chapter 2) is selected.

In Kolmogorov's theorem, the output elements have the transfer functions

$$y_j = \sum_{i=0}^{2m} \Pi_j (L_i)$$

where the Π_j (j=1.. n, n is the number of output nodes) are real and continuous and depend on ϕ and ϵ. Π cannot be represented by a prior known function because of its dependence on ϕ.

Assume L_i and y_j range over the interval [0,1]. For each L_i divide the interval [0,1] into small intervals Δq^g, g=1...∞. Let

$$y_j = x_j^g (L_0 \cdots L_{2m}) \quad \text{for interval } \Delta q^g; \text{ and}$$
$$= 0 \quad \text{otherwise}$$

for all j=1...n, where $x_j^g(L_0 \ldots L_{2m})$ is an approximation of $\sum_{i=0}^{2m} \Pi_j (L_i)$ in the interval Δq^g such that $x_j^g(L_0 \ldots L_{2m})$ is represented by the sigmoid logic function (Chapter 2). The interpretation of this approximation is that any output y_j can be approximated by a group of g output nodes which perform a sigmoid logic function as long as g is large enough, which was to be shown.

Note that the above approximation has introduced $\mathbf{Y}=\underline{0}$ in addition to the original y_j. In the classification context, considering that the classification boundary is usually defined at $y_j \ge 0.5$ (Chapter 3), the introduced function $\mathbf{Y}=\underline{0}$ does not influence the classification results, and can therefore be neglected.

CHAPTER TWO

THE NEURAL NETWORK MODEL

As pointed out earlier, the layered neural network is the most commonly used form of the neural network model for pattern recognition. In managerial pattern recognition, the major task of classification is to find a decision boundary based on a sample data set which includes information that specifies the correct class for each observation in the set. Consequently, supervised training can be used to train neural networks in classification. Among four groups of neural network classifiers, namely, probabilistic classifiers, hyperplane classifiers, kernel classifiers, and exemplar classifiers [Lippmann 1989], hyperplane classifiers are more applicable in most managerial classification problems since prior knowledge about probability distributions, kernel functions, or typical exemplars is often not available. As well, the results of managerial classification should be expressible in continuous quantitative form instead of binary 0/1 form, in order to supply the decision maker with more detailed information. The multi-layer neural network which uses the back-propagation algorithm (explained later) was therefore chosen in the present research. Thus, in the remainder of the thesis, references to neural network models are

restricted to the class of layered neural networks which use supervised training and apply the back propagation least mean square algorithm (explained later) or a modification of this algorithm during training. In this chapter we first illustrate three major aspects of this model:

(a) general topology,

(b) computational elements, and

(c) learning.

Then we will discuss the current status of neural network design in general.

2.1. OVERVIEW

The general topology of a layered neural network has been presented in Figure 1.9. Figure 2.1 shows a neural network with one hidden layer to illustrate how a neural network is able to do pattern recognition.

In Figure 2.1, there are three columns of nodes. The nodes have direct connections only to adjacent columns. The connection between the nodes carries weight (or synaptic strength) indicating the connection strength. The weights between two adjacent columns comprise the weight matrices. W^1 is the weight matrix between the output nodes and nodes in the hidden layer, and W^2 is the weight matrix between the hidden layer and input nodes. Because input nodes do not perform any transformation, a neural network with one hidden layer is commonly called a two-layer neural network. Each node in

the hidden layer or the output layer includes a function F (described in Section 2.2) to generate the input-output transformation at that node. Such a node is therefore referred to as a processing node. In this model, the functional form of F is the same for all processing nodes.

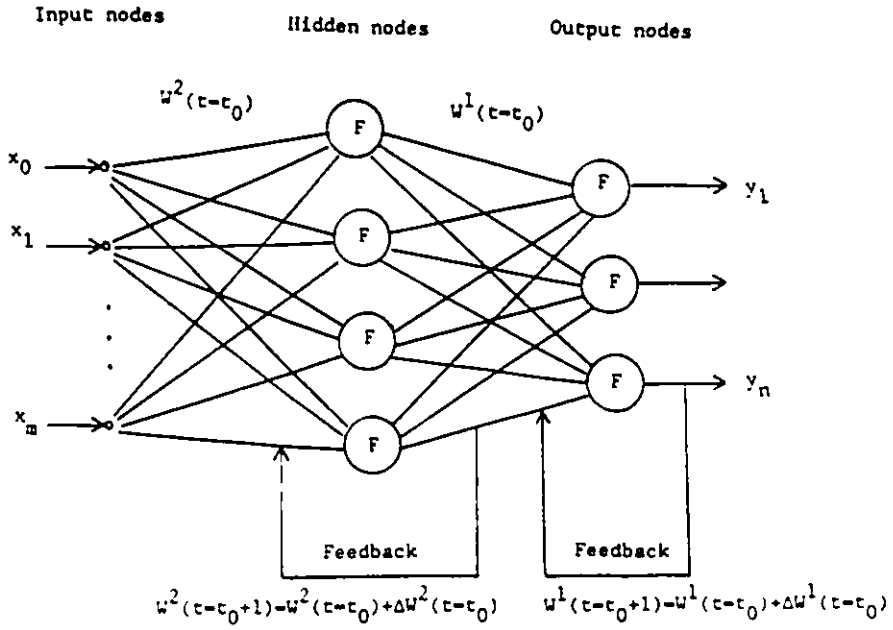


Fig. 2.1. Neural Network with One Hidden Layer

The ultimate goal in developing a neural network model through training is to adjust the network weights so that the neural network's outputs are "close" in some sense to the desired outputs for a given set of training data. This is accomplished by means of an iterative process which starts with a pre-determined set of weights. These weights are adjusted through cycles of the training algorithm which operates on the training data pattern vectors to

generate classifications. These classifications are compared with the "true" results, and weights are adjusted accordingly, until the neural network outputs correspond as closely as desired to the true results.

During the training process, at step $t=t_0$, the weights of $W^1(t=t_0)$, $W^2(t=t_0)$ are arbitrarily assigned as long as they are not all equal (the symmetry-breaking problem will be explained in Section 2.3). Each component of **X**, multiplied by the corresponding weight, simultaneously feeds into the hidden nodes. Each hidden node combines all of its inputs which are then transformed into an output by the function operator. The outputs of hidden nodes are in turn multiplied by the corresponding weights, and collected and transformed at each node of the output layer, finally giving the output **Y**.

At step $t=t_0$ we have a mapping function $\phi(t=t_0)$ which maps **X** to **Y**. Suppose we input pattern data $\mathbf{X}_s$ of a teaching sample point $s \in S$ into the neural network $\phi(t=t_0)$, to generate the corresponding classification result $Y_s(t=t_0)$. In order to obtain specific feedback information from s, $Y_s(t=t_0)$ is compared with the desired feature vector value $\mathbf{T}_s$. Let

$$E_s(t=t_0) = \frac{1}{2} \| Y_s(t=t_0) - \mathbf{T}_s \|^2 \qquad (2.1)$$

to measure the discrepancy between Y_s and T_s at step $t=t_0$, where $||V||^2$ is the inner product of V. If $E_s(t=t_0)<e$, where e is a pre-defined error tolerance which is dependent on the class selection function (Section 1.2.1), then Y_s is considered to be consistent with T_s at step $t=t_0$. If for all $s\epsilon S$ the Y_s are consistent with T_s , then the neural network representing the mapping function ϕ is the "right" classifier for S.

If $\phi(t=t_0)$ does not give the desired mapping, then $E_s>e$. Based on the error E_s, the network weights are then modified to improve the relationship between the pattern values and the desired classification results. The modification of weights in the layered neural network model proceeds by a feedback mechanism from the output layer back to the input layer. That is, $W^1(t=t_0+1)$ is calculated first, based on $W^1(t=t_0)$ and E_s, and then $W^2(t=t_0+1)$ is calculated, based on $W^2(t=t_0)$ and the change $\Delta W^1(t)$ (Figure 2.1.). The weight modification process described above is called back propagation learning. The detailed algorithm will be described in Section 2.3. There is no general convergence proof for back propagation learning in neural network models. We will return to this point later.

Determining the weights for a multi-layered neural network can be carried out by extending the principles discussed above.

2.2. COMPUTATIONAL NETWORK ELEMENTS

In a layered neural network each node of the hidden layer and the output layer is a computational element which transforms its input into an output (Figure 2.2). The commonly used functional form of the computational element α to calculate its output o_α is

$$o_\alpha = F\left(\sum_\beta w_{\alpha\beta}\, o_\beta - \theta\right)$$

where β is any node preceding α; $w_{\alpha\beta}$ is the weight at the connection between nodes β and α; θ is a parameter to be described; o_α and o_β are the output of nodes α and β, respectively; F, the activation function, produces the input/output transformation [Rumelhart et al. 1986]. A positive input weight represents an excitatory input, and a negative weight represents an inhibitory input. The summed input must exceed some threshold value θ before contributing to the activation state of the node. θ is essentially another kind of node input. For convenience, an artificial node (β=0) with output $o_0 = 1$, and $w_{\alpha 0} = -\theta$ can be used in the computational function, resulting in the general representation of a node shown in Figure 2.2b .

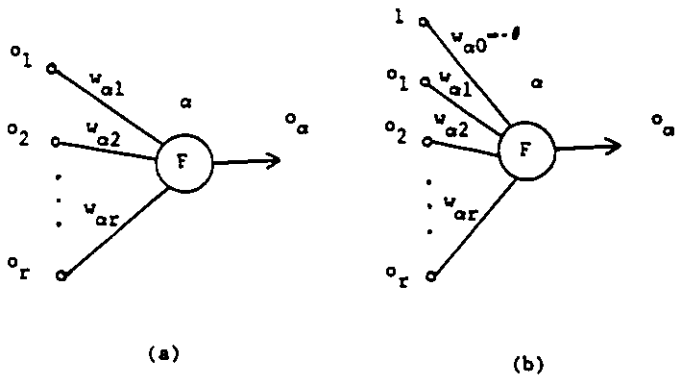


Fig 2.2. Computational Elements in a Neural Network

The three most commonly used activation functions (Figure 2.3) [Lippmann 1987] [Williams 1986] are described in the following.

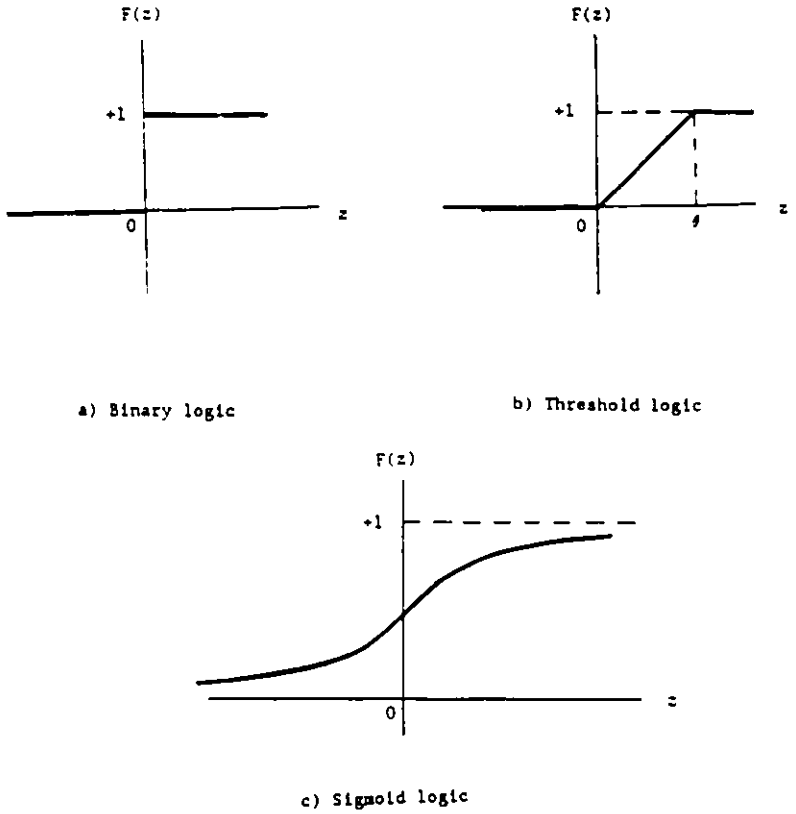


Fig. 2.3. Activation Functions

(1) Binary logic.

Binary logic (or hard limiter) was initially used in the simple perceptron [Rosenblatt 1962] [Minsky and Papert 1969]. Binary logic produces either a 0 or 1 (see Figure 2.3(a)). It is widely used in physical pattern recognition because the pixel unit in a display is usually represented in the binary form of black or white.

(2) Threshold logic

Threshold logic is a variant of binary logic. If this logic is used, the output is 0 until the input magnitude reaches a certain value θ, when it switches to 1 (Figure 2.3 (b)).

(3) Sigmoid logic

Sigmoid logic (Figure 2.3(c)) was initially used in the so-called thermodynamic models [Rumelhart et al. 1986]. If this logic is used, the output values are a stochastic function of the inputs. That is

$$o_\alpha(t) = \Pr(o_\alpha(t)=1) = \frac{1}{1+\exp(-net_\alpha / T)} \qquad (2.2)$$

where $net_\alpha = \sum_\beta w_{\alpha\beta}\, o_\beta$ and is the net input for α; T is the temperature, which determines the slope of the probability function. When T approaches zero, the logic becomes more and more like binary logic. Whenever the node has a summed net input greater than 0, the probability that $o_\alpha(t)=1$ is greater than 0.5. T sets the range of uncertainty which determines the value of $o_\alpha(t)$ (Figure 2.4).

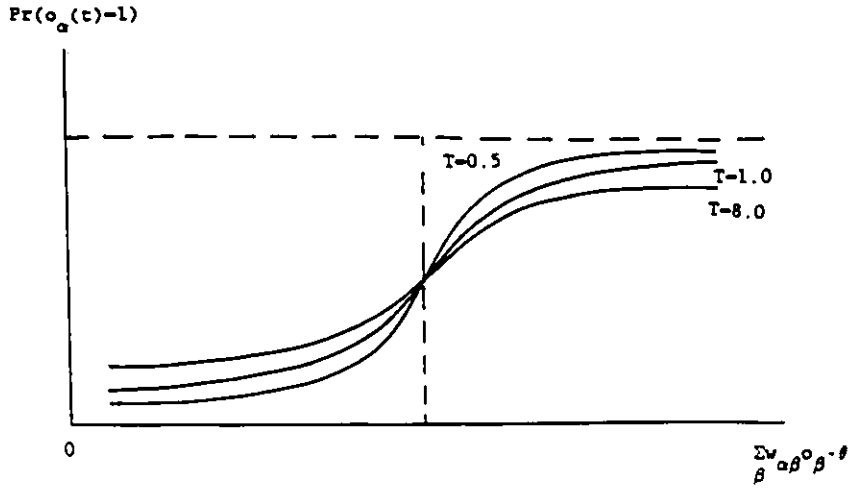


Fig. 2.4. Sigmoid Logic

Sigmoid logic has the useful property of being differentiable everywhere. Later it will be seen that this property is essential in implementing the back propagation learning algorithm, which is suitable for managerial pattern recognition problems. Hence, sigmoid logic will be employed exclusively in the present research.

2.3. LEARNING

2.3.1. THE LEARNING PARADIGM AND THE HEBBIAN RULE

As described in Section 2.1., the goal of the training exercise is to learn (adjust the network weights) to correctly classify each sample s so that when any of the set S is presented in the future, the system will classify it properly. This learning paradigm is called supervised learning, or learning with a teacher.

While this research will use supervised learning techniques, there are several other common learning paradigms in neural network systems [Nilsson 1965] [Rumelhart et al. 1986]. The typical opposite of supervised learning is unsupervised learning or competitive learning [Rumelhart et al. 1986]. When the unsupervised learning paradigm is used, there is a population of patterns from which patterns are selected at random. During the training process, the system is supposed to discover salient features of the input population. Unlike supervised learning, there is no a priori set of classes into which the patterns are to be classified. The system must develop its own feature representation of the input stimuli without benefit of a teacher.

Machine learning was first explored by Hebb [Hebb 1949] (cf. [Nilsson 1965]). Hebb's essential idea was that, if two nodes are both active, then the weight between them should be strengthened. Otherwise, the weight should be weakened. This idea has been adopted and extended in supervised learning applications. A standard formulation of Hebbian supervised learning in layered neural networks is [Rumelhart et al. 1986] :

$$(\Delta w_{\alpha\beta})_s = \eta \, [\, (d_\alpha)_s - (o_\alpha)_s \,] \, (o_\beta)_s \qquad (2.3)$$

where s is a teaching sample point, $w_{\alpha\beta}$ is the weight at the connection from node β to α, $(\Delta w_{\alpha\beta})_s$ is the change to be made to the weight after s is presented, $(d_\alpha)_s$ is the desired output of node α for s, $(o_\alpha)_s$ is the actual output of node α for s, $(o_\beta)_s$ is the output value of node β for s, and η is the constant of

proportionality representing the learning rate. Figure 2.5 is a feedback diagram depicting this learning rule.

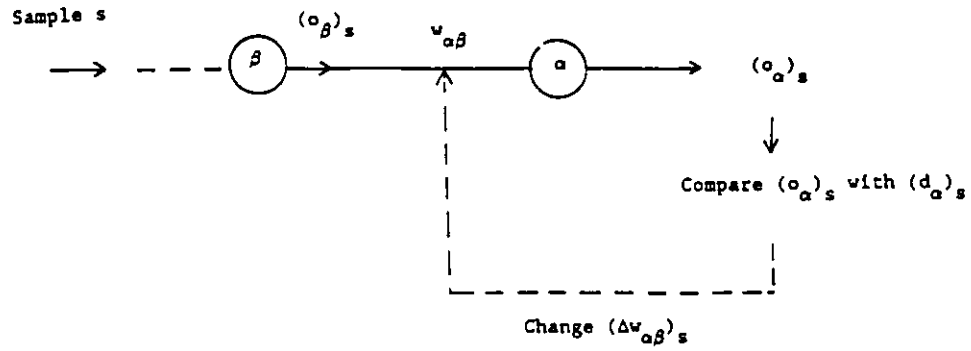


Fig. 2.5. Hebbian Learning Rule

2.3.2. BACK-PROPAGATION LEAST MEAN SQUARE ERROR LEARNING ALGORITHM

The back-propagation least mean square error learning algorithm (BPLMS) integrates two fundamental learning algorithms. One is the Least Mean Square (LMS) error algorithm suggested by Widrow and Hoff [Widrow and Hoff 1960] (cf. [Widrow et al. 1988]). This algorithm minimizes the sum of squares of the errors over the training set, where the error is defined as the difference between the desired response and the actual output. It supplies local error information for the neural network but it does not specify how to pass the error information from layer to layer. The back-propagation (BP) algorithm [Rosenblatt 1962] specifies a procedure which passes the error information (see equation 2.3) back through the network from the output layer through the hidden layers and to the input layer, resulting in adjustments to the weight matrices. Rumelhart et

al. [Rumelhart et al. 1986] generalized and integrated these two algorithms into a multilayered nonlinear activation function neural network. This BPLMS learning algorithm (BP for short) is currently widely used in pattern recognition.

The derivation of the algorithm [Rumelhart et al. 1986] is briefly:

$$\text{Let } E_s = -\frac{1}{2} \sum_{\alpha} [\, (d_\alpha)_s - (o_\alpha)_s \,]^2 \qquad (2.1.a)$$

which is the more specific expression of equation 2.1, where α stands for all output nodes of the neural network. Then

$E = \sum_{s \in S} E_s$ is an overall measure of the error. In order to adjust the weight values so that the mean square error is minimized, we have to determine what effect such changes will cause to E. Now,

$$\frac{\partial E_s}{\partial w_{\alpha\beta}} = \frac{\partial E_s}{\partial (net_\alpha)_s} \frac{\partial (net_\alpha)_s}{\partial w_{\alpha\beta}} \qquad (2.4)$$

where $$(net_\alpha)_s = \sum_{\beta} w_{\alpha\beta} (o_\beta)_s \qquad (2.5)$$

Thus $$\frac{\partial (net_\alpha)_s}{\partial w_{\alpha\beta}} = \frac{\partial}{\partial w_{\alpha\beta}} \sum_{\kappa} w_{\alpha\kappa} (o_\kappa)_s = (o_\beta)_s \qquad (2.6)$$

where κ refers to a node in the layer preceding α.

Let $$(\delta_\alpha)_s = -\frac{\partial E_s}{\partial (net_\alpha)_s} .$$

This suggests that to decrease E we should make weight changes according to

$$(\Delta w_{\alpha\beta})_s = \eta \, (\delta_\alpha)_s \, (o_\beta)_s . \qquad (2.7)$$

For output nodes in the neural network, $(\delta_\alpha)_s$ is computed as:

$$(\delta_\alpha)_s = \frac{-\partial E_s}{\partial (net_\alpha)_s} = \frac{-\partial E_s}{\partial (o_\alpha)_s} \frac{\partial (o_\alpha)_s}{\partial (net_\alpha)_s} \qquad (2.8)$$

but $(o_\alpha)_s = F_\alpha ((net_\alpha)_s)$ (see 2.2)

Thus $$\frac{\partial (o_\alpha)_s}{\partial (net_\alpha)_s} = F'_\alpha ((net_\alpha)_s) \qquad (2.9)$$

where F'(z) is the derivative of F(z) with respect to z (therefore F must be differentiable).

From (2.1.a)

$$\frac{\partial E_s}{\partial (o_\alpha)_s} = - [(d_\alpha)_s - (o_\alpha)_s]$$

thus, $$(\delta_\alpha)_s = [(d_\alpha)_s - (o_\alpha)_s] F'_\alpha ((net_\alpha)_s) \qquad (2.10)$$

when α is an output node.

If the node α is not an output node, we use the chain rule:

$$\sum_\kappa \frac{\partial E_s}{\partial (net_\kappa)_s} \frac{\partial (net_\kappa)_s}{\partial (o_\alpha)_s} = \sum_\kappa \frac{\partial E_s}{\partial (net_\kappa)_s} \frac{\partial}{\partial (o_\alpha)_s} \sum_\rho w_{\kappa\rho} (o_\rho)_s =$$

$$= \sum_\kappa \frac{\partial E_s}{\partial (net_\kappa)_s} w_{\kappa\alpha} = - \sum_\kappa (\delta_\kappa)_s w_{\kappa\alpha} \qquad (2.11)$$

where κ refers to a node in the layer succeeding α, and ρ refers to a node in the same layer of α (Figure 2.1).

Substituting (2.9) and (2.11) for the two factors in (2.8) gives

$$(\delta_\alpha)_s = F'_\alpha ((net_\alpha)_s) \sum_\kappa (\delta_\kappa)_s w_{\kappa\alpha} \qquad (2.12)$$

whenever α is not an output node.

Equations (2.10) and (2.12) can be used as follows in a recursive procedure for computing δ 's for all nodes in the layered

neural network. Briefly, first compute the δ's of the output nodes according to (2.10), and modify the weight matrix between the hidden layer and the output layer. Then compute the δ's of the hidden nodes according to (2.12), and modify the weight matrix backward. This recursive procedure proceeds back through the layers until the input layer is reached. Because the back-propagation error correction is implemented by computing δ's, the δ's are called error signals.

In summary, a neural network is described by its configuration, including its architecture and associated parameters. Neural network architecture is described by the number of layers and the number of nodes in each layer. Neural network parameters include learning rate, temperature, and error tolerance. The weights of each connection at a particular time specify the state of the neural network at that time. A neural network determines an appropriate set of weights by supervised learning from a training sample. The BPLMS learning algorithm is briefly stated as follows.

Step 1. Initialize weights $w_{\alpha\beta}$ for all α and β to small randomly distributed values.

Step 2. Present the pattern vector $\mathbf{X}$ of $s \epsilon S$. (s may be selected based on a pre-defined sequence, or randomly).

Step 3. Calculate the actual output $\mathbf{Y}_s$.

Step 4. Compare $\mathbf{Y}_s$ with the desired feature vector $\mathbf{T}_s$, and obtain the error term E_s .

If $E_s < e$ then

if $E_s < e$ for all $s \epsilon S$ then STOP;

else goto Step 2;

else Step 5.

Step 5. Modify the weights

$$w_{\alpha\beta} = w_{\alpha\beta} + \eta \, \delta_\alpha \, o_\beta \quad ;$$

and δ_α's are recursively calculated from output layer back to input layer:

$$(\delta_\alpha)_s = [\, (d_\alpha)_s - (o_\alpha)_s \,] \, F'_\alpha(\, (net_\alpha)_s \,)$$

if α is an output node,

$$(\delta_\alpha)_s = F'_\alpha(\, (net_\alpha)_s \,) \sum_\kappa (\delta_\kappa)_s \, w_{\kappa\alpha}$$

if α is a hidden node.

Goto Step 2.

Appendix II is the detailed BPLMS algorithm using the back-propagation LMS error rule for two- or three-layer neural network models, written in pseudo-language. It has been implemented using the Turbo C language for our specific research purpose. The example results in Figure 1.17 were generated with this software.

2.3.3. ISSUES RELEVANT TO LEARNING

2.3.3.1. LEARNING RATE AND MOMENTUM

The change of weights during learning is proportional to $\partial E_s/\partial w_{\alpha\beta}$ as shown in equations (2.4)-(2.10). The proportionality η is specified in advance. Larger η results in larger changes in the weights, but this can lead to oscillation instabilities during the learning process. There are no general rules to select an "optimal" η, which depends upon the characteristics of the training set S. To decrease the likelihood of oscillations as η is increased, a momentum parameter M can be introduced in the learning rule. That is,

$$\Delta w_{\alpha\beta}(t+1) = \eta\ (\delta_\alpha)_s\ (o_\beta)_s + M\ \ \Delta w_{\alpha\beta}(t) \quad .$$

This means that the current change in the weight depends partly on the change during the last iteration, tending to filter out high-frequency variation in weight change. Fast learning without oscillation can be accomplished with M≈0.9 and a large value of η (e.g. 1.0) [Rumelhart et al. 1986].

2.3.3.2. CONVERGENCE

For neural networks using Hebb's learning rule, convergence is defined as [Omidvar 1987]:

$$\lim_{t\to\infty} \Delta\ w_{\alpha\beta}(t) \to 0 \quad \text{for all connected nodes } \alpha \text{ and } \beta \ .$$

Theoretical studies (e.g. [Minsky and Papert 1969] [Hirsch 1987] [Bruck and Goodman 1987]) indicate that convergence in the fully

parallel network, one case of which is the layered neural network with BP learning, is not guaranteed. However, in practical applications, convergent solutions can be found in virtually every case by appropriate settings of the learning rate and the error tolerance (cf. [Rumelhart et al. 1986, p361]).

2.3.3.3. GLOBAL MINIMA VS. LOCAL MINIMA

Given a particular neural network architecture and training data set, a state of the neural network which generates the minimum total error is the optimal solution [Hinton and Sejnowski 1986]. The BP learning algorithm is actually a gradient descent procedure which seeks the minimum error by moving downhill along the error surface (see Section 2.3.2). In a multilayer network the error surface may be very complex, with many minima. Some of the minima correspond to an optimal solution, and are therefore global minima. (Notice that here we are discussing the minimum of the error function. There is at least one global minimum corresponding to the optimal solution. However, there may exist several global minima. In the latter case, more than one set of weight states will correspond to the optimal solution.) There also exist some minima which are not optimal, which are called local minima. These minima may also qualify as practical solutions (Figure 2.6).

The BP learning rule is not guaranteed to converge to a global minimum [Rosenblatt 1962]. A strategy for escaping from local minima is the use of a stochastic learning procedure such as randomly setting the learning rate [Rosenblatt 1962] [Ackley 1987].

However, the local minimum problem is not usually encountered in practical applications. Empirically it has been found that high-dimensional spaces (with many weights) have relatively few local minima, and the learning algorithm seems to find paths out of most local minima [McClelland and Rumelhart 1988, p135], indicating that these are probably saddle-points. Moreover, in most pattern recognition problems, the goal is not necessarily to find a minimum error state, but to find the weight matrices with which the neural network is able to map the sample patterns and the desired classes within a pre-defined error tolerance level (Figure 2.6).

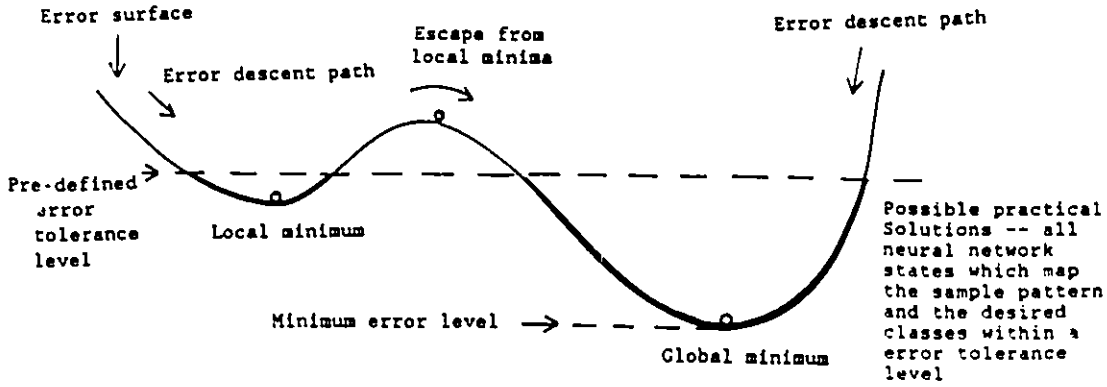


Fig. 2.6. Global Minima, Local Minima, and Possible Solutions

2.3.3.4. SYMMETRY BREAKING

At the initial step of BP learning, if all weights start out with equal values, the system will never escape this state. This is because changes in weights are propagated back through the network by

error signals. Equal weights result in equal error signals, and weights would therefore be adjusted equally. This problem can be overcome readily by "symmetry breaking", or starting the system with random values for the weights.

2.4. CURRENT STATUS OF NEURAL NETWORK APPLICATIONS

2.4.1. ACCEPTANCE CRITERIA FOR A NEURAL NETWORK MODEL

There are two possible basic criteria for acceptance of a neural network model. One is to evaluate the learning result based on the training sample [Widrow et al. 1988]. From this point of view, 100% correctness is the objective. A second approach would be to emphasize the predictive ability of the neural network model. In most practical applications [Lippmann and Gold 1987] [Guyon et al. 1989] the neural network model learns from a training data set, and then is tested with independent test data. If the test results meet a pre-defined standard, then the model is accepted.

From the theoretical point of view, neither of these two approaches is pertinent for evaluation of the performance of neural network models that deal with managerial statistical data. The first approach is overly optimistic in its estimation of misclassification for a population. The second approach is highly influenced by the sample itself. For instance, if the two classes to be classified are far apart from each other, then test results always tend to be good. Moreover, if both training data and test data are not representative samples from the population, the misclassification rate from test

results might be much lower or much higher than the true rate. This research theoretically investigates the behavior of the BPLMS neural network algorithm for classification. The fundamental approach to the evaluation of neural network models employed in this study is to compare the classification boundary generated by a neural network to the true (or optimal) classification boundary based on the simulated data. Since in practice the true boundary is not known, the only means to improve the performance of a neural network is to reduce its misclassification rate. This research will try to show how to make the classification boundary generated by the neural network closer to the true one by reducing the misclassification rate.

2.4.2. NEURAL NETWORK DEVELOPMENT CYCLE

While there have been a number of successful applications of neural networks in pattern recognition [Lippmann 1987] [Lippmann 1989], the development of neural network model theories is still in its infancy. There is still no clear understanding of what network topology is needed for a particular task [Wieland & Leighton 1987]. In particular, the design of neural networks for managerial pattern recognition has had little attention, due partly to a perceived need for business competitive advantage [O'Reilly 1989]. The prototyping approach is considered to be a suitable paradigm for neural network development [Lehar & Weaver 1987]. The prototyping procedure suggested (see Figure 2.7) is similar to that used for decision support system development [Cerveny et al. 1986].

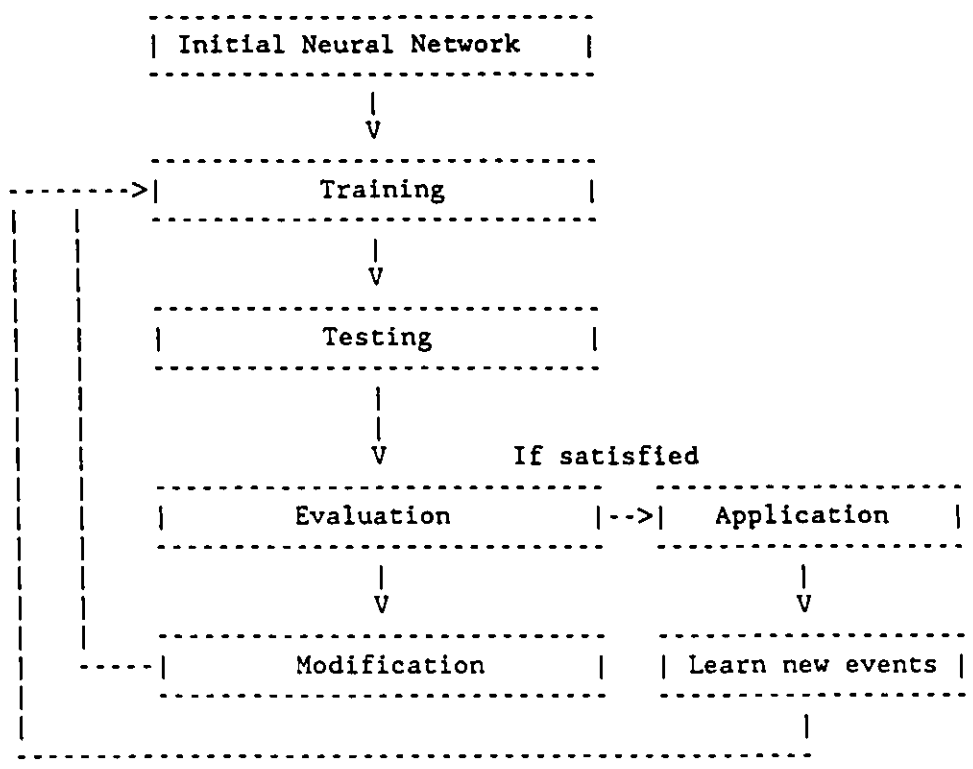


Fig. 2.7. Neural Network Prototype Development

The development cycle consists of the following stages. First, an initial neural network model is structured and trained with the training data set. It is then tested with the test data set, which is usually disjoint with the training set. If a pre-defined classification criterion is satisfied, then the neural network is ready for use. If the neural network is found to have large disparities in performance, changes of the neural network architecture and a repeat of the training process is necessary. As well, a neural network should be improved through re-learning whenever additional training samples become available. The initial neural network configuration is likely to be important to the development procedure. Because neural network configuration is

highly dependent on the problem to be solved, the selection of an initial model is a contingency process. Nevertheless, a heuristic rule may be helpful in creating the initial neural network model. Kolmogorov's mapping neural network existence theorem [Hecht-Nielsen 1987] suggested that, in a two layer neural network, (2m+1) hidden nodes are required for an m dimensional pattern vector (see Section 1.3.1.6.) This number has been recommended as a lower bound [Bruha 1989].

It is commonly accepted that the behavior of the neural network training process depends heavily upon the training set itself [Soulie et al. 1987]. Hence, a full investigation of the BPLMS neural network algorithm's behavior during training and in solving managerial pattern recognition problems is needed. The remainder of this thesis will describe such an investigation.

CHAPTER THREE

CLASSIFICATION BOUNDARIES IN NEURAL NETWORKS

Chapter 2 has indicated that, through supervised training, a neural network is able to eliminate classification error by perfectly classifying the training samples. However, it has also been found that, in the case of statistical pattern data, a complete separation of the training sample groups by a neural network often has poor classification predictability. Hence, it is very important to understand the influence of parameter values, training data sets, network configurations and network states on the training process. A good way to do this is to investigate the behavior of the classification boundary generated by the neural network during the learning process. The ultimate purpose of this investigation is to improve the predictive performance of neural networks in managerial data classification. In this work we investigate the two layer neural network in two class classification, since a two layer neural network can handle any level of difficulty in pattern recognition (see Section 1.3.1.6.) in managerial classification problems, which usually involve two classes (see Section 1.2.1). It is also assumed that Y is one-dimensional with value y and, given a pattern vector, the output value y of the neural network will be called the

classification score.

3.1. BOUNDARY GENERATION

Usually, the values of the x_i's are normalized to [0, 1] for computational convenience. The output value y of every sample point is also in [0, 1]. Hence, the X-y space is a hypercube. As an example, if the X dimension is two, the hypercube can be depicted in three dimensions (see Figure 3.1). At step t the neural network corresponds to a function surface $y=\phi(X)$, called the y-surface. Suppose that the cut-off y score for separating the two classes is 0.5. The boundary at step t then is the intersection line of the y-surface and the plane y=0.5.

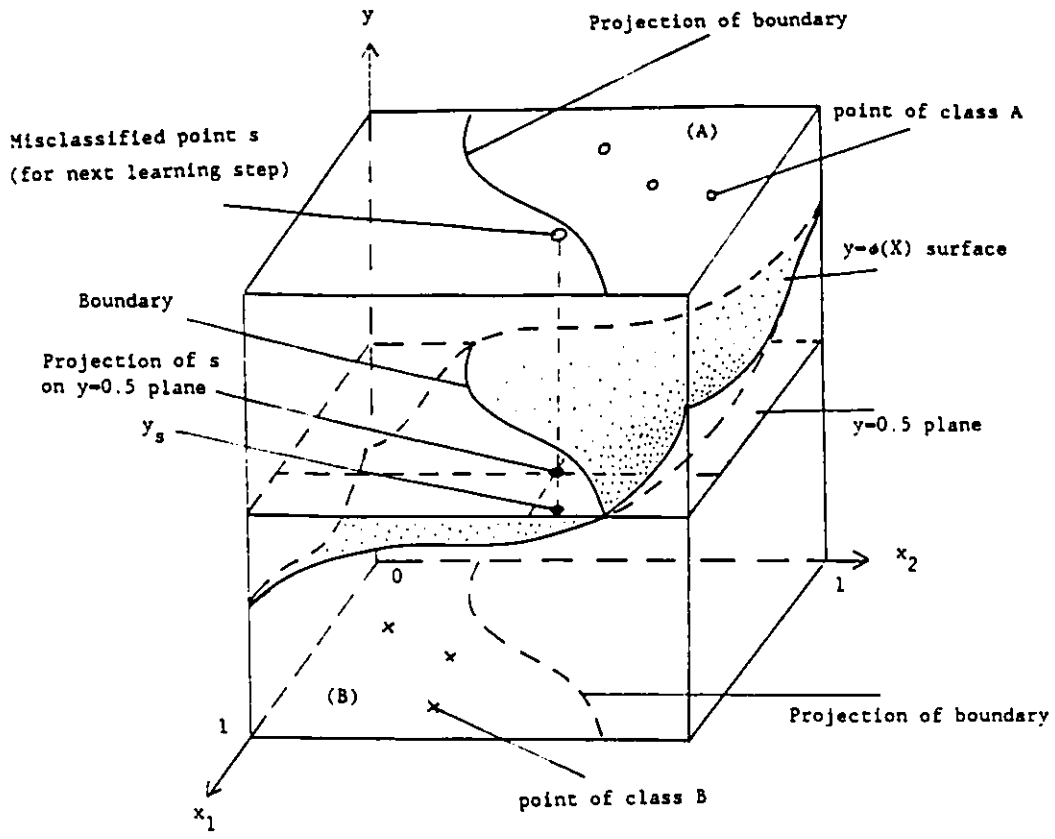
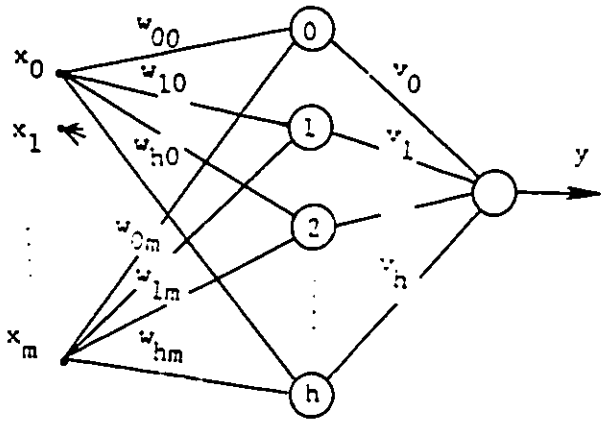


Fig. 3.1. Neural Network Hypercube

To simplify this discussion, Temperature T (i.e. the slope parameter in the sigmoid function) will be set to 1. For convenience, a number of new notations are introduced for the neural network with two layers and single output as shown in Figure 3.2.



$$U_i = (w_{i0}, w_{i1} \cdots w_{im}) \quad i=0,1 \ldots h$$

Fig. 3.2. Two Layer, Single Output Neural Network

h : the number of hidden nodes;

U_i : weight vector of the inputs into the ith hidden node,

$U_i=(w_{i0}, w_{i1}, \ldots w_{im})$, (i=1...h)

where w_{ir} (r=0...m) is the weight at the connection from the rth input node to the ith hidden node;

v_i : weight at the connection from the ith hidden node to the (single) output node;

With this notation, the neural network function $y=\phi(X)$ is expressed as:

$$y = \{ 1 + \exp [- (v_0 + \sum_{i=1}^{h} v_i o_i)] \}^{-1}$$

$$= \{ 1 + \exp [-(v_0 + \sum_{i=1}^{h} v_i (1 + \exp (-U_i X'))^{-1})]\}^{-1} \qquad (3.1)$$

At the boundary, the classification score $y = 0.5$, and the boundary function representing the relationship among the x_i's at the boundary is then

$$\{ 1 + \exp [-(v_0 + \sum_{i=1}^{h} v_i (1 + \exp (-U_i X'))^{-1})] \}^{-1} = 0.5$$

That is,

$$v_0 + \sum_{i=1}^{h} v_i (1 + \exp (-U_i X'))^{-1} = 0 \qquad (3.2)$$

In the remainder of this section we will investigate the changes to the y surface, and classification boundary movement resulting from one learning cycle; that is, from the moment just before learning a misclassified point s (step t) to the moment just after learning s (step t+1). For convenience, the variables without super-script represent the corresponding values at step t, and those with super-script 1 represent the values at step t+1. For instance, y, v_i, and U_i are actually $y(t)$, $v_i(t)$, and $U_i(t)$, respectively; and y^1, v_i^1, and U_i^1 are $y(t+1)$, $v_i(t+1)$, and $U_i(t+1)$, respectively.

Suppose that there is a misclassified point s with desired classification score ι_s=1, where ι is a one-dimensional feature vector value from T_s (the desired feature vector), and y_s<0.5 at step t. After learning point s, the function $y=\phi(X)$ is changed by

modifying the weights v_i and U_i for all i. At step t+1, the classification score y is :

$$y^1 = \{ 1 + \exp [-(v_0 + \sum_{i=1}^{h} v_i^1 (1 + \exp (-U_i^1 X'))^{-1})]\}^{-1} \quad . \quad (3.3)$$

In order to find v_i , the error signal of the output node δ_1 should be calculated. In the sigmoid function $F(u)=[1+\exp(-u)]^{-1}$, where u is the net input of the output node, $F'(u)=[1+\exp(-u)]^{-2}=F(u)[1-F(u)]$. In the present case $F'(u)=y_s(1-y_s)$. From formula (2.10), Chapter 2, we have

$$\begin{aligned} \delta_1 &= (t_s - y_s)\, y_s(1-y_s) \\ &= (1 - y_s)\, y_s(1 - y_s) = y_s(1-y_s)^2 \quad . \end{aligned} \quad (3.4)$$

According to (3.4) and formula (2.7), v_i^1 will be

$$\begin{aligned} v_i^1 &= v_i + \Delta v_i \\ &= v_i + \eta\, y_s\, (1 - y_s)^2\, [1 + \exp(-U_i X_s')]^{-1} \end{aligned} \quad (3.5)$$

Using the back propagation algorithm, the U_i^1 can be found as follows: Let δ_{2_i} be the error signal from the ith hidden node. Apply $F'(u)=F(u)[1-F(u)]$ again, but $F(u)=[1 + \exp(-U_i X_s')]^{-1}$ at this time. From formula (2.12), we have

$$\delta_{2_i} = [1 + \exp(-U_i X_s')]^{-1}(1 - [1 + \exp(-U_i X_s')]^{-1})\,\delta_1 v_i$$

$$= [1 + \exp(-U_i X_s')]^{-1}(1 - [1 + \exp(-U_i X_s')]^{-1})\, y_s(1-y_s)^2 v_i \,. \tag{3.6}$$

From formula (2.7), at step (t+1) U_i is modified to become U_i^1 ,

$$U_i^1 = U_i + \Delta U_i = U_i + \eta\,\delta_{2_i} X_s$$

$$= U_i + \eta\, v_i[1+\exp(-U_i X_s')]^{-1}(1-[1+\exp(-U_i X_s')]^{-1})\, y_s\,(1-y_s)^2\, X_s \,. \tag{3.7}$$

Hence, the change of the y surface from step t to (t+1) is

$$\Delta y = y^1 - y =$$

$$= (1 + \exp[-(v_0 + q^1)])^{-1} - (1 + \exp[-(v_0 + q)])^{-1} \tag{3.8}$$

where $q^1 = \sum_{i=1}^{h} ([v_i + \eta\, y_s\,(1-y_s)^2\,(1 + \exp(-U_i X_s'))^{-1}]$

$$*[1 + \exp(-(U_i + \eta\, v_i\,(1 + \exp(-U_i X_s'))^{-1}$$

$$*(1 - (1 + \exp(-U_i X_s'))^{-1})\, y_s(1-y_s)^2\, X_s\,)\, X')]^{-1}) \tag{3.9}$$

and $q = \sum_{i=1}^{h} v_i\,[1 + \exp(-U_i X')]^{-1}$. (3.10)

Because q^1 may or may not be greater than q, Δy may or may not be greater than zero, depending on v_i , U_i and $\ddot{X}$. This means that after learning the misclassified point s, the function surface is shifted.

The shifted amount Δy is highly dependent upon the position in the pattern space and the current state of the neural network at step t, given a certain neural network structure and a particular learning algorithm.

The function

$$y = \{ 1 + \exp [-(v_0 + \sum_{i=1}^{h} v_i (1 + \exp (-U_i X'))^{-1})] \}^{-1}$$

is a one-to-one function of **X**. When the y surface is shifted up, the boundary moves toward point s, and when the y surface is shifted down, the boundary moves away from point s (see Figure 3.3).

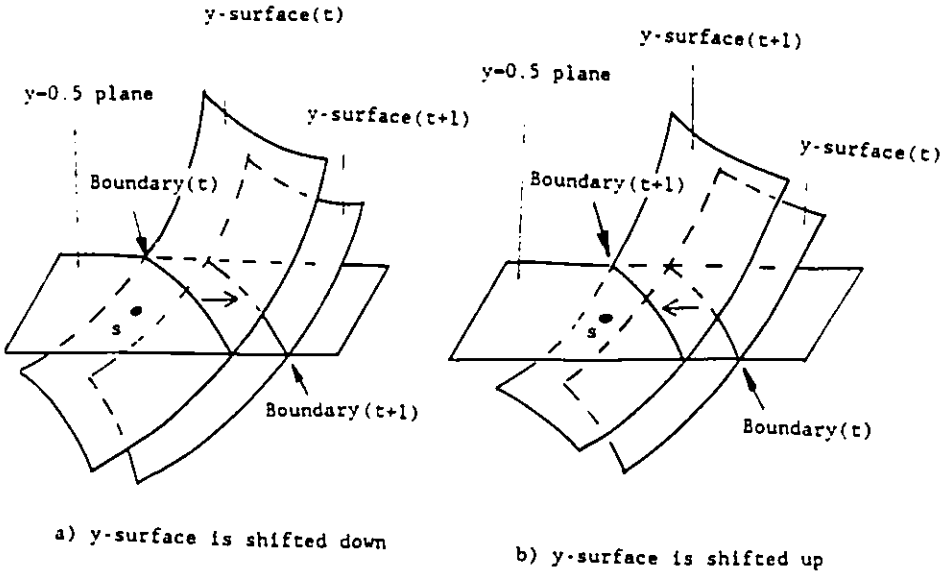


Fig. 3.3. Boundary Movement

3.2. DISCUSSION OF BOUNDARY BEHAVIOR

The foregoing analysis of boundary generation brings the following general conclusions. In terms of classification boundary generation, the behavior of the neural network is highly dynamic. All of the following factors influence boundary generation:

(1) The training sample set has a major influence on the boundary because the neural network algorithm is always attempting to modify the boundary in order to correctly separate the classes,

(2) Neural network architecture,

(3) Function parameters,

(4) Initial state (initial U_i's and v_i's),

(5) The current state, which may also be influenced by the sequence of training sample points.

A detailed prediction of boundary behavior is difficult due to the complex interrelated functions involved, and the highly dynamic nature of neural networks. However, we may look at several special cases which are helpful in understanding the boundary behavior.

3.2.1. SPECIAL CASES

3.2.1.1. POSITIVE v_i's

We have mentioned earlier that the weight matrices initially were assigned real values selected at random from a uniform distribution on (0, 1). In the special case of all positive v_i, each term in expression (3.9) is always greater than the corresponding term in expression (3.10). Hence, $\Delta y>0$. This means that, after a learning iteration using the misclassified point s, the boundary always moves toward s when all v_i are positive.

3.2.1.2. $U_i X_s'$ TENDS TO $-\infty$

When a $U_i X_s'$ tends to be negative extreme, the neural network tends not to adjust the boundary adequately in order to move towards a proper classification for s. This case often happens when some weight value is large negative (e.g. $w_{ij}<-20$), and other weights are moderate. Let $z=[1 + \exp(-U_i X_s')]^{-1}$. When $U_i X_s'$ tends to $-\infty$, z tends to zero. The term in (3.9) then becomes

$$\lim_{z \to 0} \sum_{i=1}^{h} [v_i + \eta\, y_s\, (1 - y_s)^2\, z]$$

$$* \{1 + \exp[-(U_i + \eta\, v_i\, z\, (1 - z)\, y_s\, (1-y_s)^2\, X_s)]\, X'\}^{-1}$$

$$= \sum_{i=1}^{h} v_i\, [1 + \exp(-U_i X')]^{-1}$$

yielding $\Delta y=0$ (compared with (3.8)). This means that there is little effect on learning when $U_i X'_s$ tends to $-\infty$. That is, a badly misclassified point s has little influence on boundary position and is therefore unlikely ever to be classified correctly. This case is very important, especially during real computation, because in a digital computer the sigmoid logic is never complete. That is, for example, when $U_i X'_s < -15$, the internal value of $[1 + \exp(-U_i X_s')]^{-1}$ in the computer might actually be zero.

3.2.1.3. $U_i X'_s$ TENDS TO $+\infty$

When $U_i X'_s$ tends to be positive extreme, boundary adjustment tends to be very sensitive to the location of point s (Figure 3.3(b)). This case often happens when some weight value is positively large (e.g. $w_{ij} > 20$), and other weights are moderate. In this case, $z=[1 + \exp(-U_i X_s')]^{-1}$ tends to 1. Terms in (3.9) then become

$$\lim_{z \to 1} \sum_{i=1}^{h} [v_i + \eta\, y_s\, (1 - y_s)^2\, z]$$

$$* \{1 + \exp[-(U_i + \eta\, v_i\, z\, (1 - z)\, y_s\, (1-y_s)^2\, X_s)]\, X'\}^{-1}$$

$$- \sum_{i=1}^{h} [v_i + \eta\, y_s\, (1 - y_s)^2]\, [1 + \exp(-U_i X')]^{-1}$$

$$> \sum_{i=1}^{h} v_i\, [1 + \exp(-U_i X')]^{-1}$$

yielding $\Delta y > 0$ in equation (3.8).

3.2.1.4. SMALL η, SMALL $U_i X'$, AND MODERATE v_i's

The condition of small $U_i X'$ does not usually occur. However, if we introduce temperature T and let the value of T become large, the function $[1+\exp(-U_i X'/T)]^{-1}$ could be considered as that of $[1+\exp(-U_i X')]^{-1}$ where $U_i X'$ is small. It will be seen that in the condition of small $U_i X'$ and moderate v_i, a small value of η always makes the boundary move very slowly towards the current training sample point.

If $U_i X'$ is small, $[1+\exp(-U_i X')]^{-1} \approx 0.5$ which can be derived from the approximation of $e^{-\epsilon} \approx 1-\epsilon$ for small ϵ, or directly from the sigmoid logic function (see Chapter 2, Fig. 2.4).

The terms in (3.9) become

$$q^1 \approx \sum_{i=1}^{h} [v_i+0.5\eta y_s(1-y_s)^2]\{1+\exp[-(U_i+0.25\eta v_i y_s(1-y_s)^2 X_s)X']\}^{-1} .$$

(3.11)

Further assume that η is small (e.g. $=0.01$) and the v_i's are moderate (e.g. $|v_i|<20$). Because $0.25\ \eta\ v_i\ y_s\ (1-y_s)^2\ X_s$ is relatively small, we can use the approximation $e^{-\epsilon} \approx 1-\epsilon$ again in expression (3.11). The terms in (3.9) then become

$$q^1 \approx \sum_{i=1}^{h} [v_i+0.5\eta y_s(1-y_s)^2][(2-U_iX')-0.25\eta y_s(1-y_s)^2\ v_i X_s X']^{-1} .$$

Letting $A_1=0.5\ \eta\ y_s(1-y_s)^2$, and $A_2=2-U_iX'$ for convenience, gives

$$q^1 \approx \sum_{i=1}^{h} [(v_i + A_1)\ (A_2 - 0.5\ A_1\ v_i\ X_s\ X')^{-1}]. \qquad (3.12)$$

In the same way, the terms in (3.10) are found to be

$$q \approx \sum_{i=1}^{h} v_i\ A_2^{-1} . \qquad (3.13)$$

Using these approximations, the difference between the corresponding terms for index i in (3.9) and (3.10) is

$$\begin{aligned}\Delta_i \approx \{&[\ v_i + \eta\ y_s\ (1-y_s)^2\ (1 + \exp\ (-U_iX_s'))^{-1}] \\ &* [1 + \exp(-(U_i + \eta\ v_i\ (1 + \exp(-U_iX_s'))^{-1} \\ &* (1 - (1 + \exp(-U_iX_s'))^{-1})\ y_s(1-y_s)^2\ X_s\ )\ X')]^{-1}\ \} \\ &- \{v_i\ [1 + \exp\ (-U_iX')]^{-1}\}\end{aligned}$$

giving

$$\Delta_i \approx \frac{v_i + A_1}{A_2 - 0.5 A_1 v_i X_s X'} - \frac{v_i}{A_2}$$

$$= \frac{A_2 v_i + A_1 A_2 - A_2 v_i + 0.5 A_1 v_i^2 X_s X'}{A_2 (A_2 - 0.5 A_1 v_i X_s X')}$$

$$= \frac{A_1 (A_2 + 0.5 v_i^2 X_s X')}{A_2 (A_2 - 0.5 A_1 v_i X_s X')} \qquad (3.14)$$

Note that, in our special case, $\eta \approx 0(0.01)$, $A_1 \approx 0(0.001)$, $A_2 \approx 2$, $|v_i| < 20$, if we consider that the X dimension is usually smaller than 100, giving $X_s X' < 100$. This yields

$$A_2 - 0.5 A_1 v_i X_s X' > \approx 1$$

The numerator of (3.14) has a small positive value, and the denominator is about 2. Hence, $\Delta_i > 0$, which would result in a small positive increment of Δy (see equation 3.8).

This case makes sense especially when the neural network is near the final solution. It is desired that, when learning the point s, the y surface is shifted only slightly to make the boundary include point s in the proper class without significantly changing the boundary for other points.

3.2.2. ERROR FEEDBACK

The neural network obtains feedback information from the training sample through $\iota_s - y_s$. In our case, if $y_s > 0.5$ then point s

lies on the right side of the boundary, and no error feedback is generated. However, if $y_s<0.5$, an error signal will be generated.

Intuitively, the smaller y_s is, the larger t_s-y_s becomes, and a stronger error signal should be generated. However, this is not true in this model.

Recall that the error signal from the output node

$$\delta_1 = y_s (1 - y_s)^2 \qquad \text{(see equation (3.4))}$$

From $\frac{d\delta_1}{dy_s} = 0$, we have $y_s=\frac{1}{3}$, and δ_1 reaches a maximum of $\frac{4}{27}$ (= 0.148). It is interesting to see that when y_s is close to 0.5- , $\delta_1 = 0.125$, and is close to the maximum. This means that the neural network is relatively sensitive to errors at this point. However, when y_s is very small, (i.e., the discrepancy is very large), the error signal becomes weaker instead of stronger (see Figure 3.4). This implies that a misclassified point which is away from the boundary has little influence on boundary adjustment during training, but misclassified points close to the boundary have much more influence on boundary adjustment.

The error signal from the ith hidden node back to the input layer δ_{2_i} is more complicated to calculate, and depends upon y_s, v_i, U_i, as well as X_s (see equation (3.6)). Maximization of δ_{2_i} is highly dependent upon the portion of the neural network's current

state represented by U_i. From $\frac{\partial \delta_{2_i}}{\partial x_j} = 0$, we have the maximum δ_{2_i} condition

$$U_i \; X_s' = 0 \quad . \tag{3.15}$$

This means that there is no explicit X_s that can cause a maximum error signal for the neural network. This is because the error signal at the middle layer is always affected by current weight values.

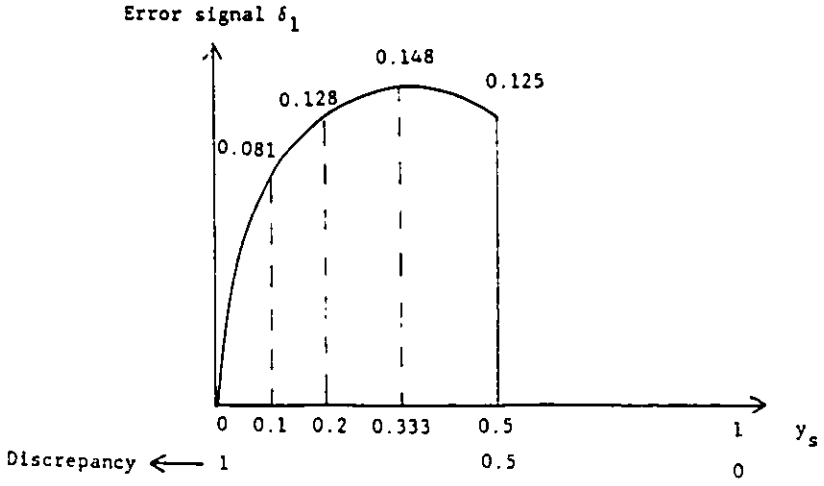


Fig. 3.4. Error Feedback Signal

In summary, the neural network shifts the function surface during the training process. However, the behavior of surface shifting and boundary movement is unpredictable. It is highly dependent on the state of the neural network at step t. Learning is an adaptive hill climbing process. This model only looks for a possible function surface, and generates a boundary which separates the teaching sample points into the desired classes.

3.3. NEURAL NETWORK WEIGHT MATRICES ARE NOT STATISTICS OF THE SAMPLE RANDOM VARIABLE X

Despite the similarity between neural networks and discriminant analysis, in that both techniques require little prior knowledge except the sample set itself, the underlying difference between them is significant. In discriminant analysis the discriminant function is a statistic of the random variable X. However, in a neural network the determination of weight matrices are not totally statistically dependent upon the sample data themselves. According to the definition of a statistic: "A function of one or more random variables that does not depend upon any unknown parameter is called a statistic" [Hogg and Craig 1978, p122], it can be concluded that the determination of neural network weights is not a statistical problem because it does depend on the architecture and initial state as well as the iteration t. From this standpoint, the statistical techniques including estimation of misclassification (e.g. leave-one-out [Toussaint 1974]) and parameter estimation (e.g. jackknife [Mosteller 1971, Miller 1974, Efron 1982]) are not directly applicable to neural networks.

3.4. UNPREDICTABILITY OF NEURAL NETWORK CLASSIFICATION

Most researchers on neural network classification are painfully aware that the classification results of neural networks are unpredictable (cf. [O'Reilly 1989]). In other words, a neural network may classify the training sample data set correctly 100

percent of the time, but still perform poorly on the test data set. This problem is caused by the unpredictability of neural networks in classification, arising from at least two reasons described as follows.

3.4.1. UNPREDICTABLE BOUNDARY GENERATION BEHAVIOR

According to the BPLMS learning algorithm, the neural network learning process proceeds until the training data set is classified 100 percent correctly. However, as discussed in Section 3.2, classification boundary generation of neural networks is not predictable. Given a certain training data set, the classification results generated by individual neural networks may differ in thousands of ways without any explicit reason. For example, given the data in Figure 3.5(a), there are an infinity of possible classification results. Figures 3.5(b)-(e) show several possible outcomes.

This indeterminacy of boundary results is one of the major reasons resulting in the unpredictability of neural networks in managerial classification problems, especially when the training sample size is not very large.

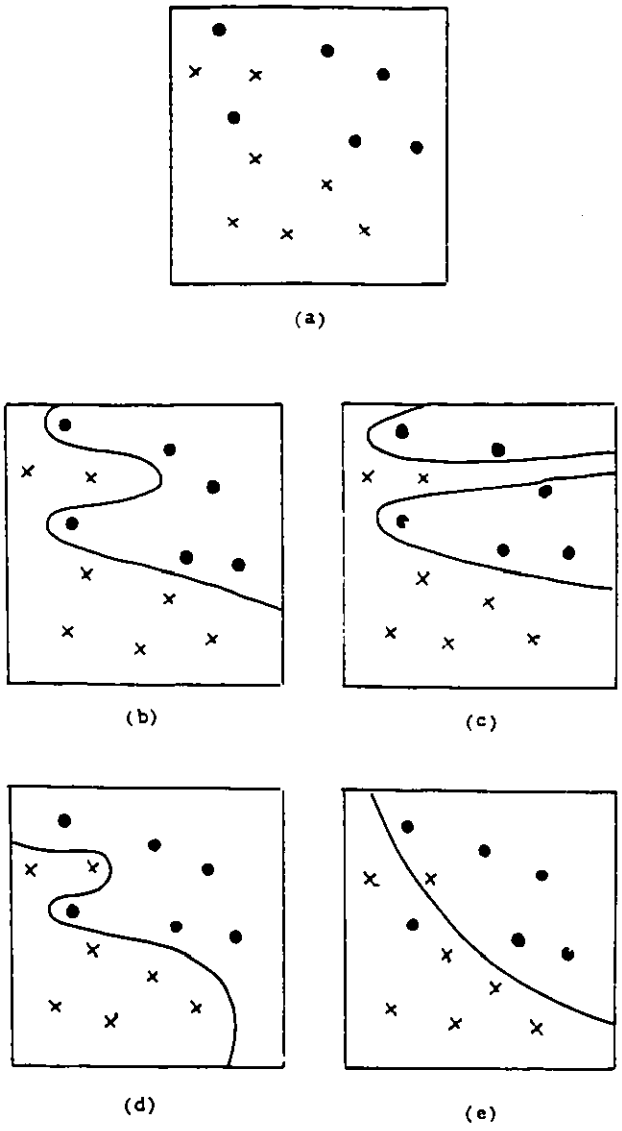


Fig. 3.5. Examples of Boundary Classification Results

3.4.2. DOGMATIC LEARNING MECHANISM

Classification using the standard neural network with the BPLMS learning algorithm works under the assumption that each

input/output pair of the training sample data truly and accurately reflects the relationship between the pattern vector and the corresponding class. This condition is questionable in practical applications. First, most sample data come directly from the real world. Usually, there are errors (or measurement noise) in measuring the pattern vector values; namely, what we have observed from a sample point is $Y=\phi(\hat{X})$ rather than $Y=\phi(X)$, where $\hat{X}$ is the measured X, but the outcome Y of the observation is still assumed to be correct. Secondly, according to artificial intelligence theory, the dimensionality of the pattern vector is never complete for an object in the real world (cf. [Rich 1983]). This argument is certainly true for managerial problems since a formal description of the pattern vector basically provides only a part of the complete information about a person, a company, or a business event. From this standpoint, what we have observed from a sample point is $\hat{Y}=\phi(X)$ rather than $Y=\phi(X)$, where $\hat{Y}$ is the real outcome of an event, caused by both observed pattern variables X and unobserved pattern variables. It is not surprising that, in some cases, two training sample points with the same (accurately measured) X have totally opposite outcomes, but both outcomes are right. This type of information, however, is difficult for neural networks to learn. Therefore, the practical application of neural networks in managerial pattern recognition should not simply pursue 100 percent correct classification of the training sample set.

Before leaving this discussion, let us return to the example in Figure 3.5. The boundary in Figure 3.5(e) seems to be better than

the others even if the samples are not 100% correctly classified, because it is a smooth curve which does not appear to have been affected by random fluctuations in the sample data, and the shape of the boundary seems to be more meaningful in the managerial sense. It is easy to interpret this phenomenon from the statistical viewpoint [Young and Calvert 1974, p206], but difficult to solve this problem with current BP learning algorithms [Kohonen et al. 1988].

3.4.3. PROSPECTIVE MEANS FOR CONTROLLING NEURAL NETWORK UNPREDICTABILITY

There are now two main issues we have examined. On the one hand, neural network techniques appear to have promise in performing managerial pattern recognition, due to their adaptive properties. However, their boundary behavior is relatively unpredictable. Although there is no doubt about classification performance on training samples, the boundary may or may not be close to an "optimal" one, since the neural network is too arbitrary in generating boundaries to be able to approach optimality, however that may be defined. We will therefore have to impose some constraints on the algorithm so that boundary generation will be more consistent with the sense of the type of problems we are attempting to solve. These constraints may come from task domain knowledge, which in this context refers to generic knowledge about managerial pattern recognition. On the other hand, because of the computational properties of neural networks, the neural network weights are not really statistics of the sample random variables. Statistical tools

are therefore difficult, if not impossible, to apply in the determination of these weights. However, a critical aspect of the data is that it is random, and standard learning techniques are highly susceptible to random fluctuations, leading to undesirable classification boundaries. We therefore need to develop a statistical technique to pre-process the training sample data and filter its statistical fluctuations. The remainder of this chapter discusses these two issues in detail, leading to a learning model which can take full advantage of the adaptive nature of the neural network technique.

3.5. PRIOR KNOWLEDGE ABOUT MANAGERIAL PATTERN RECOGNITION PROBLEMS

3.5.1. UTILITY FUNCTIONS AND MONOTONICITY

Recall the example in Section 1.1. The classification of a stock portfolio is a utility function of the stock return means and variances. Actually, a wide range of managerial decision problems involving pattern classification can be considered from the point of view of utility or preference analysis [Keeney and Raiffa 1976] [Nicholson 1978]. For instance, the classification of two classes is basically the problem of "which is preferred". In the study of utility theory, qualitative characteristics of the utility function are often introduced. Each of these characteristics implies a certain attitude of the decision maker with regard to his or her preferences. One may express these attitudes mathematically and impose some restrictions, or assumptions, on the utility function in

order to actually assess the utility. One commonly used characteristic is monotonicity [Keeney and Raiffa 1976], which is expressed as

$$[\ x_1 > x_2\ ] \leftrightarrow [\ \phi(X|x_j{=}x_1) > \phi(X|x_j{=}x_2)\ ] \quad .$$

In terms of a two class classification problem if, other things being equal, the larger (or the smaller) value a variable x_j has, the larger the classification score y becomes, then y is monotonic in x_j. The above expression is for a monotonic increasing function. For our purposes, a monotonic decrease is conceptually the same as a monotonic increase, but with the x_j direction reversed. In the assessment of a utility function, prior knowledge about monotonicity is usually available; hence, an assumption about monotonic relationships between certain independent (pattern) variables and utility is usually valid. For example, in the creditworthiness classification case, income and assets are monotonic in the creditworthiness score; namely, other things being equal, the more the better. Sometimes there are pattern variables which are not strictly monotonic. For example, in a semantic questionnaire there may be a scale such as "Too-small ... Good ... Too-large". However, such variables can be decomposed into monotonic variables.

If a monotonicity constraint is imposed on neural network learning, then the y-surface will monotonically increase with changes in the corresponding pattern variable. This will in turn result in an improvement in the predictability of boundary generation.

In linear discriminant analysis (LDA) the discriminant function $y=\sum_{j=0}^{m} b_j x_j$ (Section 1.3.1.2) is monotonic in x_j for all j, since $\partial y/\partial x_j$=constant for all j. In the case of a pattern recognition problem which is monotonic, LDA is superior (since it has an inherent monotonic property) over other distribution estimation techniques which may violate monotonicity and hence result in conflict in decision making (see Figure 3.6). For instance, given a sample, if the distributions of the pattern data are assumed to be normal with different covariance matrices, then a quadratic discriminant function can be generated which is optimal under these assumptions. However, the analysis result may violate monotonicity due to the quadratic shape of the boundary. Consider Figure 3.6(a) where the two dimensions of the pattern space denote mean and variance, respectively, in financial stock mean-variance analysis. For a fixed variance, the stock can become either good or bad along with an increase of mean value. Obviously, this makes no sense in decision making. The problem is caused by an incorrect assumption about the statistical distributions.

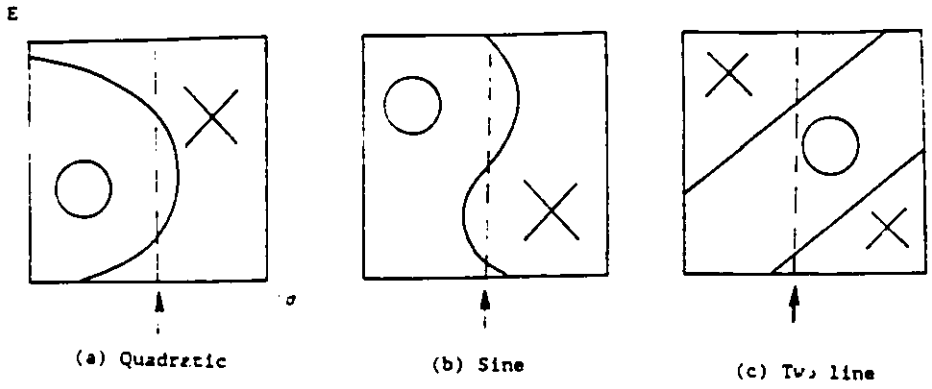


Fig. 3.6. Complex Boundaries Violate Monotonicity

On the other hand, in managerial pattern recognition, knowledge about monotonicity is often not available for some pattern variables. For example, in the classification of personal creditability, there probably is no prior knowledge about what age is a critical level, where the utility function of creditability monotonically increases up to this age level (say, 65) and then decreases. If the critical age is unknown, Age cannot be decomposed into monotonic variables, say Creditability-Increase-Age and Creditability-Decrease-Age. If LDA is employed in this case, the non-monotonicity will be violated since $\partial y/\partial x_{Age}$ can change sign. However, if the neural network technique is used, the monotonic condition constraint can be discharged in the Age variable. The advantage of doing so is more flexibility for classification in dealing with non-monotonic variables. The disadvantage is that there is not enough knowledge to constrain the classification boundary, resulting in unpredictable boundary behavior in these variables.

3.5.2. A PROPOSED HEURISTIC FOR IMPROVING THE NEURAL NETWORK BPLMS TRAINING ALGORITHM

In neural networks the desired classification score t_s is usually defined to be 1 or 0 depending on which class s belongs to, as we have no other prior knowledge. Completely separating the training set into two sets may require a very complex boundary, which would not necessarily meet the objective of having good prediction capabilities. On the other hand, the LDA generates an approximation of a "true" classification. This implies that the classification boundary obtained from LDA could be used as a beginning approximation to the boundary. Assuming that a neural network training algorithm is subjected to a monotonicity constraint, and is trained with a sample carrying the classification scores obtained from the LDA, the y surface generated would have a regular shape, and the classification boundary would be close to the linear function given by LDA, when training is completed (recall Figure 3.1). However, our goal is not to simply map the LDA result to a neural network result, but to improve on it. If we lack knowledge about the underlying distribution functions, the criterion which can be used to evaluate performance is a low misclassification rate for the training sample. To reduce misclassification it is desired to modify the classification scheme generated by LDA. This could be done by learning the misclassified points based on knowledge about the approximate classification scheme generated by LDA. The following example (see Table 3.1.) explains the heuristic approach.

	Sample points	LDA scores	LDA miscls.	Training data	Expected miscls.
Class A	A-1	0.8		A	
	A-2	0.42	x	A	
$i \geq 0.5$	A-3	0.6		A	
	A-4	0.2	x		x
	A-5	0.48	x	A	
	.....			...	
Class B	B-1	0.4		B	
	B-2	0.6	x	B	
$i < 0.5$	B-3	0.8	x		x
	B-4	0.25		B	
	B-5	0.3		B	
	.....			...	

Table 3.1. Example of the Heuristic Approach to Obtain a Proper Training Data Set

Table 3.1. left shows a classification result using LDA. There are 5 misclassified points (scores <0.5 for A and ≥0.5 for B). Some of them are close to the linear LDA boundary, and the others are not, as indicated by the classification scores. Since the "true" boundary may have better measured performance than a linear boundary function, it should have the potential to decrease the misclassification rate. Thus, it is desired to make minor modifications to the current boundary based on the available information, so that misclassification is reduced. In class A, the misclassified points A-2 and A-5 are selected to be included in the new training scheme, because the maximum increase in the scores is about +0.08 which seems to have little risk of causing new misclassifications for class B if the boundary is revised to include them. In class B, the misclassified point B-2 is included because a decrease of 0.1 in the score should result in little risk of introducing new misclassifications for class A. Misclassified points

A-4 and B-3 have large discrepancies, which seem to be caused by overlap of the data sets, and no attempt will be made to bring these into their correct classifications. The new classification result after boundary modification through training is shown in Table 3.1, right. The classification may be improved further, based on the new boundary, if the heuristic procedure can be applied again.

In practice, if the LDA result has a large misclassification rate, and clusters of misclassified points are observed (see Figure 3.7), the heuristic may be used effectively. Identification and verification of clusters is also a pattern recognition problem. In this particular case, vector analysis [Kohonen 1988] is helpful, especially in the case of high dimensionality pattern vectors. An example, with a two dimension pattern vector, of the suggested vector analysis approach of finding the clusters is depicted in Figure 3.8, and an algorithm which utilizes it is described as follows.

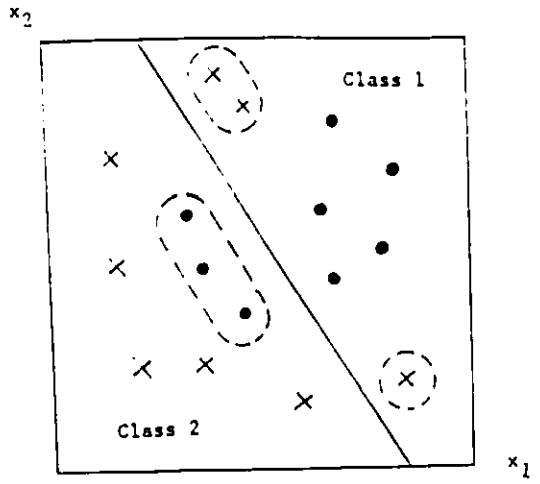


Fig. 3.7. Clusters of Misclassified Points

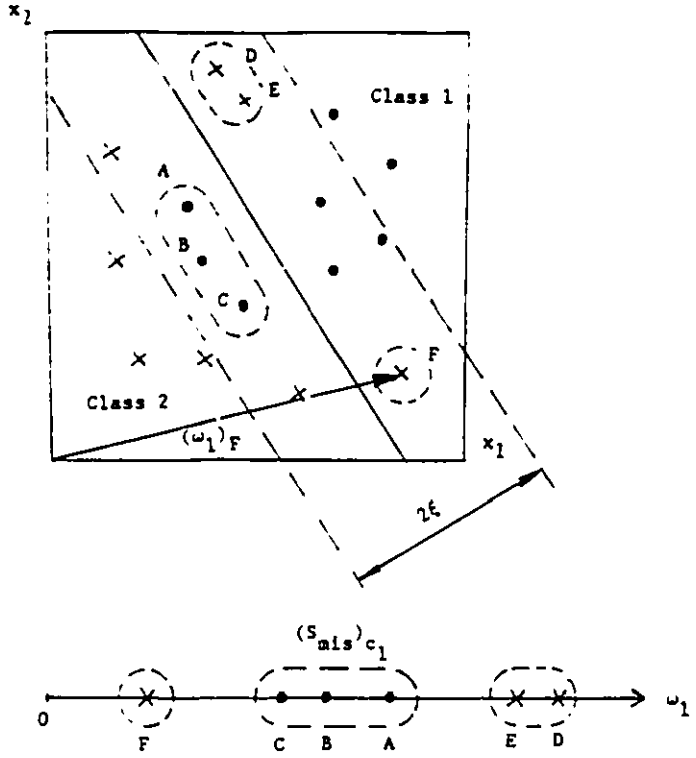


Fig. 3.8. Vector Analysis of Clusters of Misclassified Points

(1) Pre-define a small number ξ (e.g. 0.05). Define set $(S_{mis})_{c_1}$ such that $s \in (S_{mis})_{c_1}$ if s is a misclassified point from class c_1 and $|\zeta_s - 0.5| < \xi$, where ζ_s is the LDA classification score of s. In the example of Figure 3.8 $(S_{mis})_{c_1} = \{A, B, C\}$. Similarly define set $(S_{mis})_{c_2}$.

(2) Let $(\omega_1)_s$ be the angle between the vector direction of s and axis x_1, with $\cos(\omega_1)_s = \frac{(x_1)_s}{||X_s||}$. Plot $(\omega_1)_s$ for all $s \epsilon (S_{mis})_{c_1}$ and $(S_{mis})_{c_2}$ on the ω_1 axis.

(3) Screen the ω_1 axis from 0 to $\pi/2$. Those misclassified points which belong to the same class and are contiguous will comprise clusters. In the example of Figure 3.8 {F}, {A, B, C}, and {D, E} are three clusters. A few points belonging to the other class (e.g. class c_1) might also be allowed to exist within a big cluster (belonging to, for example c_2) on the ω axis.

The trade-off between the potential gain from the corrected misclassifications and the potential risk of introducing new misclassifications can also be made by vector analysis. Figure 3.9 shows an example assuming that $(S_{mis})_{c_1}$ is an observed cluster of misclassified points in class c_1.

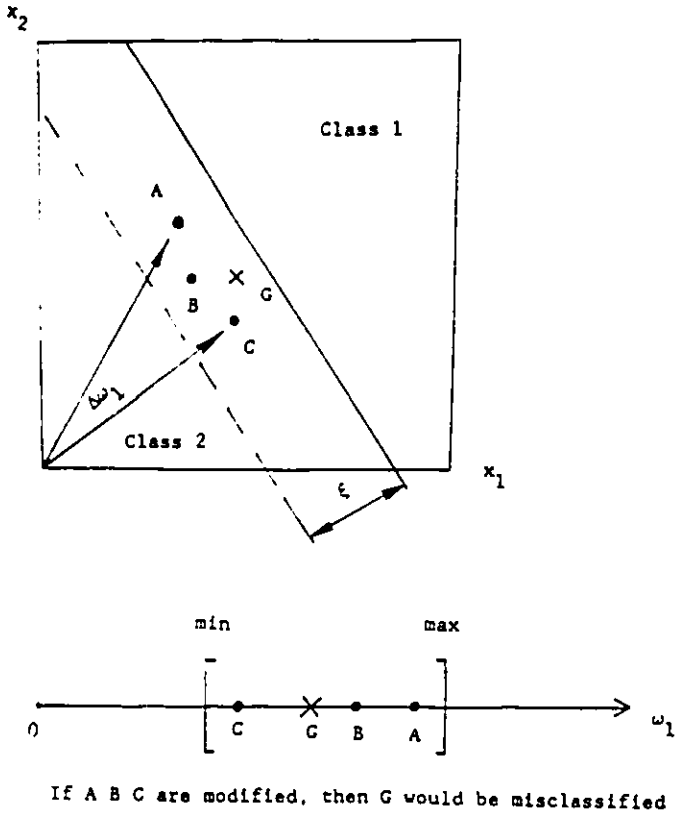


Fig. 3.9. Vector Analysis of the Trade-Off Algorithm

The trade-off algorithm is defined as follows.

(1) Define set $(S_{cor})_{c_2}$ such that $s \epsilon (S_{cor})_{c_2}$ if s is a point in class c_2 which is correctly classified by LDA, and $|\zeta_s - 0.5| < \xi$, and $(\omega_1)_s$ falls into the range $[\ \min_{(S_{mis})_{c_1}} (\omega_1)\ ,\ \max_{(S_{mis})_{c_1}} (\omega_1)\ ]$, where $\min_{(S_{mis})_{c_1}} (\omega_1)$ and $\max_{(S_{mis})_{c_1}} (\omega_1)$ are the extreme values of (ω_1)'s of the

misclassified points in $(S_{mis})_{c_1}$. In the example of Figure 3.9 $(S_{cor})_{c_2}$={G}.

(2) Let the cardinality of the sets $(S_{mis})_{c_1}$ and $(S_{cor})_{c_2}$ be denoted by $|(S_{mis})_{c_1}|$ and $|(S_{cor})_{c_2}|$, respectively. If $|(S_{mis})_{c_1}| > |(S_{cor})_{c_2}|$, boundary modification about the misclassified points in $(S_{mis})_{c_1}$ would introduce fewer new misclassifications if learning is completed using the modified training scheme.

(3) Delete se$(S_{cor})_{c_2}$ ({G} in the example) from the initial training set S. Train the neural network with the revised training data set. When training is complete, the misclassification rate will not be greater than the LDA result; otherwise, reduce ξ and repeat the vector analysis procedure until $\xi \approx 0$.

The heuristic described above requires parameters such as ξ and the desired tolerance to be pre-defined. Note that, after verifying the non-linearity of the desired boundary after LDA pre-processing, the neural network can readily generate a proper boundary based on the monotonicity-constrained training scheme. This kind of adaptive problem is difficult if not impossible to solve by other classification methods.

As in most other artificial intelligence approaches, there is no way to prove that this heuristic approach definitely results in an "optimal" solution. However, the use of prior knowledge about monotonicity and an initial LDA approximation to the boundary can

overcome the unpredictability of a neural network's boundary behavior, and improve its classification performance.

3.6. THE MONOTONIC FUNCTION NEURAL NETWORK MODEL

Two issues related to the problems of standard neural network learning have been discussed thus far. First, the standard BP learning algorithm makes a neural network's boundary behavior unpredictable. In order to ensure a classification result which is more consistent in the managerial context, it has been suggested that monotonicity should be imposed during the learning process. Second, if monotonicity is imposed during learning, the original training scheme may not be suitable, because the training scheme is often in conflict with monotonicity in classification of statistical data. Hence, pre-processing of the training sample based on LDA results is suggested. This section will summarize a learning model for neural networks in classifying statistical data under monotonicity conditions, and we will call it the Monotonic Function Neural Network Model (MF model, for short).

3.6.1. MONOTONIC CONDITION FOR NEURAL NETWORKS

If $y=\phi(X)$ is monotonic in x_j , then $\partial y / \partial x_j \geq 0$. Using the notation in Section 3.1., this condition is generalized as follows.

$$\partial y/\partial x_j = \partial\{\,[1+\exp(-(v_0+\sum_{i=1}^{h} v_i(1+\exp(-U_iX'))^{-1}))]^{-1}\} \,/\, \partial x_j$$

$$= \frac{\partial y}{\partial G}\,\frac{\partial G}{\partial x_j} \qquad \text{where } G=v_0+\sum_{i=1}^{h} v_i[1+\exp(-U_iX')]^{-1}$$

$$= y(1-y)\sum_{i=1}^{h} -\frac{\partial\,[v_i(1+\exp(-U_iX'))^{-1}]}{\partial\, x_j}$$

$$= y(1-y)\sum_{i=1}^{h} w_{ij}\, v_i \exp(-U_iX')[1+\exp(-U_iX')]^{-2} \quad .$$

Thus,

$$\partial y/\partial x_j \geq 0 \iff \sum_{i=1}^{h} w_{ij}\, v_i \exp(-U_iX')[1+\exp(-U_iX')]^{-2} \geq 0$$

for all j, provided y is monotonic in x_j . (3.16)

Condition (3.16) should be satisfied for all monotonic variables x_j . An algorithm can be employed to find this general necessary and sufficient condition. The following algorithm meets these general conditions (see Appendix III for the derivation).

For all j provided y is monotonic in x_j do:

(1) For i=1...h, define set I so that $i \in I$ if

$w_{ij}v_i < 0$;

(2) Compute

$Q_1 = 0.25 \sum_i w_{ij}\, v_i$ for those $i \in I$;

(3) Compute

$Q_2 = \sum_i w_{ij}\, v_i\, A_i$ for those $i \notin I$

where

$$A_i = \min \{[\exp(-\sum_p w_{ip})(1+\exp(-\sum_p w_{ip}))^{-2}],$$

$$[\exp(-\sum_q w_{iq})(1+\exp(-\sum_q w_{iq}))^{-2}] \}$$

where the w_{ip}'s and w_{iq}'s are elements of vector U_i, and all $w_{ip}<0$ and all $w_{iq}\geq 0$;

(4) If $Q_1 + Q_2 \geq 0$, then y is monotonic,

else monotonicity is violated;

End-do. (3.17)

During training for a sample point, if the monotonic condition would be violated due to a large change of weights, η is decreased for that sample point. Because our model does not change the principle of error hill climbing (see Section 2.3), the adaptive nature of the algorithm does not have to be changed. Experiments with this algorithm have shown that, when the proper training data set (see Sections 3.4.2. and 3.5.2.) is used, the monotonic condition is rarely violated.

For practical purposes, we may need to find a sufficient condition for monotonicity. A sufficient condition can be deduced simply from equation (3.16); that is,

$$w_{ij}v_i \geq 0 \quad \text{for all } i=1...h \text{ and corresponding } j. \tag{3.18}$$

Note that the initialized neural network with its weights of random numbers uniform on (0,1) has the monotonicity property.

3.6.2. PROPER TRAINING DATA SET

In Section 3.4.2. a heuristic algorithm to find a proper training data set based on the LDA pre-process result was suggested. Here, it is summarized in a generalized form, using the notation of Section 3.4.2. The main steps of the algorithm are depicted in Figure 3.10.

Step 1. Pre-process the sample set S with LDA, and obtain a linear classification function $\sum_{j=0}^{m} b_j x_j = 0$.

Step 2. Calculate a score $y_s = \sum_{j=0}^{m} b_j x_j$ for each $s \in S$.

Step 3. Normalize the scores from Step 2 on [0,1] such that a point on the linear boundary has score 0.5.

Step 4. Find sets $(S_{mis})_{c_r}$ (r=1,2) such that each member $s \in (S_{mis})_{c_r}$ satisfies $| \zeta_s - 0.5 | < \xi$, where ξ is a pre-decided small number and will be reduced following the iteration procedure. If $\xi \approx 0$, STOP.

Step 5. Calculate $(\omega_j)_s$ (i=1...m-1) for all $s \in (S_{mis})_{c_r}$ (r=1,2), where $(\omega_j)_s$ is the angle between the vector direction of s and axis x_j.

Step 6. Find clusters of misclassitied points $(S^g_{mis})_{c_r}$ (r=1,2; g=1...G; where G is the number of clusters

found) such that

none (or a small number within a pre-defined tolerance) of $(\omega_j)_s$ (j=1...m-1) fall into the range

$$[\ \min_{(S^g_{mis})_{c_r}}(\omega_j)\ ,\ \max_{(S^g_{mis})_{c_r}}(\omega_j)\ ]\ ,\ g=1...G\ ,$$

for all $s\epsilon(S_{mis})_{c_{-r}}$, where c_{-r} is the class other than c_r.

If $(S^g_{mis})_{c_r}$ does not exist (i.e. G=0 for r=1,2),

then STOP;

otherwise, goto Step 7.

Step 7. Find $(S^g_{cor})_{c_{-r}}$ such that $s\epsilon(S^g_{cor})_{c_{-r}}$ if s is a correctly classified point from the class other than c_r by LDA, and $|\zeta_s - 0.5|<\xi$, and $(\omega_j)_s$ falls into

$$[\ \min_{(S^g_{mis})_{c_r}}(\omega_j)\ ,\ \max_{(S^g_{mis})_{c_r}}(\omega_j)\ ]\ \text{ for all } j=1...m-1.$$

Step 8. If $|(S^g_{cor})_{c_{-r}}| << |(S^g_{mis})_{c_r}|$, g=1...G,

then goto Step 9;

otherwise, reduce ξ, and goto Step 6.

Step 9. Delete s from S if s is a misclassified point but

$s\notin(S_{mis})_{c_r}$ (r=1,2), or

$s\epsilon(S^g_{cor})_{c_{-r}}$ (r=1,2; g=1...G).

Step 10. Train the neural network on S from Step 9;

If successful (i.e. after training for a pre-defined number of iterations all $s \epsilon S$ are correctly classified), then a new classification result is obtained, which has a misclassification rate not greater than LDA on the initial sample set;

otherwise reduce ξ , and goto Step 4.

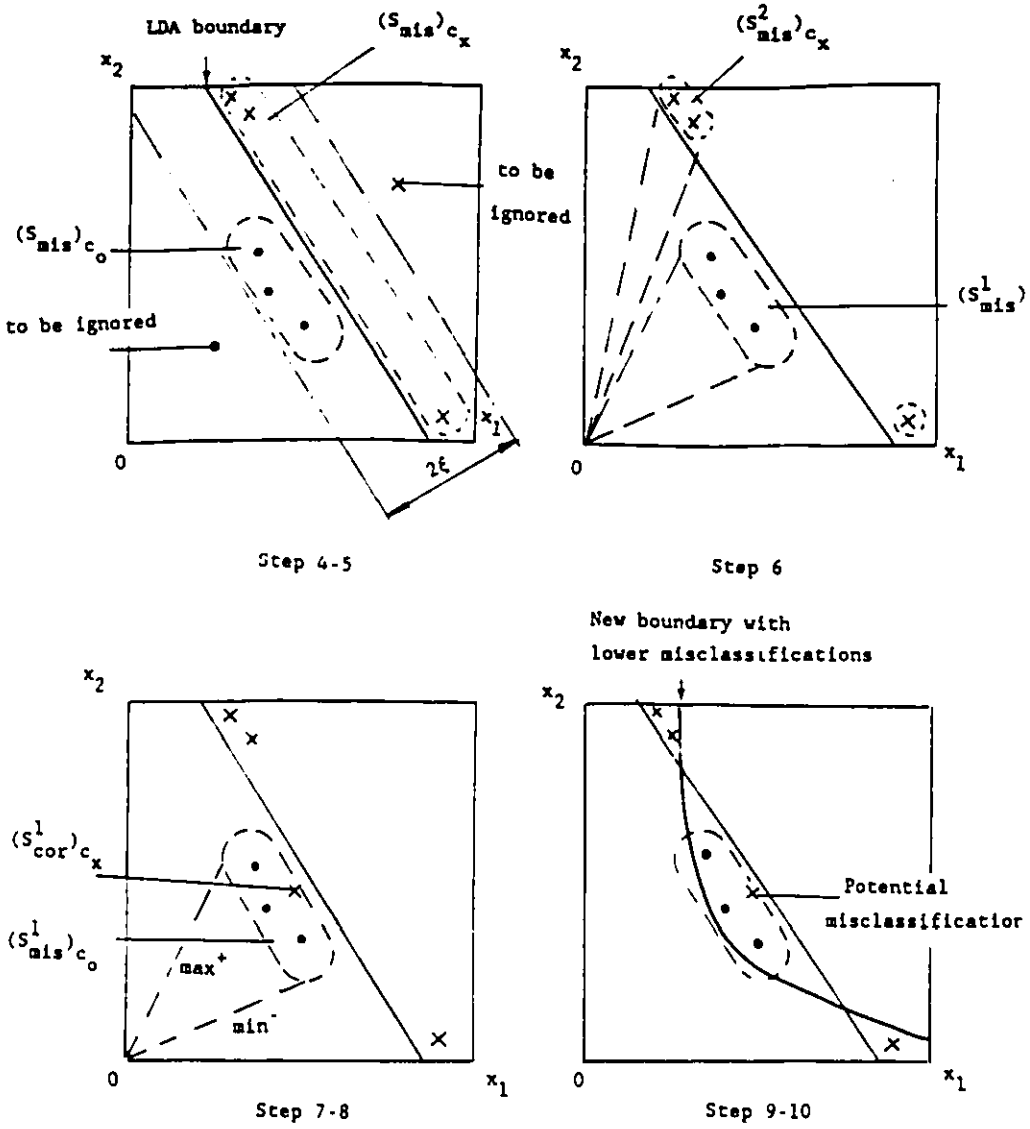


Fig. 3.10. Main Steps of the Algorithm To Find a Proper Training Data Set

As with standard neural networks using the BP learning algorithm, when a new training data point is added into the training set, the MF model would need to totally relearn the updated training set.

3.7. EXAMPLE APPLICATIONS OF THE MONOTONIC FUNCTION NEURAL NETWORK MODEL

3.7.1. AN EXAMPLE OF SIMULATED DATA

In this section an experiment using simulated data is described which applies the Monotonic Function Neural Network Model. For this experiment, the pattern vector dimension was two, so results could be displayed on a plot. The size of the data set was 40 (20 in each class). An initial sample data set was generated by the method described in Section 1.3.3.2. The true classification boundary was cosine shaped (see Figure 3.11). The LDA approach classified 33 of the 40 sample points correctly (82.5%). Using the algorithm described in Section 3.6.2, ξ was determined by the algorithm to find a proper training data set (Section 3.6.2) as 0.04. Four misclassified points were within this tolerance. However, one of the previously correctly classified points was now incorrectly classified, resulting in a net gain of three properly classified points. Thus a new training data set with 36 sample points was used to train the neural network, with 5 hidden nodes, under the monotonic condition. After 6406 learning sweeps, the final classification boundary was obtained, which is closer to the true boundary than LDA in most regions of the pattern space, resulting in an overall accuracy of 90% (Figure 3.11).

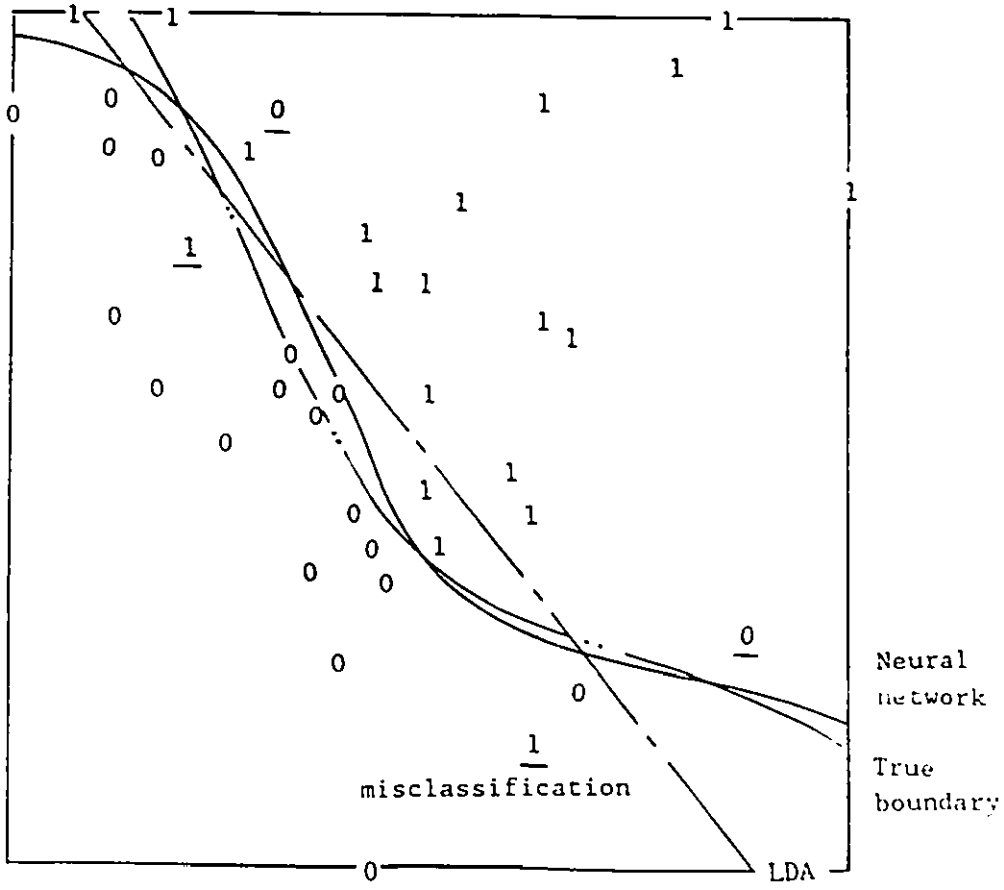


Fig. 3.11. Experimental Result for the Monotonic Function Neural Network Classifier

However, using standard neural network training on the original data set , no convergence was observed after 65454 learning sweeps.

3.7.2. AN EXAMPLE OF REAL DATA

In the last subsection , simulated data were used for testing the MF model. The advantage of simulated data is that the optimal boundary is known and can be used to evaluate the performance of the model, as shown. However, the above method is not applicable for real problems. A widely acknowledged testing method is cross-validation [Lippmann 1989], a form of the jackknife method [Efron 1982] which is commonly used in statistics. In this method, a training data set is randomly selected from the available sample. After training the classifier, the remainder of the observations are used to test the classifier. Usually, the sample is equally divided into two subsets, and each of them serves as training data set and testing data set in turn in two trials.

The data used in this experiment came from the Alpha TV Commercial data bank published in [Green 1978]. This data bank, made up from the responses of 252 male adults, is used as a kind of "running case" throughout the text [Green 1978]. The author used a subset of the data bank to conduct a linear discriminant analysis of two classes described as follows:

Class 1: 78 respondents who selected the Alpha brand of radial tires;

Class 2: 174 respondents who did not select the Alpha brand;

and the pattern variables:

1. whether Alpha was the brand selected in the respondent's last purchase of replacement tires;
2. pre-exposure interest in Alpha radial tires;

3. post-exposure believability of the Alpha commercial;

4. post-exposure interest in Alpha radial tires.

Using the 252 data points, linear discriminant analysis gives the result that (52+131)/252 or 72.6 percent of the sample is correctly classified, although the difference in the locations of the two class centroids was highly significant (F=17.98, with 4 and 247 degrees of freedom [Green 1978, p179]). Hence, as pointed out by the author, the separation effected by the linear discriminant function is not good from a practical point of view. However, the MF model can improve on this result.

The classification case being discussed is a typical managerial pattern recognition problem. The assumption about monotonic relationships between selection of Alpha radial tires and the four pattern variables, including last brand purchased, pre-exposure interest, post-exposure believability, and post-exposure interest is valid. The Monotonic Function model is therefore applicable, and an experiment was designed as follows. Every other point of the class was selected for one subset S_1, and the remainder was selected for S_2, such that each contained 39 points for class 1 and 87 points for class 2. In the first trial S_1 was the training data set and S_2 was the testing data set, and in the second trial the roles of the two data subsets were switched.

Because this was a high dimensionality problem, it is not possible to depict the population in a two dimension graph. Nevertheless, the clustering phenomenon was well observed in the vector analysis (see Appendix IV (1)). In the two trials, ξ=0.01 and

ξ=0.03 were applied respectively to obtain proper training data sets, beginning with the LDA result. Neural networks with 15 hidden nodes were employed, and the learning rate was set at 0.1. The final result including the neural network weights and classification results for the two trials is exhibited in Appendix IV (2). A comparison of the results obtained in this experiment and the LDA results is shown in Table 3.2. The experiment shows that the classification result of the MF model improves on the LDA method, although the results of the MF model were based on a limited number of trial and error processes in the determination of ξ.

		LDA	MFM	Reduced misclassifications
Train on the original data set (252 point) and test on the same data set		(52+131)/252 = 72.6 %		
Trial 1. Train on subset S1 (126 points)	Test on S1	(26+67)/126 = 73.8 %	(26+77)/126 = 81.7 %	(10/33)= 30.3 %
	Test on S2	(24+71)/126 = 75.4 %	(20+76)/126 = 76.2 %	(1/31)= 3.2 %
	Overall	74.6 %	79.0 %	
Trial 2. Train on subset S2 (126 points)	Test on S1	(31+56)/126 = 69.0 %	(25+67)/126 = 73.0	(5/39)= 12.8 %
	Test on S2	(26+58)/126 = 66.7 %	(24+71)/126 = 75.4 %	(11/42)= 26.2 %
	Overall	67.9 %	74.2 %	
Overall		71.3 %	76.6 %	

Table 3.2. A Comparison of the Percentage Correctly Classified by the MF Model And the LDA Method on the Alpha TV Commercial Study Data [Green 1978]

3.8. DISCUSSION

3.8.1. EFFICIENCY OF THE MF MODEL

A well-known problem with neural networks using the standard BP training algorithm is the significant computational time required to reach a convergent result [Specht 1990]. One may expect that, on average, the more complex the y-surface which will be represented by the neural network, the more iterations will be required during the learning process (cf. [Wieland and Leighton 1987]). In the MF model developed in this research, an underlying principle is to reduce unnecessary, even harmful, complexity of the y-surface which can result from unconstrained learning with statistical data. Using the MF algorithm, the neural network can generate the less complex y-surface easily, given the proper training data set. This can be explained by an example. Suppose we have two sample data points which have the same pattern vector **X** and the opposite outcomes of c_1 and c_2 respectively. From the statistical point of view, this phenomenon is the result of statistical fluctuations. Using the neural network with the standard BPLMS algorithm, a correct classification result can never be obtained. Suppose we change the example slightly, and the two sample points have the same outcomes as before, but with very slightly different pattern vector values, say, X and $\mathbf{X}\pm$ respectively. In this case, a neural network using the standard BP algorithm would take a long time to generate a boundary to separate the two points. However, the MF model deals with this

problem in a more astute manner. It first pre-processes the data fully, taking the statistical properties of both conflict points into account, and obtains a proper training data set. One of the conflict data points would be ignored in further learning, but would be classified as found in the pre-process result. Then the neural network would learn from the pre-processed training data set, which has fewer statistical fluctuations, and the generated y-surface would have a less complex form. The learning nature of a neural network under monotonic constraints is almost the same as the standard BP neural network, but when the training data set does not result in a monotonic boundary the learning will never be complete. In terms of neural network learning itself, the learning time of a neural network under monotonic constraints depends on the number of hidden nodes, and the dimension of the pattern variable, as well as the potential complexity of the resulting classification boundary. However, as discussed above, data pre-processing in the MF model reduces unnecessary boundary complexity. As a result, the MF model approach is usually much more efficient than the standard BP method. Neural networks under monotonic constraints in the examples using the MF model or its extensions (see Chapter 4, 5, 6) throughout this thesis were implemented with Turbo C program and executed on an IBM PS/2 microcomputer (15MHz, 80386 CPU and co-processor). Learning time ranged between 1 and 5 hours. Experiments suggested that, if the learning time lasted too long, the monotonic condition was more likely to be violated, and vector analysis should be repeated. Note that, once learning is complete, a neural network classifier is as fast as LDA in classifying new observations.

3.8.2. ROBUSTNESS OF THE MF MODEL

When developing a model dealing with statistical data, robustness is often used to evaluate the model from the statistical point of view. Robustness signifies insensitivity to small deviations from the assumed underlying situation, including randomness, independence, distribution functions, etc. [Huber 1981]. Although it is difficult to prove the present models' robustness directly since statistical tools are hard to apply, the robustness of the MF models can be compared to the LDA statistical model results. The MF model developed in the present research is designed to further reduce misclassification rates from the LDA classification technique. Given the fact that there exists a large amount of literature to explain the robustness of the LDA (e.g. [Lachenbruch 1975]), robustness arguments regarding the present basic model could be stated as follows: the robustness of the present model is at least as good as the LDA in terms of low misclassification rates on training data sets. The limited results obtained so far indicates that there is also an improvement relative to LDA in test data sets.

LDA is used as a pre-processing tool in this thesis primarily because of considerations involving robustness. Actually, there are a number of linear classifiers (e.g. the perceptron) which are able to pre-process statistical data, although some of them are less well grounded theoretically than the LDA technique.

CHAPTER FOUR

LEARNING BIAS IN NEURAL NETWORKS AND AN APPROACH TO REDUCE ITS EFFECT

4.1. MONOTONICALLY SEPARABLE PROBLEMS

In this chapter we will first discuss an extreme case of classification -- the completely separable problem, and develop the Monotonically Separable Problem Model for this type of problem. The motivation for this investigation is the need to explain the learning bias issue. The ultimate goal of this discussion is to incorporate this model into the Monotonic Function Model developed in Chapter 3, and to produce a more comprehensive approach for dealing with general managerial classification problems, as described later in this chapter.

If several classes can be completely separated by linear hyperplanes, they are called linearly separable; and if they can be separated by a polynomial surface, then the problem is called polynomially separable [Bremermann 1976]. As pointed out earlier, the "true" boundary shape or the "true" boundary function is rarely known in advance in managerial pattern recognition. Consequently, we need to extend the concept of separability in a more general sense to

managerial pattern recognition, based on the definition of monotonically separable problems as follows:

> In managerial pattern recognition, if the classes can be completely separated by monotonic surfaces (hypersurfaces), then the problem is called monotonically separable.

Figure 4.1 shows an example of monotonically separable classes.

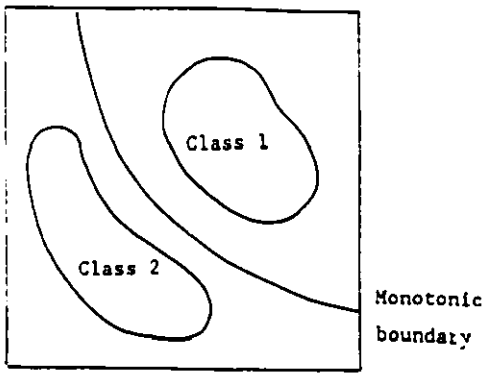


Fig. 4.1. A Monotonically Separable Classification Problem

In principle, if we do not assume probability distribution functions for the classes, since the actual probability distribution functions are rarely known in managerial classification, then a boundary which perfectly separates the classes might be considered the most desired solution, since this leads to zero misclassification. From Figure 4.1, it is easy to see that there is usually uncertainty regarding the location of the resulting boundary, especially if the two classes are relatively well separated, since there is a gap between the two classes. In this chapter, we describe ways of resolving this uncertainty. The remainder of this chapter proceeds as follows. Section 4.2 introduces the concept of learning bias in neural

networks. Section 4.3 describes a specific boundary generation behavior of neural networks, and concludes that a standard neural network with BPLMS is poor at solving this uncertainty problem. Section 4.4 explains a useful property of managerial pattern recognition problems, and Section 4.5 develops the Monotonically Separable Problem Model (MSP Model) for handling such problems. Section 4.6 discusses the relationship between the MF and MSP models developed in Chapter 3 and Chapter 4 respectively. Finally, Section 4.7 discusses neural network architecture issues concerning the minimum number of hidden nodes.

4.2. LEARNING BIAS

It is commonly accepted that the behavior of the neural network learning process is relatively unpredictable (cf. e.g.[Soulie et al. 1987]). This means that the classification decision boundary is not determined only by the statistical constitution of the training data, but is also influenced by other factors. These factors include the following:

a. Architecture of the neural network model (e.g. number of hidden nodes),

b. Parameters (e.g. learning rate) of the learning algorithm,

c. Initial state of the neural network,

d. Sequence of training data points,

e. The stopping criteria of the learning procedure.

These factors bring some inherent knowledge or learning rules to bear on the machine learning process. Because these may or

may not be pertinent to a particular task or a specific problem, they are referred to as learning bias in the artificial intelligence literature [Utgoff 1986].

The concept of bias used here differs from that of estimation bias in statistics. The latter refers to the discrepancy between the mathematical expectation of a classification boundary and the true one, if it is discussed in the classification context. The neural network classification techniques we are discussing do not rely on knowledge about probability distributions, and the classification boundary generated by a neural network is not a statistic of the pattern vectors (see Section 3.3.3). Therefore, discussion of the estimation bias of classification boundaries in neural networks is not appropriate here. Nevertheless, there are some common characteristics between learning bias and "biased" classification boundary results. As will be seen, neural networks with their individual learning bias do not generate identical classification boundaries from the same training data sets. This implies that the classification boundary generated by a standard neural network is most likely to be biased because we have no knowledge about how to set the learning bias in order to generate an "unbiased" classification boundary. The next three sections of this chapter show the learning bias effect on classification boundary generation, and suggest a method to reduce this effect during the learning process.

4.3. EFFECT OF LEARNING BIAS

In the BPLMS learning algorithm, the neural network weights are gradually modified according to the current training sample data, the current neural network state, and the currently adopted learning rate (see Chapter 2). According to this algorithm, the learning procedure stops whenever a final training sample point is correctly classified; that is, the error between the current classification result and the desired value falls within a pre-defined tolerance. This stopping criterion is considered to be a strong bias factor since, as will be shown later, it often has a synergy with other bias factors. The following example explains the effect of learning bias in neural network classification. The example will show that the closeness of the final classification boundary to the final training sample point is a function of the learning rate η (see Figure 4.2(a) and (b)).

In this example, a one-hidden-layer and single-output-node neural network model (Figure 3.2) is considered. Using the notation in Figure 3.2, according to the layered neural network model with a temperature of 1, the relationship between **X** and y is:

$$y = \{ 1+\exp[-(v_0 + \sum_{i=1}^{h} v_i(1+\exp(-\mathbf{U}_i\mathbf{X}'))^{-1})]\}^{-1} \qquad (4.1)$$

(see equation (3.1), Chapter 3). As usual, suppose that the sample in class 1 has a desired output of 1 and the sample in class 2 has a desired output of 0, then the decision rule is that if, given a pattern vector **X** of an observation s, the output of the neural

network $y_s > 0.5$, then classify s as class 1; otherwise, classify s as class 2.

We now consider the last learning event. It is reasonable to assume that, just before the last learning event of point s in class 1, the classification boundary is very close to s but does not correctly classify it; that is,

$$y_s = 0.5- \quad . \tag{4.2}$$

From (4.1) and (4.2), we have

$$v_0 + \sum_{i=1}^{h} v_i \left[1 + \exp\left(-U_i X_s' \right) \right]^{-1} = 0- \quad . \tag{4.3}$$

According to the BPLMS learning algorithm, after learning s, the weights are modified based on error signals, and the output of the neural network corresponding to the pattern vector X_s becomes

$$\begin{aligned} y_s^1 = \Bigg\{ 1 + \exp\Bigg[-\Bigg(v_0 + \sum_{i=1}^{h} \Big(\Big[v_i + \eta\, v_s\, (1-y_s)^2 \\ * \left(1 + \exp\left(-U_i X_s'\right)\right)^{-1} \Big] \\ * \Big[1 + \exp\Big(-\Big(U_i + \eta\, v_i\, \left(1 + \exp(-U_i X_s')\right)^{-1} \\ * \left(1-(1+\exp(-U_i X_s'))^{-1}\right) y_s (1-y_s)^2 X_s \Big) X_s' \Big) \Big]^{-1} \Big) \Bigg) \Bigg] \Bigg\}^{-1} \end{aligned} \tag{4.4}$$

(see equations (3.3) to (3.7), Chapter 3). The superscript of 1 on y_s indicates the subsequent learning event. From (4.2) and (4.4), we have

$$y_s^1 \approx \left\{1+\exp\left[-\left\{v_0+\sum_{i=1}^{h}\left\{[v_i+0.125\eta(1+\exp(-U_iX_s'))^{-1}]\right.\right.\right.$$

$$\left.\left.\left.* [1+\exp(-(U_i+0.125\ \eta\ v_i(1+\exp(-U_iX_s'))^{-1}\right.\right.\right.$$

$$\left.\left.\left.* (1-(1+\exp(-U_iX_s'))^{-1})\ X_s)\ X_s')]^{-1}\right\}\right\}\right\}^{-1} .$$

Considering the case of reasonably small η (say, 0.1), we have

$$y_s^1 \approx \left\{1+\exp\left[-\left\{v_0+\sum_{i=1}^{h}\left\{[v_i+0.0125(1+\exp(-U_iX_s'))^{-1}]\right.\right.\right.$$

$$\left.\left.\left.* [1+\exp(-(U_iX_s'))]^{-1}\right\}\right\}\right\}^{-1} \qquad (4.5)$$

since $\quad |0.0125v_i(1+\exp(-U_iX_s'))^{-1}$

$$* (1-(1+\exp(U_iX_s'))^{-1})\ X_sX_s'| \le 0.003125|v_i|,$$

which is usually very small.

From (4.3) and (4.5), we have

$$y_s^1 \approx (1+\exp[\ -0.0125\sum_{i=1}^{h}(1+\exp(-U_iX_s'))^{-2}])^{-1} . \qquad (4.6)$$

Taking the average case into account, suppose that

$$(1+\exp\ (-U_iX_s'))^{-1} \approx 0.5 \quad .$$

Then

$$y_s^1 \approx [\ 1+\exp\ (\ -0.003125\ h\ )\ ]^{-1} \qquad (4.7)$$

where h is the number of hidden nodes. Considering an example of moderate h=10, we have

$$y_s^1 \approx 0.508 \ ; \quad \text{and}$$

$$\Delta y_s = y_s^1 - y_s \approx 0.008.$$

This means that the boundary usually does not move very far from the last learned point s. That is, when a small learning rate η is used, the classification boundary is most likely to be close to the final training sample point (Figure 4.2(a),(b)). Although we have discussed this issue based on a one-hidden-layer and one-output-node neural network, this conclusion is generally true for layered neural networks, and this phenomenon is readily observed in experiments. Obviously, the "true" classification boundary of an arbitrary problem does not necessarily have this property (i.e. close to a particular sample point). It can therefore be concluded that the learning bias factor of setting the learning rate η has an influence on the classification boundary result.

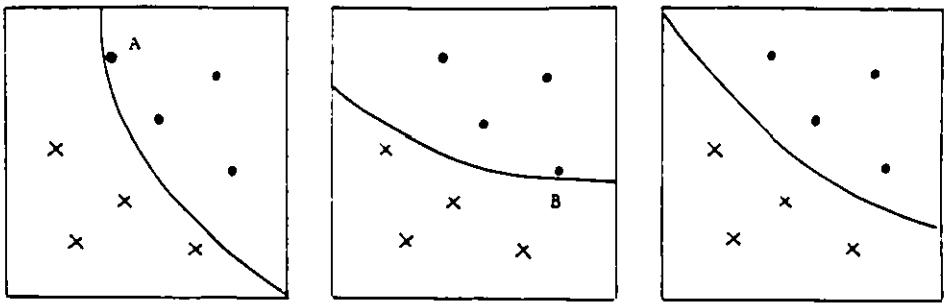


Fig. 4.2. Effect of Small Learning Rate η

In the above discussion, we concentrated on the bias effect caused by the learning rate η, while making some reasonable assumptions about moderate values of U_i (in equation 4.6) and h (in equation 4.7). In fact, all factors constituting learning bias

listed in Section 4.1 have an influence on the boundary generation behavior. For instance, if we fix the learning rate η , we find from equation (4.7) that the larger the number of hidden nodes h, the larger Δy_s will be, and the farther away the final boundary will be from the last learned point s, when the remaining underlying assumptions of equation (4.7) hold. As well, from equation (4.6), we find that Δy_s also depends on the current state of the neural network, namely, the weight matrix $\mathbf{U}_i$'s. Because the current state in turn depends on the initial state of the neural network, especially when the number of learning iterations is small, Δy_s may be influenced by the initial state of the neural network. Moreover, since the sequence of training examples may play an important part in determination of which sample point is the last learned one (s in the above discussion), this sequence will also have an influence on the final classification result.

The discussion above has shown that neural networks with their individual bias factors may generate boundaries in many different ways for the same sample data set, even when each of these boundaries can completely separate the two classes. As there is no knowledge about how to set the bias factors for a particular problem, it is difficult to find a general methodology to judge which boundary is the "best". Intuitively, it appears to be desirable to find an "unbiased" boundary which lies centrally between the two classes (see Figure 4.2(c)). Yet, consideration should not be limited to finding a single "unbiased" boundary, but also to supply the decision maker with more information about the uncertainty regarding the gap between

the two classes. This consideration is analogous to statistical confidence interval estimation.

4.4. A TYPE OF AMBIGUITY IN MANAGERIAL PATTERN RECOGNITION

In order to find a method to reduce the effect of learning bias and supply proper classification information for decision making, the major properties of monotonic managerial pattern recognition (see Chapter 3) are briefly reviewed as follows. From the point of view of preference analysis or utility [Keeney and Raiffa 1976], managerial pattern recognition for two classes is basically a problem involving two sets of decision acts: feasible (or preferred) and not-feasible (not-preferred). For example, the classification of creditworthiness is a decision to determine whether credit should be extended to a customer based on the given descriptions of the customer. Relative to this point, dominance is an important property of preference analysis [Keeney and Raiffa 1976]; that is, assuming that the preference increases in the same direction for all x_j's ,

If $x_j' \geq x_j''$ for all j=1...m , and

$x_j' > x_j''$ for some j,

then X' $(x_1' \ldots x_m')$ dominates X'' $(x_1'' \ldots x_m'')$,

since X'' is a noncontender for the "best" act.

In Figure 4.3(a), let S_1 and S_2 be the sets of consequences associated with the decision acts "preferred" and "not-preferred", respectively. The subset S_2^* that dominates all other consequences in S_2 is called the efficient frontier of S_2 . Since all of the points on S_2^* will result in the same decision ("just not-preferred"), S_2^* is an indifference curve [Keeney and Raiffa 1976, p79]. Similarly, the subset S_1^* that is dominated by all other consequences in S_1 is the efficient frontier of S_1 . S_1^* is also an indifference curve representing the decision of "just preferred" . If the sample data correctly represent the consequences associated with the managerial decision, then S_1^* and S_2^* never intersect [Keeney and Raiffa 1976]; and the two classes are considered monotonically separable.

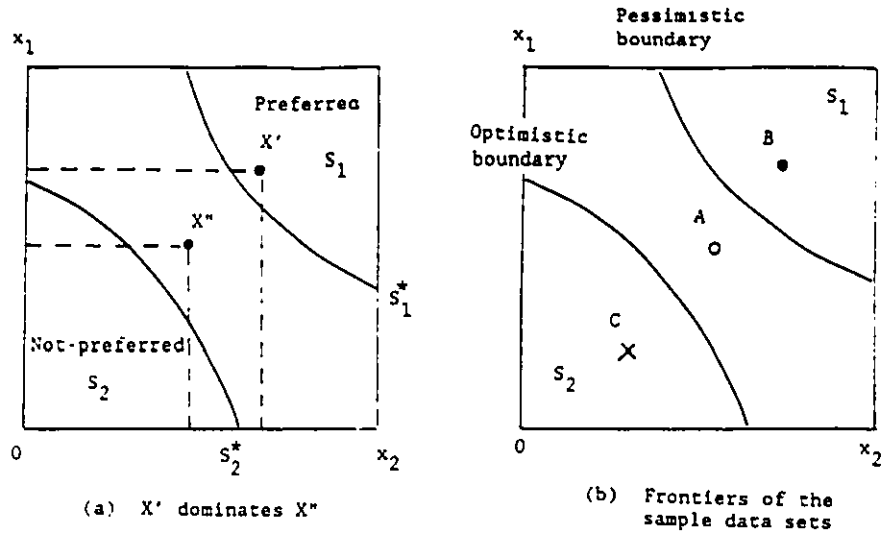


Fig. 4.3. Frontiers of Decision Consequences

Based on the dominance property, the sample points which constitute the frontiers of the corresponding sample sets are found simply by comparison of the x_j's (see Figure 4.3(a)). If two curves (or surfaces, or hypersurfaces, depending on the dimensionality of **X**) pass through the frontier points of the sample sets, they are the special cases belonging to the set of all classification boundaries for this problems. The frontier of the sample points in the "not-preferred" class is an "optimistic" boundary, as a new observation would tend to be classified optimistically. Similarly, the frontier of the sample points in the "preferred" class is a "pessimistic" classification boundary. Suppose that a new event A to be classified is observed in the space between the two frontiers (see Figure 4.3(b)). This results in an ambiguity where, according to the

"optimistic" boundary, A should be preferred since A is not surrounded by the "not-preferred" frontier based on the available data. However, according to the "pessimistic" boundary, A should be classified in the not-preferred class since it is not surrounded by the "preferred" frontier.

The fact is that, in the classification of monotonically separable problems, neural networks arbitrarily generate boundaries which fall in the region between the two frontiers. Neural networks suffer from learning bias, and they cannot generate an "unbiased" boundary. They are therefore unable to resolve the ambiguity problem discussed above. The next section will suggest a model to overcome this deficiency, based on monotonicity conditions.

4.5. A MODEL TO REDUCE THE EFFECT OF LEARNING BIAS

4.5.1. MONOTONICALLY SEPARABLE PROBLEM MODEL

As discussed in the previous sections, a single standard neural network is not likely to be able to generate an "unbiased" boundary due to learning bias. As well, a single standard neural network does not supply any information about the gap between the two frontiers. On the other hand, the two frontiers specify the classification of a new observation to some degree (e.g. preferred - questionable - not-preferred). It therefore seems natural to take the two frontiers into consideration, by employing two neural networks to implement the two frontiers. The model which includes

this feature is called the Monotonically Separable Problem (MSP) Model, and is depicted in Figure 4.4.

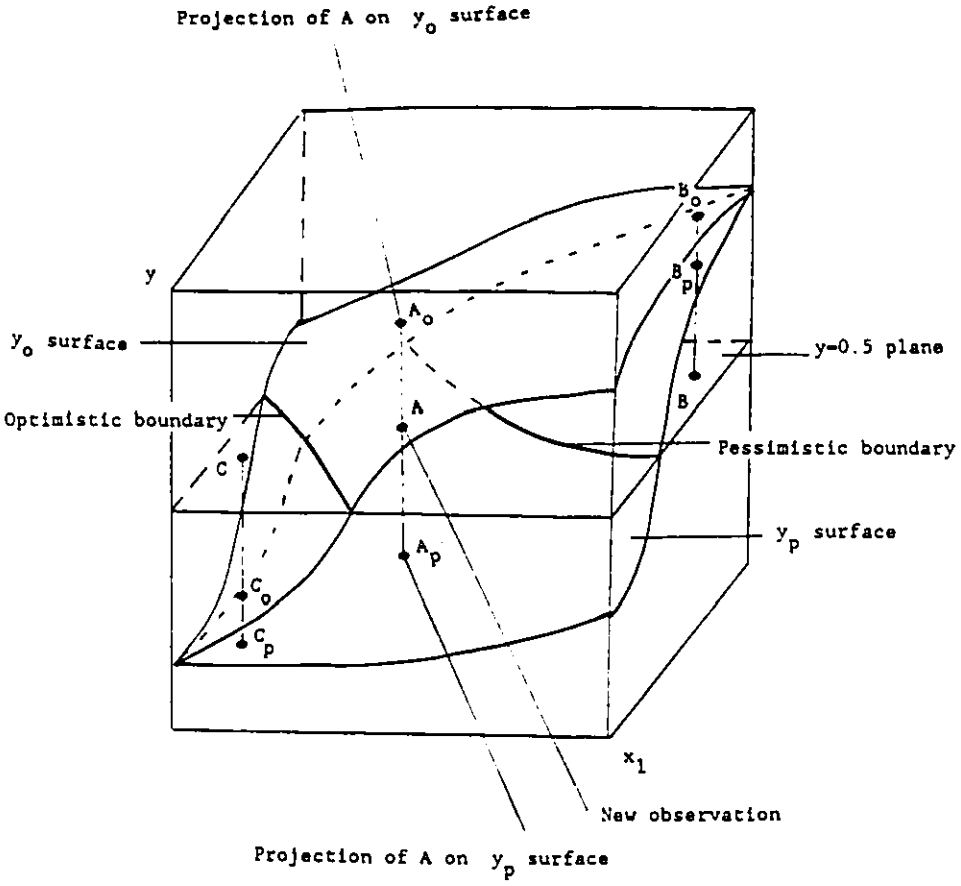


Fig. 4.4. Monotonically Separable Problem Model

Suppose that we have two neural networks, say, NN_o and NN_p, which generate optimistic and pessimistic frontiers respectively. The classifier consisting of the two neural networks supplies more complete information than a single neural network classifier which will contain unknown learning bias effects. For example, if a new

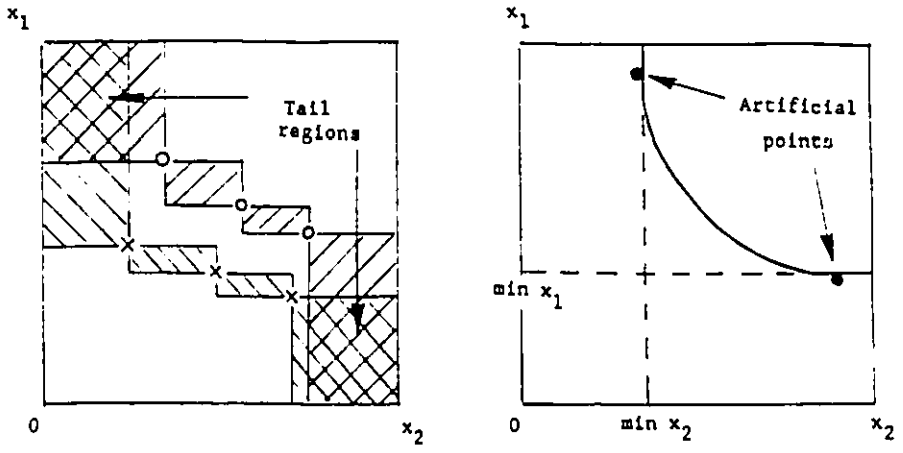
observation point is on the same side of the two frontiers (e.g. B or C in Figures 4.3(b), 4.4), its class would be decided with more assurance. Otherwise (e.g. A in Figures 4.3(b), 4.4), the decision maker would be less certain, and would have to more carefully consider other contingent situations which are not necessarily included in the pattern variable data.

Because the monotonicity and dominance properties do not define the frontier curve (or surface, or hypersurface) itself, there still is a range within which a frontier could arbitrarily be located (shaded regions in Figure 4.5(a)). The generation of frontiers by a neural network is also influenced by learning bias. Obviously, at this time the uncertainty regions of the frontiers should be much smaller than the entire gap between two possible frontiers, as shown in Figure 4.5(a). In practice one could expect to obtain frontiers with low curvature in the usual cases (cf. [Wieland and Leighton 1987]). Furthermore, we still can reduce the effect of learning bias if additional knowledge is available, as illustrated by the following instance. In practice, sample points with extremely high or extremely low pattern variable values are usually not available. This results in a large range of the possible frontiers that fall in the boundary tails (see Figure 4.5(a)). On the other hand, in managerial applications some knowledge about the extreme values of each pattern variable can often be estimated. In creditworthiness classification, for example, the decision maker might say that

IF and ONLY IF
salary is greater than 20,000
THEN
the person could be creditworthy.

Such a control could be implemented by adding artificial training sample data (see Figure 4.5(b)). In the above example several points ($X|x_{salary}$=20000) could be introduced in the tail regions of the "preferred" frontier. These artificial training data points force the neural network to generate a more specifically meaningful frontier during the learning process.

Note that we have thus far introduced two ways of reducing the unpredictability of boundary generation for a neural network. One method (Chapter 3) uses the monotonicity constraint imposed on the learning machine. The method discussed here trains the neural network directly with explicit knowledge represented by artificial data training points.



(a) Range of possible sample frontiers

(b) Reduce the bias effect by adding knowledge

Fig. 4.5. Determining the Frontiers of Sample Sets

4.5.2. THE INTEGRATED NEURAL NETWORK TO SUPPLY ASSURANCE INFORMATION

In practice, users would like to have an integrated neural network to supply assurance information for a new observation between or near the frontiers. To do so, we may combine the two neural networks NN_o and NN_p into one as shown in Figure 4.6. Suppose that the outputs of the two neural networks have the same range, say, $[y_{min}$-0.1, y_{max}-0.9], and the classification score on their generated frontiers is 0.5. Given an observation s between or near the sample frontiers, the output pair $[(y_p)_s , (y_o)_s]$ gives a degree of assurance through the relation

$$y_{min} < (y_p)_s < y_s < (y_o)_s < y_{max} .$$

Consider

$$y_s = [(y_p)_s + (y_o)_s] / 2 ,$$

where

$$y_s = [(y_p)_s + (y_o)_s] / 2 = 0.5$$

if s is at a supposed "unbiased" boundary. This gives

$$y_p + y_o = 1 \qquad (4.8)$$

at the "unbiased" boundary, which will reduce learning bias (see Figure 4.7).

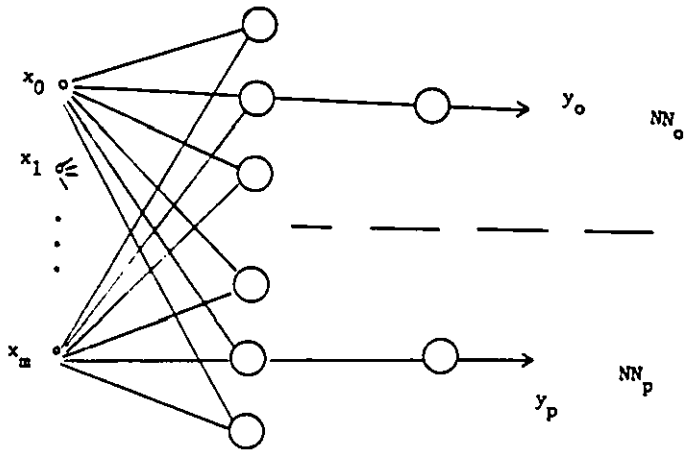


Fig. 4.6. The Integrated Neural Network to Supply Assurance Information

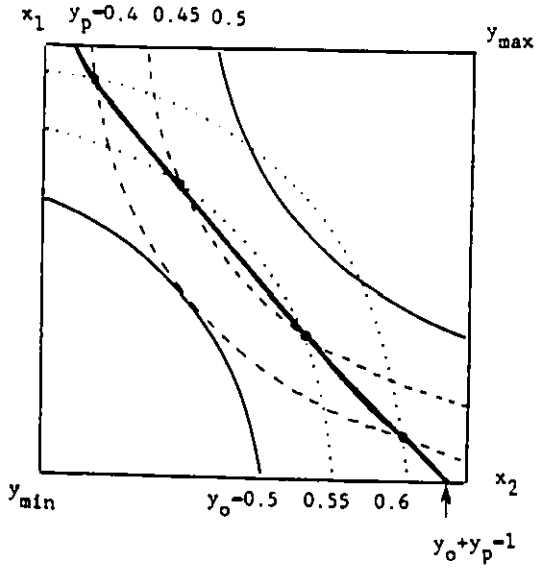


Fig. 4.7. "Unbiased" Boundary

The steps to implement the integrated MSP neural network are briefly summarized as follows.

(1) Apply the dominance property to find the frontier point sets S_1^* and S_2^* for the two classes of sample data sets S_1 and S_2 respectively.

Add artificial points in tails where necessary and appropriate.

(2) Build a neural network with one hidden layer and two output nodes.

The neural network is evenly divided into two parts, say NN_1 and NN_2 .

(3) Train NN_1 with S_1^* , and train NN_2 with S_2^* such that, for i=1,2,

$y_i(x_1=\ldots=x_m=0)=0+$, and $y_i(x_1=\ldots=x_m=1)=1-$,

and $y_i(X_s \mid s\epsilon S_i^*)=0.5$. The monotonicity constraint is imposed during training.

Then, given a new observation between or around the sample frontiers, the pair $[y_1 , y_2]$, where y_1 and y_2 are the outputs of NN_1 and NN_2 , respectively, supplies the assurance information.

4.6. THE RELATIONSHIP BETWEEN THE MONOTONIC FUNCTION (MF) MODEL AND THE MONOTONICALLY SEPARABLE PROBLEM (MSP) MODEL

The MSP model described above is for solving monotonically separable problems. However, when the sample points in the two classes overlap, the frontiers of the two data sets cannot be simply determined according to the MSP model. On the other hand, the MF model (Chapter 3) is able to solve problems with overlapping data; but when we use the MF model, we find that there always exists a region within which the class of all boundaries with the same misclassification rate lies, given the fact that the number of sample points is finite. The magnitude of the uncertainty region depends on particular circumstances, as shown in Figure 4.8. Therefore, given a sample set to be learned, we can first use the MF model to identify the sample points which carry significant statistical fluctuations. We can then find a monotonic boundary to completely separate the rest of the sample set with the lowest possible misclassification rate, then treat the problem as a monotonically separable problem and use the MSP model to solve it, as shown in Figure 4.8(b)(c). The final result will be less biased than that of the MF model by itself.

Figure 4.9 shows an experimental result using the experimental data described in Section 3.7.1 (compare Figure 4.9 with Figure 3.11). Using the combination of the MF and MSP models, we obtained a 90% correct classification rate, where a large fraction of the true boundary falls in the uncertainty region generated by the MSP model. In this drawing, misclassified data points are underlined.

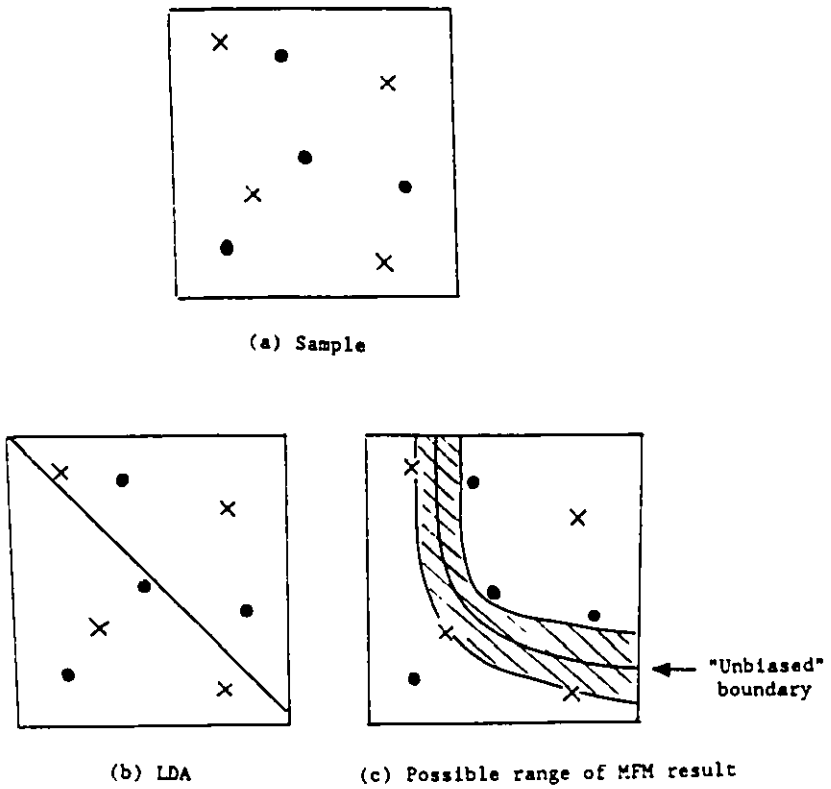


Fig. 4.8. Example of the Uncertainty of Boundary Generation

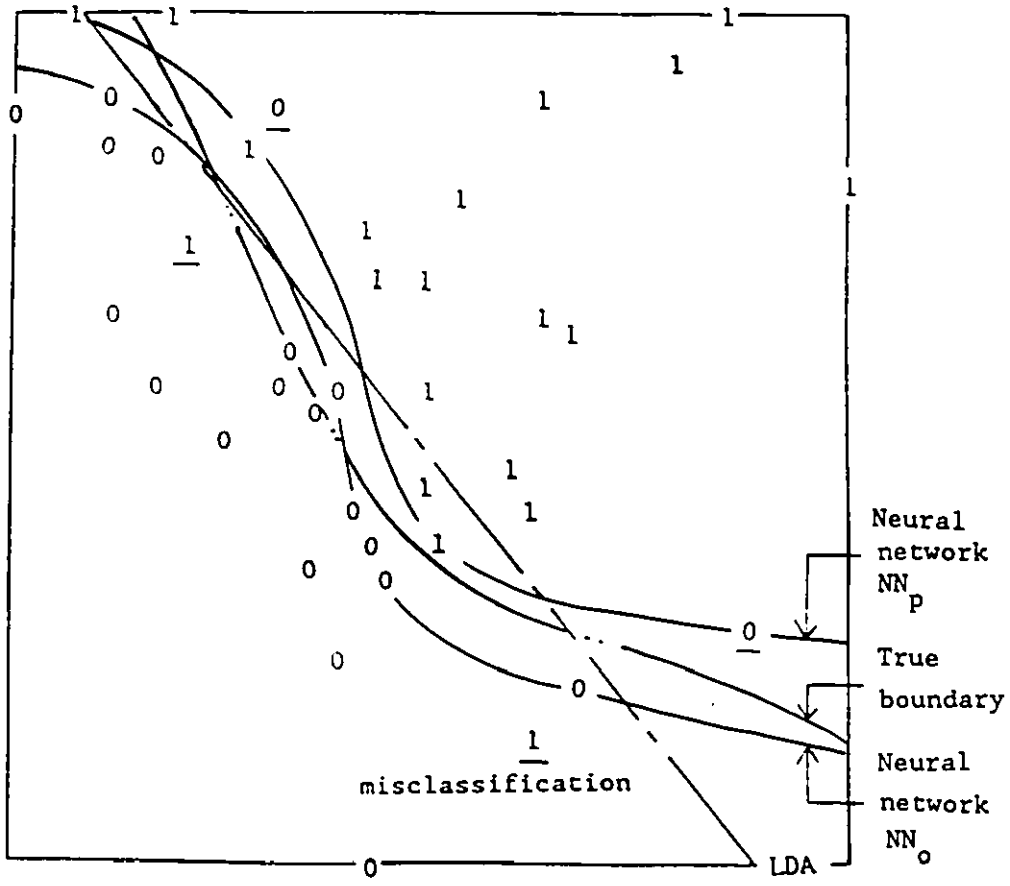


Fig. 4.9. Experimental Result with Combined MF and MSP Models

(Compare with Fig. 3.11)

4.7. MINIMUM NUMBER OF HIDDEN NODES

When implementing the MSP model, if we have found the frontiers of the sample data sets, then we face the additional question of how many hidden nodes are required in order to generate the desired boundary. The neural network with too few hidden nodes is not able to generate sufficiently complex boundary functions. However, one always wants to keep the number of hidden nodes h as small as possible, since larger h results in increased computation time. This is therefore a neural network capacity problem. There is published research on binary node neural network capacity, based on entropy theory (e.g. [Denker and Wittner 1987]), but theoretical estimates for the continuous sigmoid logic case are not available [Denker and Wittner 1987] [Sietsma and Dow 1988]. A formal discussion of the minimum number of hidden nodes is beyond the scope of this research. The present discussion is therefore informal, and based on practical considerations of the required number of hidden nodes for sigmoid logic neural network models. We work from the standpoint that the neural network is a good computational device for non-linear interpolation of data [Wieland and Leighton 1987].

Consider the typical neural network with one hidden layer and a single output node. Suppose that we have identified the set of points $S^* \subseteq S$ through which the true boundary must pass. Let the cardinality of S^* be ν^*. Recall (equation (3.2)) that the boundary function representing the relationship between v_i's , U_i's , and **X** is

$$v_0 + \sum_{i=1}^{h} v_i (1 + \exp(- U_i X'))^{-1} = 0 \qquad (4.9)$$

Equation (4.9) is used to determine the weights v_i and the U_i's based on the known **X** values. It is a non-linear equation with (h+1) variables v_i and h(m+1) variables w_{ij} (constituting the U_i's). This totals h(m+2)+1 variables, where m is the pattern vector dimension, and h is the number of hidden nodes. Consider that each point in S^* has its individual **X** value, and each **X** value determines one relation between the v_i's and w_{ij}'s. We thus have ν^* equations for the h(m+2)+1 variables:

$$v_0 + \sum_{i=1}^{h} v_i (1 + \exp(- U_i X_1'))^{-1} = 0$$

$$v_0 + \sum_{i=1}^{h} v_i (1 + \exp(- U_i X_2'))^{-1} = 0$$

. . . . . .

$$v_0 + \sum_{i=1}^{h} v_i (1 + \exp(- U_i X_{\nu^*}'))^{-1} = 0 \qquad (4.10)$$

Assume that, to solve this coupled set of equations, each equation can eliminate one variable. (Note that this assumption does not hold strictly in the non-linear equation case. However, if $U_i X_r$, i=1...h, could be considered approximately constant for all r=1...ν^* during the back propagation learning process (Kung & Hwang 1988), then the above coupled set of non-linear equations would degenerate into a linear set of equations). If ν^* exceeds h(m+2)+1, under this

assumption equation (4.9) will be underdetermined. Accordingly, the minimum number of hidden nodes required to generate a boundary passing through all of the points in S^* is:

$$h_{min} = (\nu^* - 1) / (m + 2) \qquad (4.11)$$

If h is less than h_{min} the neural network does not have the capacity to generate the desired boundary, since it would not have the required curvature to pass through all the points. The solution is to increase h.

Note that equation (4.9) itself does not define which part of the pattern space is in class 1 or class 2. The solution of the equation only determines the boundary function. As well, the resulting y-surface is not necessarily monotonic. The monotonic constraint also makes the issue more complicated. However, using more hidden nodes will improve the ability to generate the desired monotonic y-surface. At the same time, more hidden nodes increase computation time, so there is a need to select the number of nodes so that the desired surface can be generated during the training process without unnecessarily compromising computing efficiency.

The conclusion from equation (4.11) is approximate, and only shows that h_{min} is relatively sensitive to ν^*. Nevertheless, the following relevant arguments are based on the above discussion.

(1) If we consider a neural network as a computing tool in non-linear interpolation, then its capacity, represented by the number of hidden nodes (h), determines the complexity of the boundary function

which can be represented by the number of points (ν^*) on the boundary.

(2) Considering that the frontier of the sample data set is a kind of boundary as discussed in Section 4.4, the number of points in the frontier set S^* should be taken into account in designing the neural network structure for the MSP model.

In the case of problems to be solved by the MF model, the data set can be pre-processed, and the minimum number of hidden nodes can be determined from equation (4.11) and a count of the frontier points (Figures 4.8, 4.9).

CHAPTER FIVE

FUZZY SET REPRESENTATION OF NEURAL NETWORK CLASSIFICATION BOUNDARIES

The ultimate objective of traditional statistical classification methods is to find a clear cut-off classification boundary to divide the pattern space into two or more decision or classification regions based on some pre-defined criterion [Nilsson 1965] [Young and Calvert 1974]. Since fuzzy set theory was suggested in the 1960s [Zadeh 1965], pattern recognition problems have been intensively studied in the fuzzy set sense, especially when applying pattern recognition concepts in the social context [Bossel et al. 1976]. In fuzzy theory, class membership is not binary, but is represented by the value of a gradually changing function which can take on intermediate values between 0 and 1. In this way a pattern class need not have a sharp cut-off but may have a gradual fade-out [Bremermann 1976]. The major attractions of fuzzy set theory in pattern recognition are twofold. First, it is difficult, if not impossible, to find a "true" or optimal clear cut-off classification boundary in a real problem (see Section 3.1 discussion). Decision makers also need information about uncertainty for particular real events. Secondly, considering pattern recognition as a model for

cognitive processes, the use of fuzzy sets is a promising approach to providing imprecise class membership information [Bossel et al. 1976] [Zadeh 1984], especially in the case where probability theory is difficult to apply directly.

There have been several studies associating neural networks with fuzzy set theory [Klimasauskas 1989]. Kosko [Kosko 1987], for example, suggested combining fuzzy knowledge with neural networks in expert system reasoning. Shiue [Shiue 1987] and Keller [Keller 1985] used fuzzy set theory in designing learning algorithms for neural networks and perceptrons, respectively. However, research on representing fuzzy membership in neural network classification problems is rare.

This chapter develops a neural network model to represent fuzzy membership functions in managerial pattern recognition. The remainder of the chapter proceeds as follows. Section 5.1 describes the boundary representation problem in statistical classification. Section 5.2 briefly reviews fuzzy theory concepts. Section 5.3 describes how a fuzzy boundary relates to neural network classification, and Section 5.4 is a general discussion of the suggested approach.

5.1. A PROBLEM IN STATISTICAL CLASSIFICATION

In order to explore problems with statistical classification techniques, we begin with the two class linear discriminant analysis (LDA) classification method. Suppose that a

linear boundary $y = \sum_{j=1}^{m} b_j x_j$ separates the pattern space into two regions as shown in Figure 5.1. The linear boundary is optimal only under the assumption that the sample data have multivariate normal distributions with common covariance [Lachenbruch 1975].

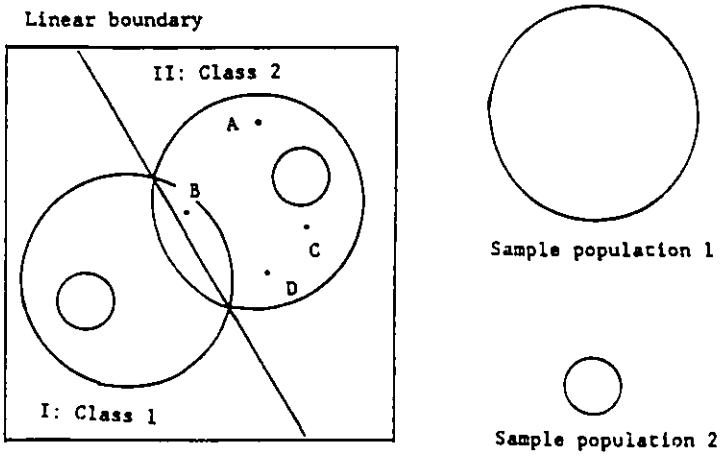


Fig. 5.1. Linear Discriminant Analysis

It is worth noting two closely related characteristics of the LDA result. First, the linear boundary itself reveals nothing about the statistical behavior of the sample data distributions. For instance, the two very different sample populations in Figure 5.1 can theoretically result in the same classification boundary. In one situation, observations are widely distributed over the pattern space and there is a large overlap between the two data set classes, but in the other situation there is little if any overlap. In statistical classification techniques, information reflecting the overlap of the data classes is provided through error rate estimates. In LDA, the error rate estimate is $\Phi(-A/2)$, where Φ is the cumulative normal

distribution function, $A^2 = (\bar{X}_1-\bar{X}_2)\, S^{-1}(\bar{X}_1-\bar{X}_2)'$, $\bar{X}_1$ and $\bar{X}_2$ are the sample means of the two pattern vectors, and S is the sample covariance matrix [Lachenbruch 1975]. Like other statistical methods, the error rate estimate in the LDA is based on the statistical behavior of the entire sample set of both classes. This brings up the second characteristic of LDA; that is, the probability of correct classification, or the probability of misclassification, is defined as a property of the two regions of the pattern space which are divided by the sharp boundary, rather than the property of a particular observation. For instance, in Figure 5.1, if an observation belonging to class 1 is observed in region I, then the probability of correctly classifying it is, say, 90%, (or, the misclassification rate is 10%, equivalent to Type I error in the hypothesis test context,) no matter whether the observation is A or B. In other words, probability does not supply information about the "likeness", or membership, of a particular point belonging to its class. One may calculate the classification score according to the linear function $y = \sum_{j=1}^{m} b_j x_j$, which may represent a kind of "likeness" of an observation belonging to its class. However, as pointed out earlier, the linear function itself does not carry any information about the overlap degree of the two data set classes; hence, the classification score can not completely represent the membership which associates each observation with its class. These two characteristics of the LDA result suggest that uncertainty

information may be provided for a particular point in a more natural way.

The above discussion is based on the linear discriminant analysis case; however, the general argument extends to all statistical classification methods. In fact, any sharp boundary function suffers from the same problem discussed above. Pure statistical tools are not able to solve the membership problem, especially in the cases where the precise estimation of the probability distribution is not possible.

Although a neural network itself is not a statistical tool, the MF model developed in Chapter 3 does find a clear-cut classification boundary such that the division results in as low a misclassification rate as possible. The MSP model developed in Chapter 4 is a supplementary tool to find a range within which a sharp boundary is likely to lie. The present chapter extends the neural network model to provide decision makers with class membership information in the fuzzy set representation mode. The motivation for this work stems from the following considerations. First, statistical tools cannot be used to derive uncertainty information since we do not use strong assumptions about probability distributions which are often used in statistical pattern recognition. Secondly, considering the pattern recognition machine as a decision support tool, a classifier should supply more natural information regarding the class membership of a particular observation. Thirdly, the adaptive property of a neural network makes it possible to represent a managerial classification problem with fuzzy set information, as discussed in Section 5.3.

5.2. FUZZY SET CONCEPTS

This section will briefly introduce the basic concepts of fuzzy sets. The most recent and comprehensive literature review of fuzzy set theory is found in [Dubois and Prade 1989]. Zadeh's [Zadeh 1965] original idea of a fuzzy set is to consider a membership function $fz_c(\mathbf{X})$ which associates $\mathbf{X}$ (pattern vector of a point s) in the space Ω with a real number in the interval [0, 1] that represents the "grade of membership" of $\mathbf{X}$ in class c. For example, a person holding large amounts of assets is more likely to belong to a creditworthy class than someone holding less assets. A set can be given which represents creditworthiness as a membership function of the amount of assets (see Table 5.1).

Assets	Creditworthiness
0	0
100	0.1
200	0.3
300	0.5
400	0.7
500	0.9
1000	1.0

Table 5.1. A Fuzzy Relationship between Assets and Creditworthiness

It is also possible to show fuzzy relationships using a graph (see Figure 5.2(a)).

Two basic points regarding fuzzy set theory should be noted. (1) The grade of membership is subjective and context-dependent. There is not much point in treating the grade as a precise number [Zadeh 1984]. In many applications it is sufficient to represent the

grade of membership as a fuzzy number, say, approximately 0.8. A fuzzy set whose membership function takes fuzzy values is called ultrafuzzy (see Figure 5.2(b)).

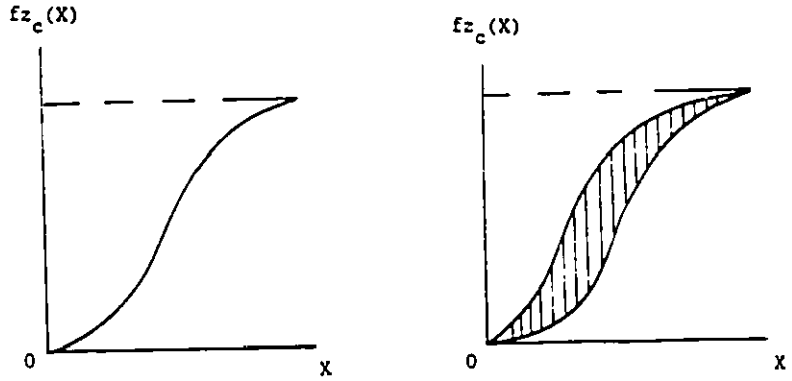


Fig. 5.2. Graphs Representing Fuzzy Relationships (adapted from [Zadeh 1984])

(2) The relationship between the grade of membership and probability is not explicit. For example, we may say that a person belongs to the creditworthy class with a membership grade of 0.8. 0.8 is not the probability with which the person is a member of the class, but is a vague representation of membership which is context-dependent. However, probability in the present discussion context is a definite measure. As discussed in Section 5.1, classical probability calculations are based on the entire population. In the example of Figure 5.1, it makes no sense to distinguish the probabilities of points A and B belonging to class 1. However, in a fuzzy set there exists a difference between A and B in terms of membership. Despite the significantly different concepts of probability and fuzzy sets, there is a certain relationship between the two, at least in the

theoretical sense. According to [Zadeh 1968, p422], the probability Pr(c) of class c is defined by

$$Pr(c) = \int_{\Omega} fz_c(X)\, dPr(X) \quad . \tag{5.1}$$

The interpretation of the above expression is that the probability of a fuzzy event $X \epsilon \Omega$ is the expectation of its membership function $fz_c(X)$.

Practically, however, the above equation can not be used to deduce the membership function directly. Sometimes, the membership function represents an individual's own idea of a vague category. In this case, a possible method to define the membership function would be similar to the approach used in defining subjective probabilities [Zadeh 1984, p6]. In other cases the membership function may be determined from statistical data [Dubois and Prade 1988, p19]. For instance, in order to determine the membership of X in Class 1, one may test X, say, 100 times to see how many times it is classified as Class 1. However, there is no commonly accepted practical method of determining the membership function. Nevertheless, fuzzy set theory emphasizes more the information structure (logical aspects) and the relation of the items of information to real events in dealing with imprecision and uncertainty (cf. [Dubois and Prade 1988])

Zadeh [Zadeh 1965] described how fuzzy sets can be manipulated by set operations. The classical set operations union and intersection can be extended by the following formulas [Zadeh 1965]:

For all $X \epsilon \Omega$,

$$fz_{c_1 \text{ OR } c_2}(X) = \max[fz_{c_1}(X),\ fz_{c_2}(X)] \tag{5.2}$$

$$fz_{c_1 \text{ AND } c_2}(X) = \min[fz_{c_1}(X),\ fz_{c_2}(X)] \tag{5.3}$$

A justification of the above logical operations was given by Bellman and Giertz [Bellman and Giertz 1973]. However, the complement $\bar{c}$ of c defined by the membership function [Zadeh 1965]

$$fz_{\bar{c}}(X) = 1 - fz_c(X) \tag{5.4}$$

is difficult to justify [Bellman and Giertz 1973] [Dubois and Prade 1980] [Negoita 1984]. The reason for rejection of the definition (5.4) is that the natural value of "not c" in terms of that of c may not arise from normal sharp mathematical intuition [Bellman and Giertz 1973]. A number of forms of natural conditions with related assumptions have been suggested for the complementation function (cf. [Dubois and Prade 1980]). A more general expression of complementation was developed by Sugeno [1974], and called λ-complementation [Sugeno 1977] [Dubois and Prade 1980, p127]:

$$fz_{\bar{c}}(X) = [1 - fz_c(X)] \ / \ [1 + \lambda\ fz_c(X)]\ ,\ \lambda > -1 \tag{5.5}$$

where λ is a parameter. Obviously, when $\lambda = 0$, the fuzzy λ-complementation is Zadeh's complementation definition. However, research (e.g. [Sugeno 1977, p98]) has shown that values of λ in the neighborhood of zero have not been observed in experiments. Figure 5.3 depicts a membership function and its complementation with various λ values.

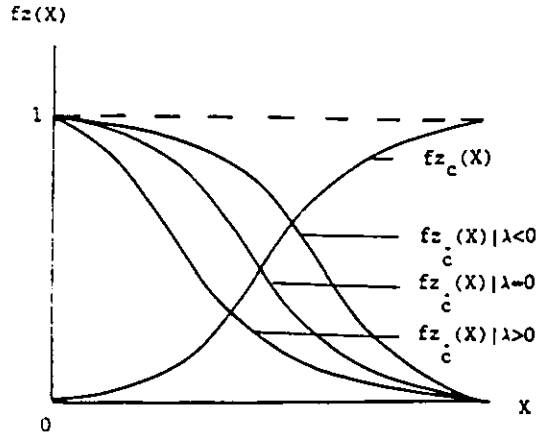


Fig. 5.3. λ-Complementation Relationship

The analysis of the complement of a fuzzy set is meaningful in the two class classification case. If we consider $c_1=\bar{c}_2$, then expression (5.5) becomes

$$fz_{c_2}(X) = [1 - fz_{c_1}(X)] \,/\, [1 + \lambda\, fz_{c_1}(X)] \;,\; \lambda>-1 \qquad (5.6)$$

Various theories are associated with the fuzzy complementation concept. For example, according to Shafer [Shafer 1976], suppose a belief function Bel(c) is a measure of a fuzzy set c, then

$$Bel(c) + Bel(\bar{c}) \leq 1 \qquad (5.7)$$

which means that a lack of belief in $X \epsilon c$ does not imply a strong belief in $X \epsilon \bar{c}$. On the other hand, according to Zadeh [Zadeh 1978], the possibility function Pos(c) is a measure of a fuzzy set c, where

$$Pos(c) + Pos(\bar{c}) \geq 1 \qquad (5.8)$$

meaning that "c is possible" does not necessarily imply "$\bar{c}$ is impossible". All of these theories could be expressed in the form of equation (5.5), provided that a proper λ range could be defined. However, no unique natural complementation concept has yet been commonly accepted, nor has a definite range of λ been specified. The more practically meaningful utilization of fuzzy complementation probably should not be separated from the specific problem to be solved. The remainder of this chapter uses the fuzzy complementation concept to examine the problem discussed in section 5.1.

5.3. THE FUZZY SET MODEL

5.3.1. λ-COMPLEMENTATION IN TWO CLASS CLASSIFICATION

Before building a neural network model to solve our problem, fuzzy λ-complementation is studied in more detail in the two class classification situation.

Suppose we have a fuzzy function $fz_{c_1}(\mathbf{X})$ which represents the membership of **X** in Class 1, where **X** is a point with m-dimensional attributes. For convenience, the function is depicted in two dimensions with **X** and $fz_{c_1}(\mathbf{X})$ coordinates as shown in Figure 5.4. Note that, when the dimensionality of **X** is larger than 1, the fuzzy membership function (e.g. $fz_{c_1}(\mathbf{X})$) is a surface, or hypersurface.

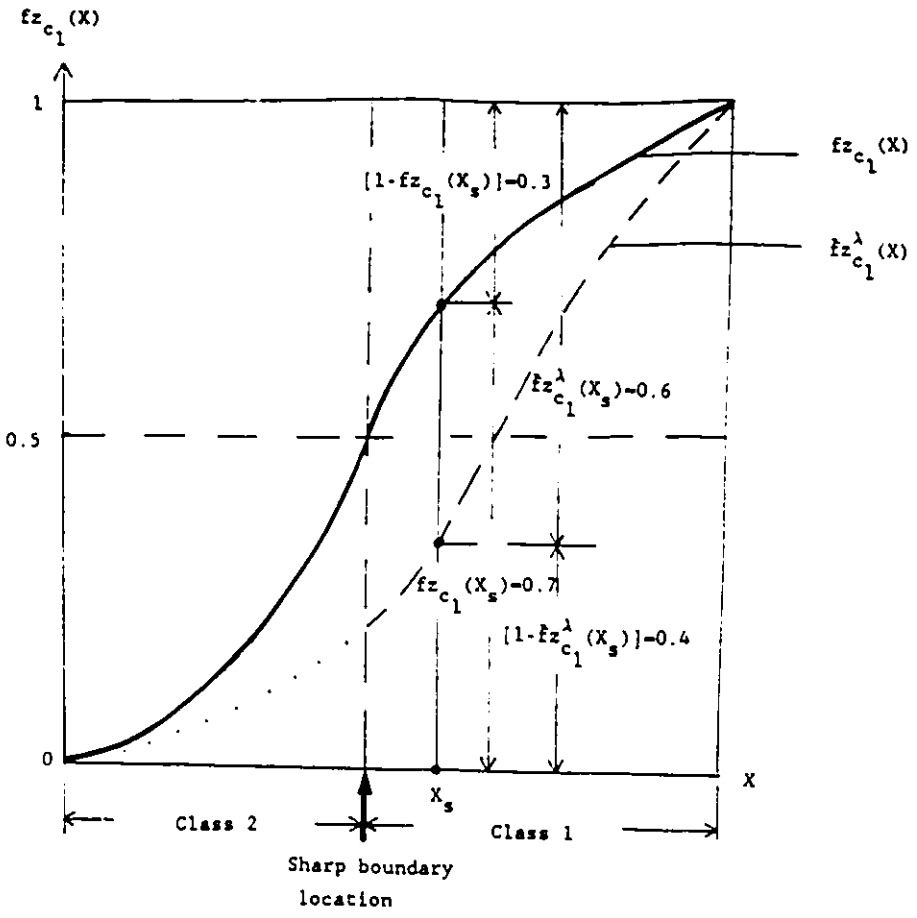


Fig. 5.4. Fuzzy Membership Functions in the Two Class Classification Case

To accommodate the fuzzy membership function to a sharp classification boundary, let

$X \epsilon c_1$ when $fz_{c_1}(X) \geq 0.5$,

$X \epsilon c_2$ otherwise; where $c_2 = \bar{c}_1$.

The values of X which satisfy $fz_{c_1}(X)=0.5$ define the sharp classification boundary. In fuzzy set terms, these points are called crossover points [Dubois and Prade 1980, p10]. As discussed earlier, the main objective of fuzzy set representation is not to pursue an exact fuzzy function, but to investigate its logical information and uncertainty aspects. In the light of this, a single fuzzy membership curve has little implication about uncertainty in the classification case, because we may arbitrarily define the function values provided that the location of the sharp boundary is fixed. Suppose that we have produced a fuzzy boundary membership function through some method such as the boundary function of LDA in two class classification. This fuzzy function value emphasizes more the degree of how far a point in the space deviates from the sharp boundary (see Section 5.1, which describes classification scores). However, it does not supply information about overlap of the data sets in addition to misclassification. In order to provide information about uncertainty based both on factors of potential misclassification (i.e. distance to the sharp boundary) and existing misclassification (i.e. overlap degree of the sample data), the single fuzzy membership function must be supplemented. In other words, we have to have two fuzzy membership functions in order to supply more complete information for classification of a new observation. One of these functions evaluates how far the new observation is away from the sharp boundary, and emphasizes more its "possible" membership. Another fuzzy membership function specifies the overlap degree of the two data sets, revealing more information about "belief".

The following develops a model for this purpose, and illustrates it with an example.

For a new sample point s with pattern vector X_s , based on the fuzzy function $fz_{c_1}(X)$ as shown in Figure 5.4, we might say that

Based on information about a given sharp boundary,

s belongs to Class 1 with membership 0.7 , and
belongs to Class 2 with membership 0.3 ;

if we have no information about existing misclassifications. On the other hand, suppose that we have a known misclassified sample point s' with the same pattern vector as point s; namely, $X_{s'}=X_s$. The complementation relationship would need to be modified. Obviously, the membership value of s' belonging to Class 2 would need to be adjusted so that $fz_{c_2}(X_s) > 1 - fx_{c_1}(X_s)$. According to λ-complementation theory (see equation (5.6)), λ should be in the range $-1<\lambda<0$ in this case. For example, after modification, the description of the membership values of s might become :

Based on both the sharp boundary and
the existing misclassification s',
a new observation with pattern vector X_s is

possible to be in Class 1 with membership 0.7,
and possible to be in Class 2 with membership 0.6;

OR, in other words, the new observation is

believed to be in Class 1 with membership 0.4,
and believed to be in Class 2 with membership 0.3.

In the above statement, we have the relationship between "possible" and "belief" as

$$Bel(c_i) = 1 - Pos(c_j), \quad i\neq j;\ i,j=1,2. \tag{5.9}$$

This theoretical relationship is discussed in [Dubois and Prade 1986].

Based on the existing misclassifications, we can supplement a fuzzy membership function with another fuzzy membership function, denoted $\tilde{fz}^{\lambda}_{c_1}(X)$ which is a λ-complementation of $fz_{c_1}(X)$. This provides more information about the uncertainty caused by misclassification. The $\tilde{fz}^{\lambda}_{c_1}(X)$ function is depicted in Figure 5.4 by the dotted line. Note that, conceptually, both $fz_{c_1}(X)$ and $\tilde{fz}^{\lambda}_{c_1}(X)$ are fuzzy membership functions on [0,1]. However, in our classification problem, the relationship between $fz_{c_1}(X)$ and $\tilde{fz}^{\lambda}_{c_1}(X)$ is hard to interpret intuitively when the active range is on the left side of the crossover point (Figure 5.4). The more meaningful discussion in that active range should be based on the corresponding X and $fz_{c_2}(X)$ coordinate (see Figure 5.5).

In order to determine the complementary relationship of two class membership, it is necessary to determine λ. Generally, the value λ differs from person to person depending on the individual's subjectivity [Sugeno 1977, p98]. In Section 5.4 we will show that the procedure of building the fuzzy set model is usually iterative. The initial selection of λ may be based on the ratio of the numbers of misclassifications to correct classifications. We may define extreme cases where:

$\lambda = -1$ when the number of misclassifications and correct classifications are equal; (In this case, the two data sets totally overlap, and $fz_{\bar{c}}(X)=1$ which means that we never believe an observation belongs to either class);

$\lambda = 0$ when no misclassification is observed, (In this case, $fz_{\bar{c}}(X)=1-fz_{c}(X)$ which means that we should accept the sharp boundary);

so that λ is normally in the range $-1<\lambda<0$.

The issue of determining practical values of the fuzzy membership functions will be discussed in section 5.4.

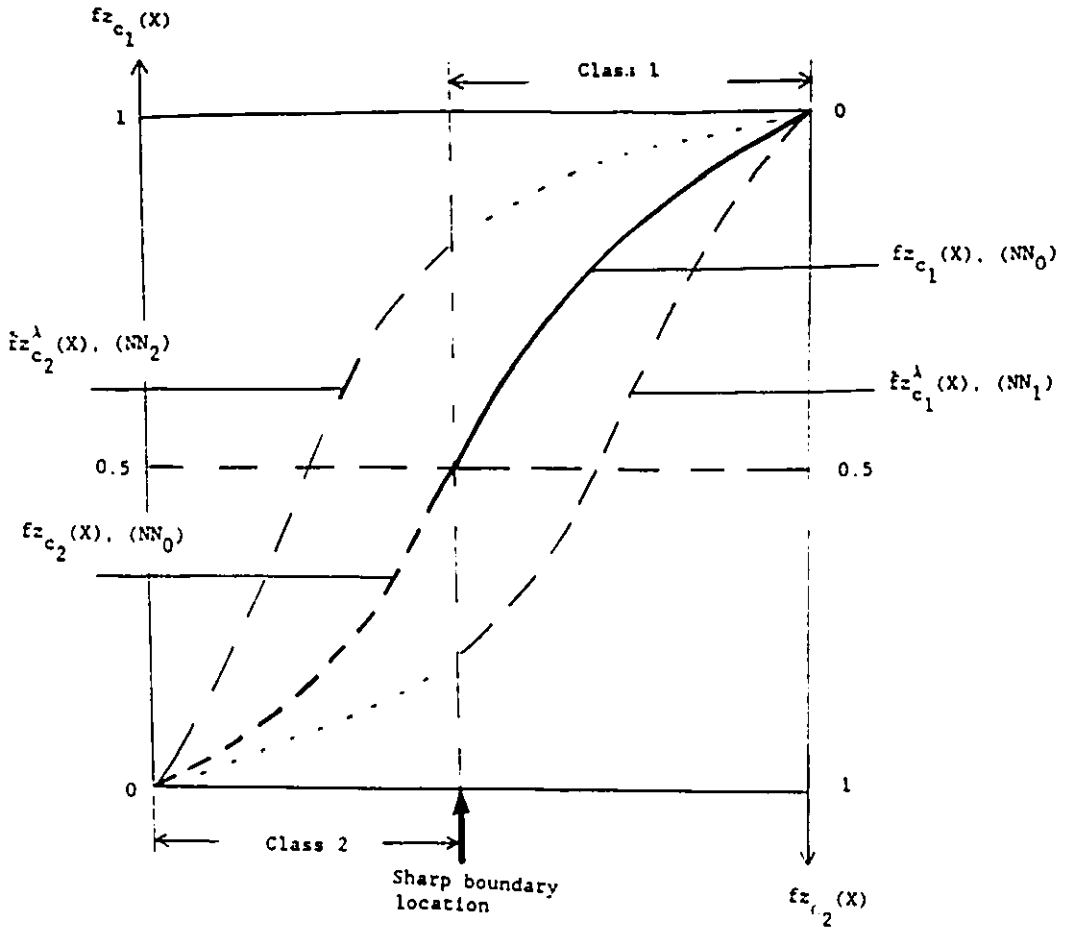


Fig. 5.5. Fuzzy Membership Functions Implemented in Conjunction with Neural Network Classification

5.3.2. FUZZY REPRESENTATION IN THE TYPICAL NEURAL NETWORK

The typical neural network employed in managerial pattern recognition generates a boundary separation of two classes (Figure 3.2), and has three characteristics which are relevant to fuzzy membership functions:

(1) If the neural network learning process is completed under the monotonicity constraint, the y surface must be monotonic.

(2) y is a continuous function in the pattern space (see equation (3.1)).

(3) y values range over the open interval (0, 1). If we define two constants

$$y_{max} = y\ [\ X \mid X = (1, 1, \ldots 1)\ ] \qquad \text{(e.g. 0.8)}$$

and

$$y_{min} = y\ [\ X \mid X = (0, 0, \ldots 0)\ ] \qquad \text{(e.g. 0.2)},$$

then the function y can be implemented through the learning process such that

$$0 < y_{min} \leq y \leq y_{max} < 1 \quad .$$

Note that, having these characteristics, the y function implemented by this kind of neural network is a fuzzy membership function, but subnormal, since the extreme values y_{min} and y_{max} are not 0 and 1 respectively [Bellman and Zadeh 1970] [Dubois and Prade 1980, p10]. However, the normalization can be implemented simply by a linear transformation (cf. [Bellman and Zadeh 1970])

$$y' = (\ y - y_{min}) \ / \ (y_{max} - y_{min}) \qquad (5.10)$$

so that $y' \in [0, 1]$. This normalization would ensure that all fuzzy set expressions (e.g. equation (5.6)) are applicable.

5.3.3. THE FUZZY MEMBERSHIP MODEL

According to the analysis in Section 5.3.1, two fuzzy membership functions (i.e. $fz_c(X)$ and $fz_{\bar{c}}(X)$) provide better

uncertainty information than a single fuzzy membership function in the two class classification case, since factors of both potential misclassification and existing misclassification are taken into account. Also, as pointed out in Section 5.3.2, a properly designed neural network classification algorithm can also incorporate a fuzzy membership function. Hence, a neural network model would be able to provide more complete information about uncertainty in terms of fuzzy membership functions.

A suggested neural network model, called the Fuzzy Membership (FM) Model, consists of three individual neural networks, each of which corresponds to one of the fuzzy membership functions shown in Figure 5.5. A suggested algorithm to implement the Fuzzy Membership (FM) Model is defined as follows.

Step 1. Based on the given sample, train the neural network using the MF model to find a sharp classification boundary with the neural network NN_0. (Details are provided in Chapter 3).

Step 2. Find misclassification sets $S_{mis_{c_1}}$ and $S_{mis_{c_2}}$ such that: $s \epsilon S_{mis_{c_1}}$ if misclassified point s is in the c_1 region, and $s \epsilon S_{mis_{c_2}}$ if misclassified point s is in the c_2 region.

If $S_{mis_{c_1}}$ or $S_{mis_{c_2}}$ is empty, this means that

there is no information available regarding the fuzzy nature of the given problem, and accept the sharp boundary; else go to the next step to develop the fuzzy boundary.

Step 3. Determine λ subjectively, based on the ratio of the numbers of misclassifications to correct classifications, so that $-1<\lambda<0$.

Step 4. For each $s \epsilon S_{mis_{c_1}}$ or $S_{mis_{c_2}}$ calculate $y(X_s)$ using the neural network NN_0.

Step 5. Normalize the membership value for these misclassified points

$$y'(X_s) = [\, y(X_s) - y_{min}] \,/\, (y_{max} - y_{min}),$$

where y_{max} and y_{min} are the extreme output values of NN_0 .

Step 6. For each misclassified point s assign

$$y'_{c_1}(X_s) = y'(X_s) \qquad \text{if } s \epsilon S_{mis_{c_1}},$$

$$y'_{c_2}(X_s) = 1 - y'(X_s) \quad \text{if } s \epsilon S_{mis_{c_2}}.$$

Step 7. Calculate λ-complementation values for the misclassified points

$$\bar{y}^{\lambda'}_{c_1}(X_s) = 1 - [1 - y'_{c_1}(X_s)] \,/\, [1 + \lambda\, y'_{c_1}(X_s)]$$

$$\bar{y}^{\lambda'}_{c_2}(X_s) = [1 - y'_{c_2}(X_s)] \,/\, [1 + \lambda\, y'_{c_2}(X_s)]$$

Step 8. De-normalize $\bar{y}^{\lambda'}_{c_1}(X_s)$ and $\bar{y}^{\lambda'}_{c_2}(X_s)$ for neural

network learning purposes such that

$$\bar{y}^{\lambda}(X_s) = \bar{y}^{\lambda'}(X_s)\ [y_{max} - y_{min}] + y_{min}\ .$$

Step 9. Train the neural network NN_1 (under the MF model and, usually, with the same topology as NN_0 and the same extreme output values y_{max} and y_{min}) with the sample set $S_{mis_{c_1}}$ such that each sample point has the λ-complementation value $\bar{y}^{\lambda}_{c_1}(X_s)$.

Step 10. Repeat Step 9 for neural network NN_2 trained with $S_{mis_{c_2}}$.

The neural network model consisting of NN_0, NN_1, and NN_2 will provide more information about fuzzy uncertainty in managerial pattern recognition.

5.3.4. AN EXAMPLE APPLICATION OF THE FUZZY MEMBERSHIP MODEL

Figure 5.6 shows an experimental result using the data described in Section 3.7.1 (compare Figure 5.6 with Figure 3.11). Based on the sharp boundary generated by NN_0 and the misclassification information obtained using the MF model (Chapter 3), the two fuzzy membership functions $\tilde{f}z^{\lambda}_{c_1}(X)$ and $\tilde{f}z^{\lambda}_{c_2}(X)$ were generated by two neural networks NN_1 and NN_2, respectively. The fuzzy membership functions represented by the y-surfaces of the

neural networks are three dimensional in this case, because the pattern vector **X** has two dimensions. We use contour lines to represent the two complementation fuzzy membership functions in the diagram (compare Figure 5.6 with Figure 5.5). A value of λ was subjectively selected as -0.3 for this application; although the λ range is (-1,0), and the misclassification ratio 4/36 is not very high. Figure 5.6 shows three groups of lines. The first one is the sharp boundary (shown to be close to the true boundary) representing the neural network NN_0. Any new observation in the pattern space will be assigned with a classification score by the neural network NN_0. This classification score indicates the <u>possibility</u> of the new observation belonging to one of the two classes. For example, according to NN_0, point A(0.8,0.3) is classified as class 1 with the classification score 0.55, which means that A <u>possibly</u> belongs to class 1 with grade 0.55. On the other side of the sharp boundary, point C(0.5,0.3) is assigned by NN_0 with classification score 0.45, which only means that C <u>possibly</u> belongs to class 0 with grade 0.55. (Note that we do not discuss the issue of C belonging to class 1 here; see Section 5.3.1). The second group of lines are contours of $\hat{f}z_{c_1}^{\lambda}(\mathbf{X})$ generated by NN_1. They represent the <u>belief</u> of a new observation belonging to class 1, given the new observation is classified as class 1 according to the sharp boundary. For example, according to the complementation fuzzy membership function as shown in the contours of $\hat{f}z_{c_1}^{\lambda}(\mathbf{X})$, A is <u>believed</u> to belong to class 1 with

grade 0.4, which indicates the uncertainty of A belonging to class 1 caused by the influence of misclassifications. On the other hand, B(0.85,0.4) is classified as class 1 with more certainty. The neural network NN_0 represented by the sharp boundary assigns B with 0.65 possibility grade, and the contours of $\tilde{f}z_{c_1}^{\lambda}(X)$ representing NN_1 assigns B with 0.55 belief. Both are greater than 0.5. The third group of lines are contours of $\tilde{f}z_{c_2}^{\lambda}(X)$ generated by NN_2. They represent the belief of a new observation belonging to class 0, given the new observation is classified as class 0 according to the sharp boundary. For example, though C possibly belongs to class 0 with grade 0.55, it is believed to belong to class 0 with grade 0.5 as shown in the contours of $\tilde{f}z_{c_2}^{\lambda}(X)$, which means that C is in a critical status due to the influence of misclassifications. From Figure 5.6 one may see that, in the unshaded regions, the membership of a new observation will be more certain than observations in the shaded region which might be considered as a "fuzzy boundary" for a particular λ value. Within the shaded region, the belief functions for both class 1 and class 0 are less than 0.5.

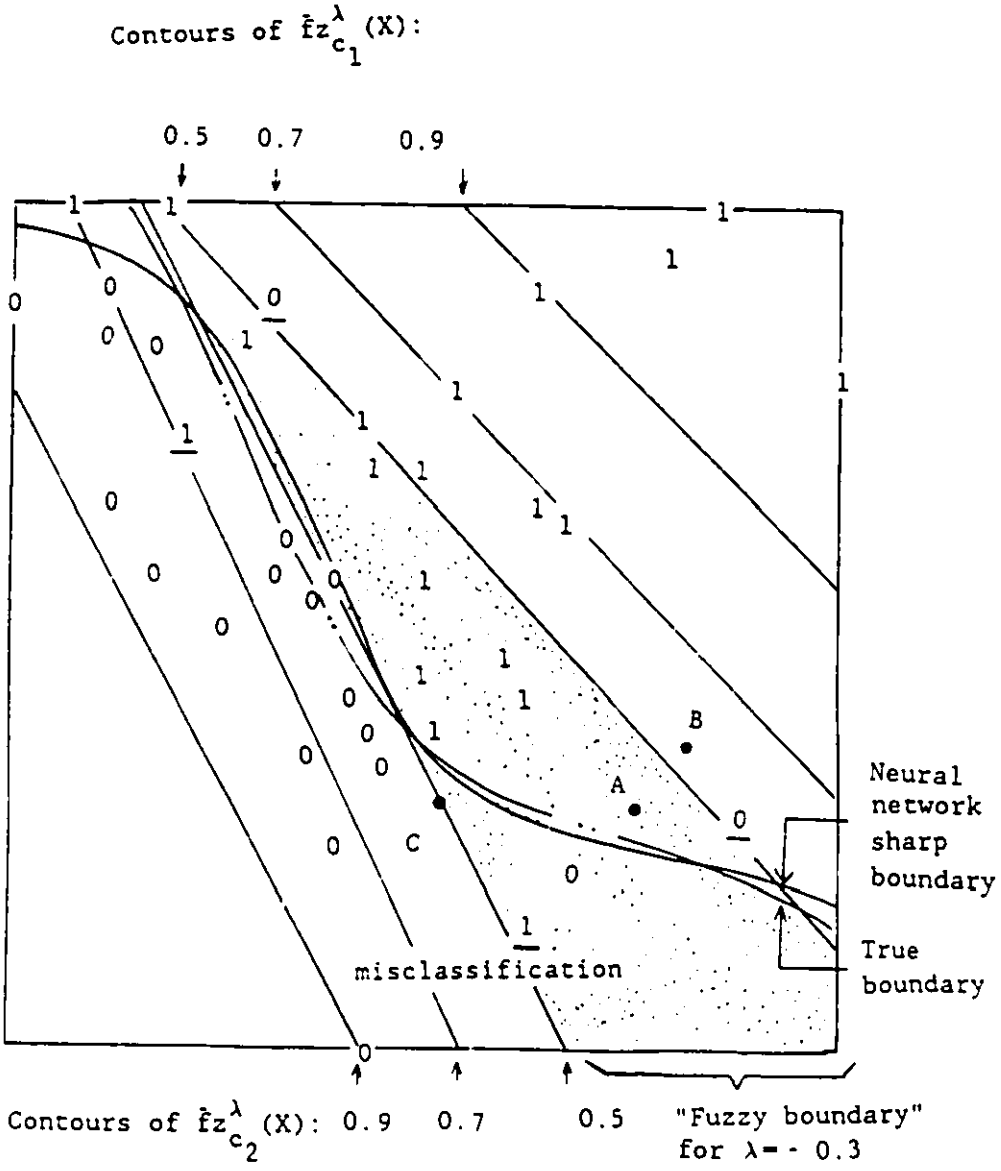


Fig. 5.6. Fuzzy Membership Functions for the Example in Section 3.7.1 (compare with Fig.3.11)

5.4. DISCUSSION

5.4.1. PRACTICAL FUNCTION VALUES

An important question from fuzzy set theory is how to actually derive membership functions. Answering this question is critical for practical applications, and the following three points address the question.

(1) Fuzzy sets are relatively "subjective". Theoretically, no fuzzy set can be proved to be true, in the absence of the user's opinion (see [Zadeh 1965, 1984]). Therefore, any fuzzy set decision model which does not incorporate a particular user's opinion is at most a prototype.

(2) The fuzzy membership functions represented by a particular neural network model prototype may have differing values; however, in the suggested model a certain value of the fuzzy membership (0.5± in the Figure 5.5 example) is always established at the assumed sharp boundary. This ensures that no mistakes occur in the "yes/no" sense. Thus, the suggested model is at least a good prototype.

(3) There are several methods which can be used to find a "practical" fuzzy membership function (note: not the "true" fuzzy membership function) (cf. [Dubois and Prade 1980, p257] [Bremermann 1976, p116]). In general, one may consider an interactive procedure to construct a practical fuzzy membership function for a given problem, which is briefly described as follows. First, develop a prototype for the fuzzy membership functions, based on the given data. Then generate a series of (artificial) typical sample data.

Input these sample data into the prototype MF model. Each input observation generates an output. The users (decision makers) observe these input/output pairs and their associated values for possibility and belief, which are based on the prototype fuzzy membership functions. They may accept the results, or modify λ to get a better subjective feeling for the classification. Only minor modifications are allowed to the fuzzy membership function shape, but not significant changes to the crossover points defined by the sharp boundary. The neural network model is then re-trained, taking advantage of its highly adaptive nature. At this stage, the aim is to minimize any discrepancies between the prototype fuzzy membership functions and the users' subjective thinking. The final neural network model then includes the subjective factors discovered in this way, but it would only be applicable to this set of decision makers and to this particular situation. It may also be time dependent, since decision makers' opinions may change over time. On the other hand, since the model was based on knowledge extending beyond the scope of the limited sample data, the final model would be a generalized knowledge representation that can be associated with this environment (cf. [Dubois and Prade 1980, p358]). The point is that the neural network model acts as an artificial intelligence tool to aid humans in obtaining and accumulating knowledge. It may aid the decision maker directly, or act as a component in a fuzzy reasoning machine - a sort of expert system. It should be no surprise that a good model of this kind would often be time consuming and expensive to construct; as well, there is no universally applicable managerial classifier which can be built from a limited sample data set.

5.4.2. THE RELATION BETWEEN MFM, MSPM, AND FMM

In this subsection we will discuss the relation between the three models developed in Chapter 3, 4, 5, respectively; namely, MFM (Monotonic Function Model), MSPM (Monotonically Separable Problem Model), and FMM (Fuzzy Membership Model).

Given a sample data set, we may judge if the problem is monotonically separable simply by using the dominance principle. If it is, then MSPM can be used to find the "unbiased" sharp boundary as described in Chapter 4. In this case, there would be nc way to apply FMM because no misclassification was observed. If the problem is not monotonically separable, then MFM should be employed to find a sharp boundary which can improve classification performance beyond the linear discriminant function's classification result. It is also possible to use MSPM to reduce the learning bias effect to obtain a more "unbiased" sharp boundary as shown in Chapter 4. In order to supply classification information on class overlap in a natural way, FMM could be employed at this point to develop a good prototype model reflecting fuzzy membership functions. These could then be refined through interactions with the users, as discussed in the previous section.

CHAPTER SIX

GENERALIZATION TO MORE THAN TWO CLASSES

A summary of the MF neural network model approach to managerial pattern recognition as defined in Chapter 3 was :

> In two class managerial pattern recognition problems, the sample data are first pre-processed by a linear classifier (i.e. linear discriminant analysis) to reduce significant statistical fluctuations. The pre-processed data are then analyzed by vector analysis in order to determine a proper training sample set for the neural network. Finally, the objective classifier for the problem is the BPLMS neural network algorithm, completing the learning process under monotonicity constraints.

This model has been entirely addressed to the k=2 (k is the number of classes) managerial classification problem. However, the basic idea of the model may be extended readily to k>2 problems. This chapter discusses related issues, as follows: Section 6.1 briefly reviews classification methods for k>2 problems. Section 6.2 discusses decision region complexity in k>2 classification, and a possible extension of the k=2 model. Section 6.3 illustrates k>2 neural networ' models.

6.1. CLASSIFIERS FOR k>2 CLASSIFICATION

The k>2 classification problem has not been as extensively studied as the k=2 problem for several reasons. First, as will be seen, the theoretical application of Bayes rule to obtain optimal solution is not difficult in the k>2 case; however, practical applications using Bayes rule require necessarily strong assumptions. Secondly, because of difficulty in obtaining an overall optimal solution for k>2 classes, the two class method is often employed as a starting point in k>2 classification, as discussed in more detail in Section 6.1.2. From this standpoint, the k>2 problem can be considered as a direct extension of the k=2 problem.

There basically are three categories of statistical methods in k>2 classification [Hand 1981], described in the following subsections.

6.1.1. BAYES RULE

The most fundamental statistical method is Bayes rule. For the k class problem,

$$Pr(c_i|X) = Pr(c_i)\, f_{c_i}(X) \Big/ \sum_{j=1}^{k} Pr(c_j)\, f_{c_j}(X) \quad , \quad i=1 \ldots k$$

are calculated. The decision rule is:

$$Pr(c_i|X) = \max_j Pr(c_j|X) \quad \text{--->} \quad X \epsilon c_i$$

In practical applications (e.g. [Chien and Killeen 1982]), estimates of all the probability distribution functions are needed,

along with assumptions about prior probabilities and conditional probability distributions.

6.1.2. FISHER'S APPROACH

Fisher's approach to k=2 problems (see Section 1.3.1.2) can be generalized to problems with k>2 classes, by finding the linear compound B which maximizes the ratio γ of between-group variance to within-group variance, where

$$\gamma = \frac{B' \Sigma_b B}{B' \Sigma_w B} \quad .$$

Here, Σ_b is the between-group covariance matrix, and Σ_w is the within-group covariance matrix. Fisher's method tacitly assumes that m≥k, where m is the pattern vector dimension [Duda and Hart 1973], and it cannot be used, for example, on the three class problem with a two dimensional pattern vector. The solution includes no more than min[k-1, m] discriminant functions. Suppose D discriminant functions are obtained by Fisher's method. The goal selection rule is: assign the observation to c_i if

$$\sum_{r=1}^{D} [B_r' * (X - \mu_i)]^2 = \min_j \sum_{r=1}^{D} [B_r' * (X - \mu_j)]^2$$

where μ_j (j=1..i..k) is the mean of the pattern vectors of class c_j. However, very little is known about the characteristics of this method, and it cannot be shown to be optimal [Lachenbruch 1975].

6.1.3. LINEAR FUNCTION CLASSIFIER

A practical method is simply to extend two class linear discriminant analysis heuristically so that the classification space is divided by a number of linear classifiers [Duda and Hart 1973] [Hand 1981]. There are three schemes for k>2 classification, described as follows:

(1) First separate c_1 from $c_2 \ldots c_k$, then separate c_2 from $c_3 \ldots c_k$, and so on, until c_{k-1} and c_k are separated (see Figure 6.1). The problem with this scheme is that the set with the small index is separated first, and therefore tends to be assigned a bigger region. This method needs (k-1) linear classifiers.

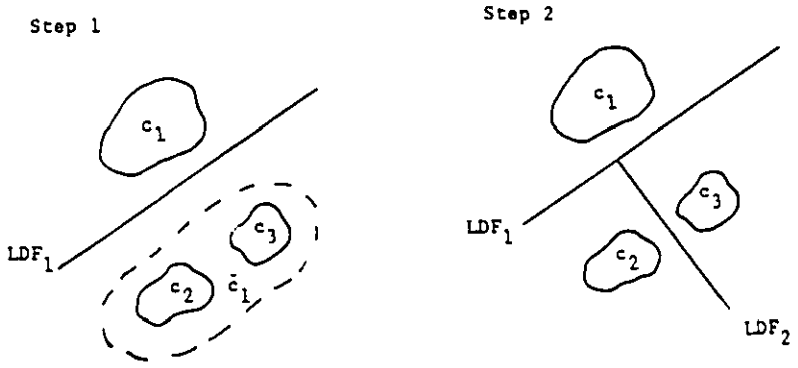


Fig. 6.1. Linear Functions in the k>2 Case (Scheme 1)

(2) Carry out an iterative process over all i, i=1...k-1 such that the c_i re separated from all c_j (j=1...i-1, i+1...k) (see Figure 6.2). This method also needs (k-1) linear functions. The problem is

that there are undecided regions which may not be assigned to a class (see the shaded region in Figure 6.2).

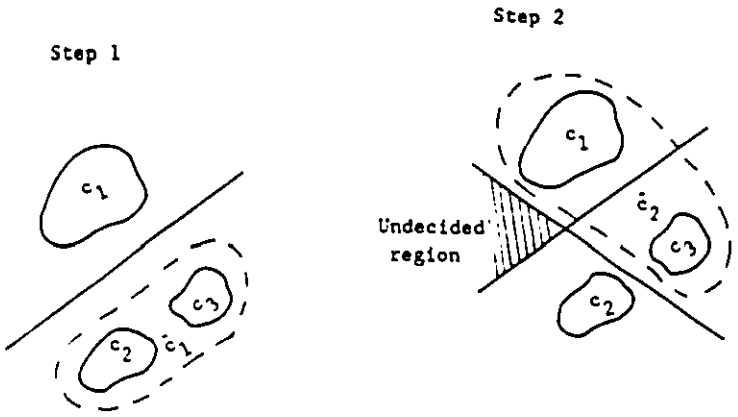


Fig. 6.2. Linear Functions in the k>2 Case (Scheme 2)

(3) Separate two of the k classes at a time, ignoring the other classes each time (see Figure 6.3). This requires a total of k(k-1)/2 linear classifiers. Apart from possibly excessive computational requirements, this method also leads to undecided classification regions.

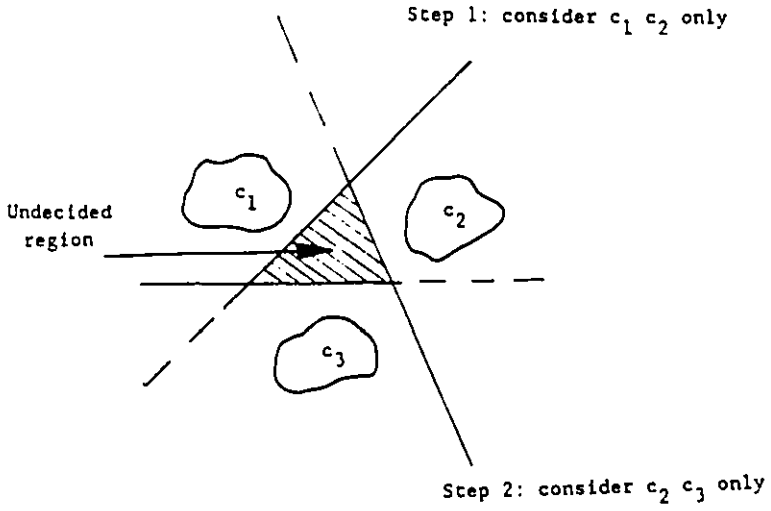


Fig. 6.3. Linear Functions in the k>2 Case (Scheme 3)

For our purposes, two important properties of these schemes should be noted:

(1) the optimality of none of these methods has been proved, if the optimal solution is to maximize the probability $Pr(c_j|X)$ for all classes c_j , j=1...k, from the Bayesian point of view (cf. [Lachenbruch 1975]). No one scheme is superior over the other two in all situations. There is also no sound theoretical basis for any of the extensions to the k>2 linear discriminant analysis methods, and

(2) the regions generated by the linear discriminant functions discussed above are convex [Hand 1981].

6.1.4. OTHER CLASSIFIERS

In real k>2 classification problems the decision region is not necessarily convex, and the boundaries are not necessarily linear. Besides Bayes rule, there are several methods available to solve this type of classification problem, with its highly irregular regions. One approach is to split complex decision regions into subclasses using a linear classifier [Hand 1981] (see Figure 6.4). Obviously, a priori knowledge about how to split the complex region must be available. Another more general approach used for disjoint region cases is a piecewise learning machine method [Nilsson 1965]. If this method is used, class prototypes are required (see Figure 6.5). The decision rule used is to minimize the Euclidean distance from each observation to a specified point in a given prototype (Figure 6.5).

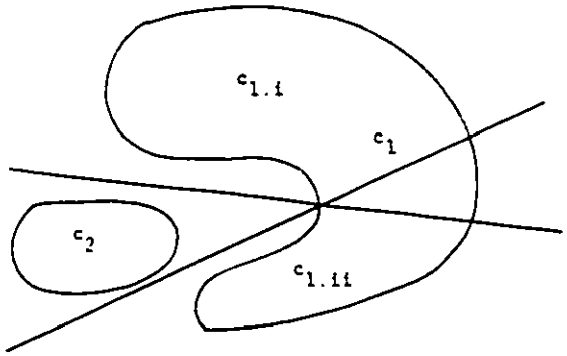


Fig. 6.4. Split Complex Decision Region into Subclasses (adapted from [Hand 1981])

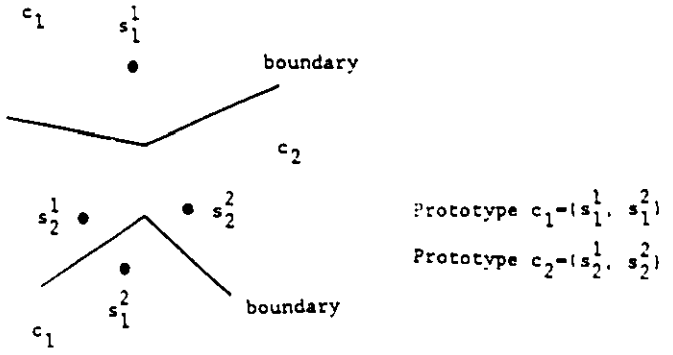


Fig. 6.5. Minimum Distance Classifier in Piecewise Learning Machine (adapted from [Nilsson 1965])

6.2. A POSSIBLE EXTENSION OF THE k=2 MF MODEL

Suppose that we have no other knowledge except for the statistical data set consisting of discrete sample points. The linear function classifier would seem to be the only feasible method to develop a starting "prototype" in k>2 classification. Given this approximate classification result the neural network model with its

highly adaptive nature could be used to improve on this result as in the k=2 case. The idea of extending the k=2 model to the k>2 case is then straightforward; that is, given a k>2 problem, use linear classifiers to obtain initial approximate classification boundaries, then employ neural network techniques to reduce misclassifications. In order to develop this extended model, decision region complexity must be examined, as follows:

Complexity level 1 The simplest form of decision region is the convex shape with linear boundary. As shown in Section 6.1, in principle this form of decision region may be generated by linear classifiers (see Figure 6.6(a)).

Complexity level 2 A more complex form of decision region is a convex region with a non-linear boundary between each pair of the k classes (see Figure 6.6(b)).

Complexity level 3 An even more complex form of decision region is a continuous non-convex region with monotonic boundaries between each pair of the k classes (see Figure 6.6(c)).

Complexity level 4 The most general and the most complex form of decision region is highly irregular; that is, each decision region may or may not be disjoint, and the decision boundaries have no specific convex or monotonicity properties (see Figure 6.6(d)(e)).

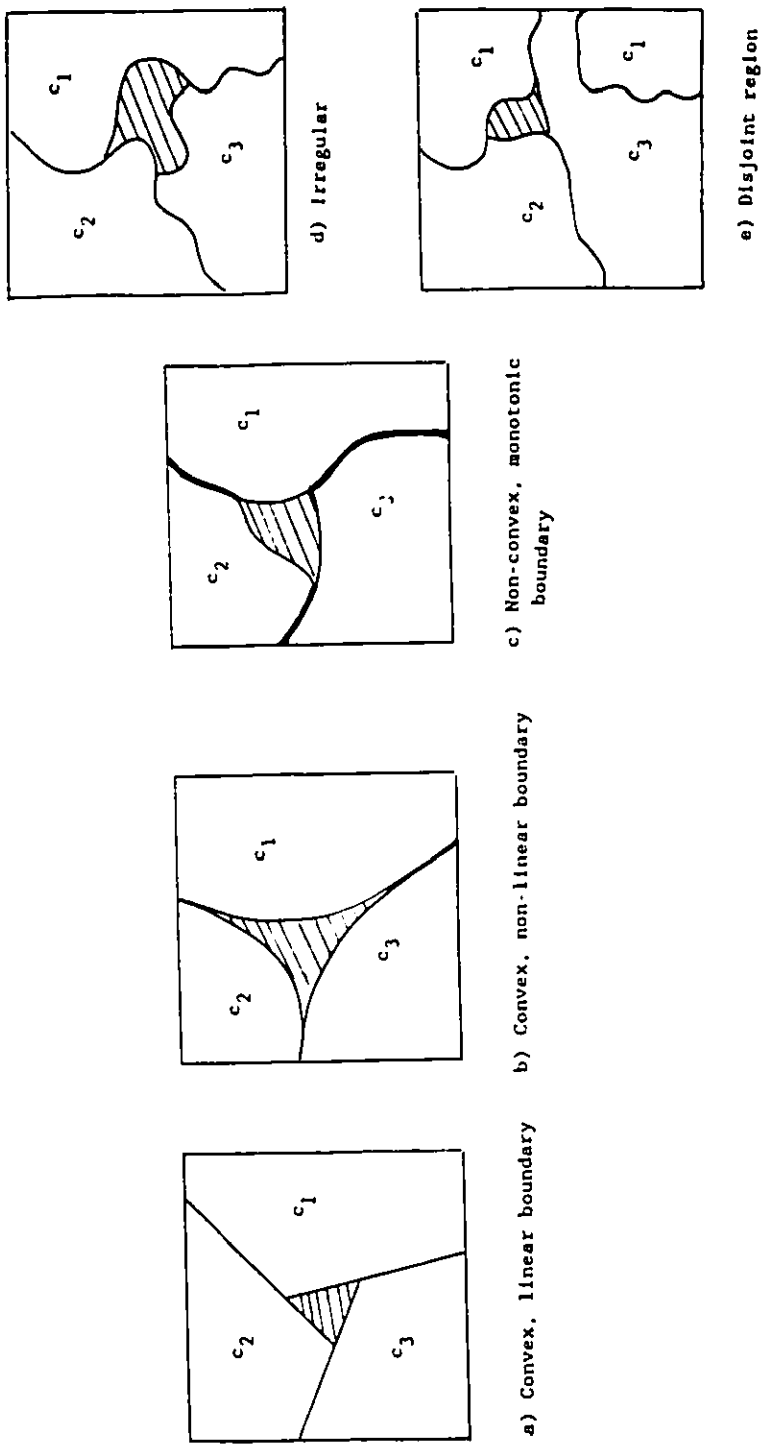


Fig. 6.6. Decision Region Complexity Spectrum

Establishing the level of monotonic boundary in the complexity spectrum described above is based on the following three considerations:

(1) Accuracy.

In the complexity spectrum shown in Figure 6.6, each preceding form is a special case of the succeeding one. For instance, a convex region with a linear boundary is a special case of a convex region with a non-linear boundary. On the other hand, there is a wide range of complexity between decision regions which are convex and those which are irregular. It is desirable to select the type of decision region which is an organized set of non-convex decision regions. This can be satisfied by establishing monotonic boundaries between each pair of the k classes.

(2) Consideration of neural network learning constraints.

It is natural to consider monotonic boundaries between pairs of decision regions, since this is a natural extension of the k=2 MF model.

(3) Practical considerations.

The most important consideration is application in practical situations. As has been discussed, k>2 classification problems are often treated as a series of k=2 classification problems (see Section 6.1). When two of the k classes (or one class vs. the remaining (k-1) classes) are independently considered, one may find some monotonic (either monotonic increasing or decreasing) relationship between the pattern vector **X** and the likely classification score y, based on

generic knowledge about the particular problem. For example, in an oil spill identification problem [Chien and Killeen 1982, p664], during chemical testing of a spill sample, a spectrum is run on the sample, and the distance between its particular spectral line and that of the suspect sample is calculated as a pattern variable of the sample. Typically, the distance increases monotonically as the suspect and spill spectra become dissimilar. Sometimes, the relationship between the initially defined x_j (j-th component of X) and y may not be strictly monotonic. However, if there is knowledge about the inflection point, one may decompose x_j into two or more dimensions in order to obtain a monotonic relationship. This is illustrated in Figure 6.7.

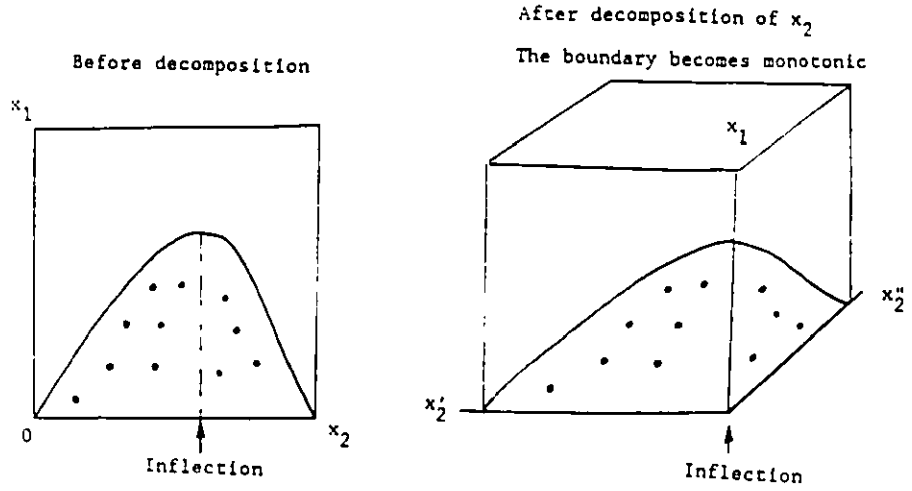


Fig. 6.7. Decomposition of A Pattern Dimension

Note that, if the slope of the boundary at the inflection point is not very large, the classification result is not likely to be very sensitive to the accuracy of location of the inflection point (Figure

6.7); in other words, knowledge about the inflection point does not have to be very precise. The important point here is that information about an inflection point, even if imprecise, still makes it possible to apply linear functions efficiently to obtain an approximate classification result. As well, it will provide constraints on neural network learning behavior, which can be used to further improve classification results.

Based on the above considerations, the extension of the k=2 MF model becomes straightforward.

6.3. THE GENERALIZED MF MODEL

6.3.1. TYPICAL NEURAL NETWORK TOPOLOGIES FOR k>2 CLASSIFICATION

As discussed in Section 6.1, there are two possible numbers of linear discriminant functions which could be required for a given k>2 classification problem : (k-1) or k(k-1)/2. Accordingly, two types of neural network structures can be developed for k>2 classification; those with (k-1) output nodes and those with k(k-1)/2 output nodes, respectively (see Figure 6.8).

Note that each output node corresponds to one linear function in interpretation of the meaning of that output. For example, suppose that we adopt the classification scheme shown in Figure 6.1 for a three class problem, and use two output nodes, y_1 and y_2 , in the neural network model. Then, the output of y_1 corresponds to LDF_1 and y_2 corresponds to LDF_2. The output

classification rules would be: iff $y_1>0.5$, classify the observation as c_1 ; iff $y_1<0.5$ and $y_2>0.5$, classify it as c_2 ; iff $y_1<0.5$ and $y_2<0.5$, classify it as c_3 .

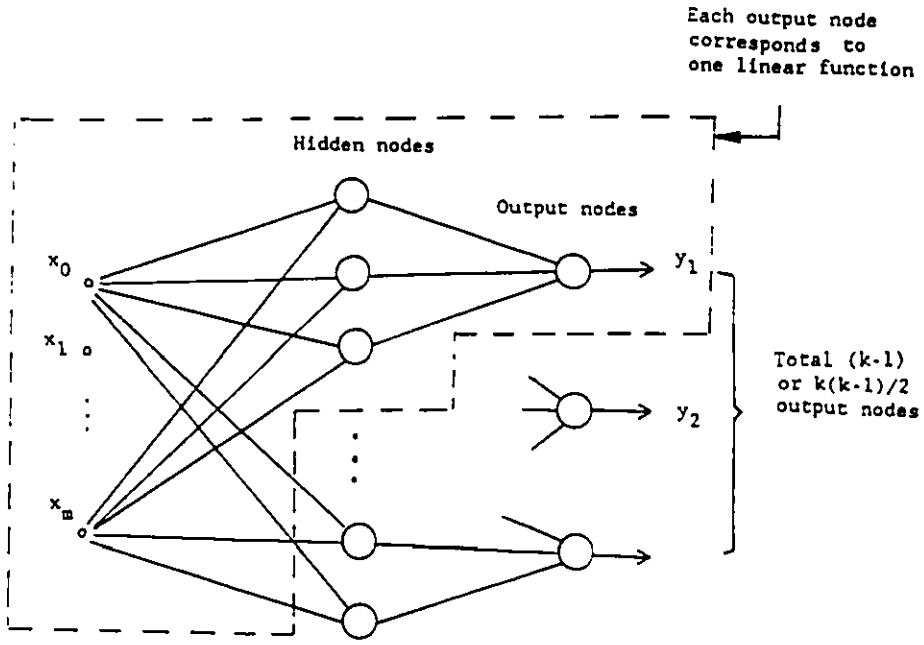


Fig. 6.8. Two Types of Typical Topologies of Neural Networks

6.3.2. A GENERAL ALGORITHM FOR NEURAL NETWORKS IN k>2 CLASSIFICATION

As discussed above, if knowledge about monotonic conditions is available, it is possible to extend the learning model for k=2 classification to the k>2 case. The Generalized Monotonic Function Model is briefly stated as follows.

Step 1. Select the separability criterion, i.e. (k-1) or k(k-1)/2 for the number of linear classifiers to be used.

Step 2. Decompose any non-monotonic variables which exist, based on knowledge about monotonic conditions.

Step 3. Pre-process the sample data set using the LDA approximation.

Step 4. Use vector analysis to find any clusters so that the modification of the pre-processed linear boundary to include these clusters would result in a lower misclassification rate.

Step 5. Build a neural network (as shown in Figure 6.8). Select the number of output nodes and hidden nodes based on the separability criterion and the estimated minimum number of hidden nodes (see Section 4.7) respectively.

Step 6. Ignore any sample points which carry significant statistical fluctuations according to the LDA pre-process and the vector analysis result. Train the neural network with the MF model, using the modified training data set. The final classification result will have a misclassification rate which is no greater than that of pure linear discriminant functions.

Because this model is a direct extension of linear discriminant analysis, there will usually be undecided regions as in the linear discriminant analysis cases (Figures 6.1-6.3). When an observation appears within an undecided region, the class selector will be unable to decide its class according to the pre-defined class selection rule (see Section 6.3.1 explanation). If this happens, a possible conclusion is that the new observation to be classified does not necessarily belong to any known class. For example, in oil spill identification [Chien and Killeen 1982], a suspect sample falling into an undecided region probably belongs to a class which is not included in the known class set. However, if the class set is supposed to be complete, a possible resolution is to classify a new observation in the nearest class based on its neural network classification score in this situation (see Figure 6.9). As discussed earlier, in the current model shown in Figure 6.8, each output node corresponds to one boundary function. Its geometrical interpretation can be seen by reference to Figure 3.1; that is, each part of the neural network in Figure 6.8 corresponds to a y-surface. Therefore, an n output neural network actually implements n y-surfaces. In the usual case, there may be undecided regions between these y-surfaces.

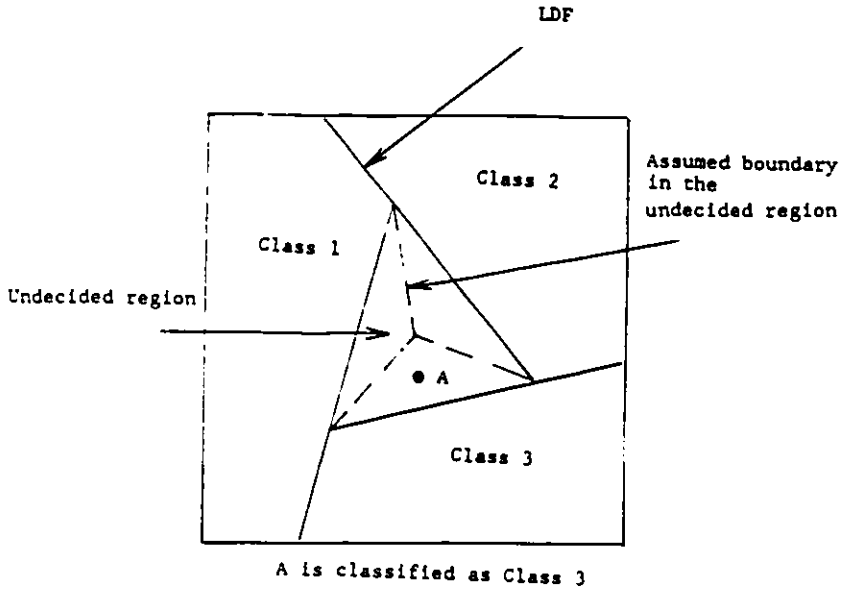


Fig.6.9. Classifying a New Observation Falling in an Undecided Region

6.4. AN EXAMPLE OF k>2 CLASSIFICATION

Fisher's iris data [Fisher 1936] have been widely used in the literature for studying pattern recognition (e.g. [Sammon 1969] [Chien 1978] [James 1985]). The data which are given in Appendix V consist of four variables on 50 sample points in each of three classes of Iris: Setosa, Versicolor, and Virginica. The four pattern variables are sepal length and width, and petal length and width. Using the algorithm proposed in Section 6.3, Fisher's data was first pre-processed by linear function classifiers as in classification scheme 1 (Figure 6.1); that is, class 1 (Group I in the real data) is first separated from non-class 1 (Group II and Group III), and class 2 (Group II) is then separated from class 3 (Group III). Three

points were misclassified, resulting in 98% accuracy. It was assumed that each species differs from others on these variables in a monotonic manner. The Monotonic Function Model for the k>2 problem was then applied. In this case m=4, and the sample size of each class was 50, which was considered as a reasonable upper limit for ν^* in equation (4.11). Accordingly, a neural network with nine hidden nodes was employed corresponding to each linear function classifier, and there were two output nodes. One point (ID number 134 in Group III, see Appendix data), that was misclassified by the linear function classifiers, was ignored in the neural network learning process. As a result, 149 of 150 sample points were classified correctly, resulting in 99.3% accuracy. The final neural network weights are shown in Appendix IV. The problem solution cannot be displayed on a two dimensional plot because the input data have four dimensions.

6.5. DISCUSSION

In this chapter the generalized MF model has been described. The extension of the Monotonically Separable Problem model (Chapter 4) to k>2 classification is straightforward. If the extension of the MSP model is applied to a k>2 problem, two frontiers generated by a neural network with two output nodes (Figure 4.6) corresponds to the boundary generated by the MF model for each pair of class sets (either c_i and c_j , $i \neq j$, or c_i and $\bar{c}_i$, depending upon the separability criterion). The two frontiers generated will reduce the

effect of learning bias in neural networks and will supply assurance information. Usually, one may take the "unbiased" boundary (see formula (4.8), Chapter 4) as the final classification result.

The extension of the Fuzzy Membership model (Chapter 5) to k>2 classification is not difficult to develop. However, the practically meaningful interpretation of the fuzzy membership in the k>2 cases is questionable. For instance, suppose the separability criterion is adopted with scheme 3 (Figure 6.3), namely, each sharp boundary separates c_i and c_j ($i \neq j$). Corresponding to a sharp boundary, there would be a pair of fuzzy membership functions $\tilde{f}z^{\lambda}_{c_i}(X)$ and $\tilde{f}z^{\lambda}_{c_j}(X)$ to supply fuzzy uncertainty information with respect to c_i and c_j only. In this model, if extended to the k>2 problem, no generalized fuzzy uncertainty information would be available on the membership of an observation in all of the classes. Similarly, if the separability criterion is adopted with scheme 1 (Figure 6.1), or scheme 2 (Figure 6.2), the fuzzy uncertainty information mainly concerns the membership of an observation in c_i and $\bar{c}_i$ (where $\bar{c}_i=(c_{i+1}\cdots c_k)$ for scheme 1, and $\bar{c}_i=(c_1\cdots c_{i-1},c_{i+1}\cdots c_k)$ for scheme 2), rather than its membership in an individual class. This research suggests that the lack of available theory about fuzzy membership for k>2 classification, rather than the lack of powerful implementation tools, is the main obstacle to this generalization.

CHAPTER SEVEN

CONCLUSIONS AND DISCUSSION

7.1. GENERAL CONCLUSIONS

Regarding the information sufficiency problem in neural network classification, Denker and Wittner [Denker and Wittner 1987] pessimistically pointed out:

> Consider the following objective : first, the network should be very powerful and versatile, i.e., it should implement any function (truth table) you like, and secondly, it should learn easily, forming meaningful generalizations from a small number of training examples. Well, it is information-theoretically impossible to create such a network.

This is very true for the standard BPLMS neural network in classification applications. Even more pessimistically, with standard neural networks it is impossible to control boundary functions unless some assumptions are made about their shape, and it is difficult to develop meaningful generalizations.

Neural network researchers are painfully aware of these disadvantages, and have been trying to improve available algorithms. There have been two important published works on neural networks

which deal with statistical data in classification. One deals with the learning vector quantization model [Kohonen et al. 1988]. This approach is based on the nearest-neighbor method. In this method the number of processing units in the input domain is predetermined. Each unit has a predetermined d-element reference vector, and each unit is associated with one of the classes of the input samples. The learning process is used to update the unit. Another model dealing with statistical classification data is the probabilistic neural network [Specht 1990a, 1990b]. The probabilistic neural network is actually a parallel processing device for statistical functions. The hidden nodes (pattern units in the probabilistic neural network model) assume a kind of kernel function with a pre-defined smoothing parameter. The learning process sets the weight vector directly and connects the neural network nodes properly. Both of the above models apply statistical concepts directly to neural networks. As discussed above, the major limitation of these two methods in the context of our discussion is that strong assumptions are necessary about certain parameters, and the selection of these parameters influences the results for a particular problem. Therefore, the validation of these models is basically empirical, based upon benchmarking studies.

The present research, however, tries to fully exploit the adaptive nature of neural networks, and utilizes a type of generic problem domain knowledge, (i.e. monotonicity), to improve neural network learning ability in generating a proper classification boundary. In the models developed in the present research no assumptions are necessary about statistical properties of the sample data set. Instead, more general knowledge about classification is

considered and is consolidated in the neural network through the learning process. The argument is that the information (or prior knowledge) required in the present neural network model is more readily available and less restrictive than that required for other classification techniques.

As discussed throughout the thesis, the monotonic constraint overcomes the unpredictability of classification boundary generation in neural networks. The Monotonic Function model is able to train with statistical data at a much faster speed than in the standard BPLMS neural networks, and is robust in dealing with statistical data (Chapter 3). Monotonicity is also useful in reducing the effect of learning bias in neural networks, a problem which has not been addressed elsewhere. This can be very useful in supplying decision makers with assurance information (Chapter 4). Neural networks with monotonic properties can also represent fuzzy membership functions in classification, and supply decision makers with more natural uncertainty information represented in fuzzy membership forms. This has the potential to serve as a component in fuzzy reasoning (Chapter 5).

The application of monotonicity in neural network classification is not limited to typical managerial pattern recognition problems. Rather, the neural network under monotonic constraints is a promising classifier which can be used in other fields with two or more class classification (e.g. target identification [Breiman et al. 1984] [Chien and Killeen 1982] [Barnard and Casasent 1989]). If a neural network is to be used as an artificial intelligence machine for improving understanding

through the interplay of multiple sources of knowledge [Rumelhart et al. 1986], then, as has been argued, the characteristic of monotonicity is a critically important component of the knowledge sources used in classification.

7.2. REMARKS

Let us recall the comparison of various classification techniques shown in Table 1.1. The restrictions of each technique make it very clear that no single technique can be absolutely superior over the others in all situations. On the other hand, it has been shown in the present research that the BPLMS neural network approach preceded by a linear discriminant analysis pre-processor and supplemented by problem domain knowledge (e.g. monotonicity, and artificial data points) can significantly improve classification performance in many cases. Although the present research has not implemented an integrated classification system, there is no apparent obstacle to building one. There are three main components of an integrated classification system which could accomplish the desired functions, as follows:

(1) An expert system component :

- Verify monotonic variables;
- Identify the inflection points in non-monotonic variables, and decompose these into monotonic variables;

. Define specific knowledge about the extreme values of each pattern variable for a particular class (see Section 4.5.1) and transform it into artificial training data;

. Check fuzzy membership functions generated by the neural networks.

The performance of the functions listed above relies heavily on human expertise. Consequently, most of these functions would be accomplished interactively with the user, and the expert system would include an advice-giving function to support the user.

(2) Linear discriminant analysis component :

. Pre-process the statistical sample data set.

(3) Neural network component :

. Generate the decision boundary, or boundaries;

. Supply fuzzy membership functions and uncertainty information about the decision boundaries.

A possible way to implement an integrated system would be to employ an expert system shell which would host the discriminant analysis and neural network components.

7.3. FUTURE RESEARCH

The present research might be extended in at least two directions. One is related to the neural network learning algorithm. As discussed in the thesis, the present research has solved the

unpredictability problem of neural networks for managerial classification problems; however, certain components of the classification model, such as vector analysis, do not belong to the neural network model itself. The question of whether the neural network could handle unpredictability problems solely through learning remains unsolved, and should be addressed.

Another possible research direction is related to applications. Direct application of the models developed in this thesis to the solution of practical classification problems is simple and straightforward. However, since we usually have no idea about the "true optimal" decision boundary in any particular problem, there would be little theoretical value of a single practical application in terms of general evaluation of accuracy and efficiency of the developed models. The question of whether the models developed in this research could extend into other applications, such as model management in decision support systems, is an interesting area which can be explored.

APPENDIX I.

GENERATING RANDOM SAMPLES WITH TWO CLASSES TO TEST THE NEURAL NETWORKS

A program generating the desired random sample with a two-variate variable performs as follows.

(1) Generate 0-1 uniform random numbers.

(2) Generate N(0,1) random numbers.

(3) Generate two-variate normal distribution.

(4) Shift into two Bernoulli groups.

(5) Transform the linear boundary to desired shape.

In more detail :

(1) Using time as the seed generate 0-1 uniform random numbers. This is implemented simply by Turbo C math functions.

(2) Following the procedure suggested by Law and Kelton (1982), generate N(0,1) normal distribution sample. Suppose we take uniform random numbers u_1 and u_2 , then let $v_1 = 2u_1 - 1$, $v_2 = 2u_2 - 1$, and $w = v_1^2 + v_2^2$. If $w > 1$ throw the result away, and try again; else let $y = (-2 \ln w / w)^{0.5}$, and $z_1 = v_1 y$, $z_2 = v_2 y$; then z_1 and z_2 are random samples from the N(0,1) normal distribution.

(3) In order to generate a two-variate normal distribution, the procedure suggested by Scheuer and Stoller (1962) is used.

Suppose the desired bi-normal distribution has μ_1, μ_2 and

$$\Sigma = \begin{vmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{vmatrix} .$$

Let $c_{11} = \sigma_{11}^{0.5}$, $c_{21} = \sigma_{21} / \sigma_{11}^{0.5}$, $c_{22} = (\sigma_{22} - \sigma_{21}^2 / \sigma_{11})^{0.5}$, then $x_1 = c_{11} * z_1 + \mu_1$ and $x_2 = c_{21} * z_1 + c_{22} * z_2 + \mu_2$ are the desired bi-variate normal random observation.

(4) Shift the homogeneous sample into two Bernoulli groups following

$$x_1' = x_1 + b\ y - b\ (1 - y)$$

$$x_2' = x_2 - c\ y + c\ (1 - y), \quad y = 0, 1$$

where b, c are parameters which specify the transformation.

y is a constant rather than a random variable, to ensure the two-group sample is still a bi-variate normal distribution.

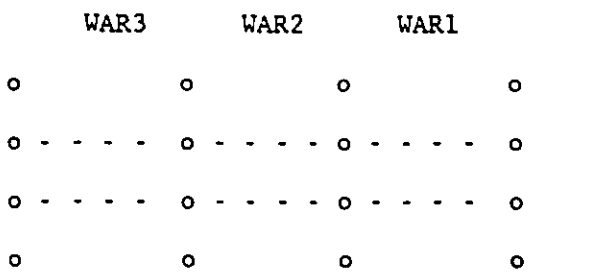
(5) By Bayes rule or discriminant analysis the boundary between the two shifted groups is linear (see Chapter 1). In order to generate an arbitrary boundary, we may simply transform the coordinate. For example, if we let $x_2'' = e^{x_2'}$, the boundary is shifted to an exponential shape. Note that after the transformation the sample is no longer a bi-variate normal distribution.

Finally, the sample is normalized into the range $x_1[0, 1]$, $x_2[0, 1]$.

APPENDIX II.

BACK PROPAGATION ALGORITHM

Architecture



n	m	h	r
IND1	HN2	HN1	OUF
TISN	OUH2	OUH1	TOSN
input	hidden2	hidden1	output

Variables

Defined by user:

IND	Number of input dimension
OUF	Number of feature output
HN1	Number of first hidden layer nodes
HN2	Number of second hidden layer nodes
ETA	Learning rate
MEPS	Error maximum tolerance
NTE	Number of teaching samples
TEM	Temperature

MOM Momentum

Defined by the program:

LRS Learning sweep number

OUH2(HN2) Array of each HN2 node output

OUH1(HN1) Array of each HN1 node output

OUT(OUF) Array of each output node

TIS(NTE, IND1) Array of teaching sample input

TOS(NTE, OUF) Array of teaching sample output

TISN(NTE, IND1) Array of normalized sample input

TOSN(NTE, OUF) Array of normalized sample output

WAR1(HN1, OUF)

Array of weights between output and first hidden layers

LWC1(HN1, OUF) Array of last weight changes in WAR1

DLT1(OUF) Error signal at WAR1

If HN2 not=0

WAR2(HN2, HN1)

Array of weights between first and second hidden layers

LWC2(HN2, HN1) Array of last weight changes in WAR2

DLT2(HN1) Error signal at WAR2

WAR3(IND1, HN2)

Array of weights between second and input layers

LWC3(IND1, HN2) Array of last weight changes in WAR3

DLT3(HN2) Error signal at WAR3

Else (HN2=0)

WAR2(IND1, HN1)

LWC2(IND1, HN1)

DLT2(HN1)

EPS Sum of error square

FL1 Flag for EPS > MEPS

BUF Working storage

Functions

F(ALH) = 1 / (1 + e ** (-ALH)/TEM)

Subroutines

.Initial-accept accept IND, OUT, HN1, HN2, NTE, ETA, MEPS

.Initial-assign LRS=0, FL1=0, IND1=IND + 1

.Initial-generate

Accept TIS and TOS

Normalize input, output TISN, TOSN from TIS, TOS

TISN(1)=1

Generate initial weights values for WAR1 WAR2

if NH2 not= 0 generate random values for WAR3

Main programming of back-propagation algorithm

```
1.  Call  Initial-accept
    Call  Initial-assign
    Call  Initial-generate

2.  Do  t=1  To  NTE
        /* compute actual output */
        If  HN2 not= 0  Then
            OUH2(1)=1
            Do  m=2  To HN2
                ALH=0
                Do  n=1  To  IND1
                    ALH=ALH+WAR3(n,m)*TISN(t,n)
                End-do
                OUH2(m)=F(ALH)
            End-do
            OUH1(1)=1
            Do  h=2  To  HN1
                ALH=0
                Do  m=1  To  HN2
                ALH=ALH+WAR2(m,h)*OUH2(m)
                End-do
```

```
        OUH1(h)=F(ALH)
    End-do
Else
    OUH1(1)=1
    Do  h=2  To HN1
        ALH=0
        Do  n=1  To IND1
        ALH=ALH+WAR2(n,h)*TISN(t,n)
        End-do
        OUH1(h)=F(ALH)
    End-do
End-if

Do  r=1  To  OUF
    ALH=0
    Do  h=1  To  HN1
    ALH=ALH+WAR1(h,r)*OUH1(h)
    End-do
    OUT(r)=F(ALH)
End-do

/* compute-EPS */
EPS=0
Do  r=1  To  OUF
EPS=EPS+(TOSN(t,r)-OUT(r))**2
End-do
```

```
If  EPS > MEPS
    Then FL1=1
      /* Call Back-propagation */
      Do  r=1  To  OUF
          DLT1(r)=OUT(r)*(1-OUT(r))*(TOSM(t,r)-OUT(r))/T
      End-do

      Do  h=1  To  HN1
          Do  r=1  To OUF
              WAR1(h,r)=WAR1(h,r)+ETA*DLT1(r)*OUH1(h)+
                                  +MOM*LWC1(h,r)
              LWC1(h,r)=ETA*DLT1(r)*OUH1(h)
          End-do
      End-do

      Do  h=1  To HN1
          BUF=0
          Do  r=1  To  OUF
              BUF=BUF+DLT1(r)*WAR1(h,r)
          End-do
          DLT2(h)=OUH1(h)*(1-OUH1(h))*BUF/T
      End-do

  If  HN2 not= 0  Then
      Do  m=1  To  HN2
          Do  h=1  To  HN1
              WAR2(m,h)=WAR2(m,h)+ETA*DLT2(h)*OUH2(m)+
```

```
+MOM*LWC2(m,h)
            LWC2(m,h)=ETA*DLT2(h)*OUH2(m)
        End-do
    End-do
    Do  m=1  To  HN2
        BUF=0
        Do  h=1  To  HN1
        BUF=BUF+DLT2(h)*WAR2(m,h)
        End-do
        DLT3(m)=OUH2(m)*(1-OUH2(m))*BUF/T
    End-do

    Do  n=1  To  IND1
        Do  m=1  To  HN2
            WAR3(n,m)=WAR3(n,m)+ETA*DLT3(m)*TISN(t,n)+
                                +MOM*LWC3(n,m)
            LWC3(n,m)=ETA*DLT3(m)*TISN(n)
        End-do
    End-do
Else (HN2=0)
    Do  n=1  To  IND1
        Do  h=1  To  HN1
            WAR2(n,h)=WAR2(n,h)+ETA*DLT2(h)*TISN(t,n)+
                                +MOM*LWC2(n,h)
            LWC2(n,h)=ETA*DLT2(h)*TISN(t,n)
        End-do
    End-do
```

```
          End-if
        /*  End of Back-propagation  */

        Else
             FL1=0
        End-if
    End-do

3.  If FL1=1
      Then  LRS=LRS+1
            Goto  2
    Else  Call Print-all-weight-and-LRS
```

APPENDIX III.

MONOTONIC CONDITION

This Appendix is a derivation of the monotonic condition in the MF model, corresponding to algorithm (3.17), Chapter 3. The numbers in brackets to the left correspond to the numbered sections of the algorithm (3.17).

(1) Let

$$\sum_{i=1}^{h} w_{ij}\, v_i \exp(-U_iX')[(1+\exp(-U_iX'))]^{-2} =$$

$$\sum_{i\in I} w_{ij}\, v_i \exp(-U_iX')[(1+\exp(-U_iX'))]^{-2} +$$

$$\sum_{i\notin I} w_{ij}\, v_i \exp(-U_iX')[(1+\exp(-U_iX'))]^{-2}$$

where I is a set such that, if $i\in I$, $w_{ij}v_i<0$, that is,

$$\sum_{i\in I} w_{ij}\, v_i \exp(-U_iX')[(1+\exp(-U_iX'))]^{-2}<0 \ ;$$

and if $i\notin I$ for $w_{ij}v_i\geq 0$, that is,

$$\sum_{i\notin I} w_{ij}\, v_i \exp(-U_iX')[(1+\exp(-U_iX'))]^{-2}\geq 0 \ ;$$

(since $\exp(-U_iX')\ [\ (1+\exp(-U_iX'))]^{-2} > 0$).

(2) Monotonicity $\langle\!\!-\!\!\rangle$ $\sum_{i=1}^{h} w_{ij}\, v_i \exp(-U_iX')[(1+\exp(-U_iX'))]^{-2} \geq 0$,

and each term in the sum is independent. Thus,

$$\min \sum_{i \in I} w_{ij} v_i \exp(-U_i X')[(1+\exp(-U_i X'))]^{-2}$$

$$+ \min \sum_{i \notin I} w_{ij} v_i \exp(-U_i X')[(1+\exp(-U_i X'))]^{-2} \geq 0$$

is a necessary and sufficient condition for monotonicity.

Denote $\min \sum_{i \in I} w_{ij} v_i \exp(-U_i X')[(1+\exp(-U_i X'))]^{-2}$ and

$\min \sum_{i \notin I} w_{ij} v_i \exp(-U_i X')[(1+\exp(-U_i X'))]^{-2}$ as Q_1 and Q_2, respectively.

$\exp(-U_i X')\ [\ (1 + \exp(-U_i X')]^{-2}$ has a maximum of 0.25 at $U_i X'=0$ (because at least one point $X=\underline{0}$, and this maximum occurs at that point), hence

$$Q_1 = 0.25 \sum_{i \in I} w_{ij} v_i .$$

(3) We now need to find $Q_2 = \min \sum_{i \notin I} w_{ij} v_i \exp(-U_i X')[(1+\exp(-U_i X'))]^{-2}$ (when $w_{ij} v_i \geq 0 \quad \forall i$).

Let $R_i = \exp(-U_i X')[(1+\exp(-U_i X'))]^{-2}$. R_i is a function of the quantity $\sum_{r=0}^{m} w_{ir} x_r$, since $R_i = \exp(-\sum_{r=0}^{m} w_{ir} x_r)[(1+\exp(-\sum_{r=0}^{m} w_{ir} x_r))]^{-2}$.

The behavior of R_i as a function of $\sum_{r=0}^{m} w_{ir} x_r$ is shown in Figure A.1.

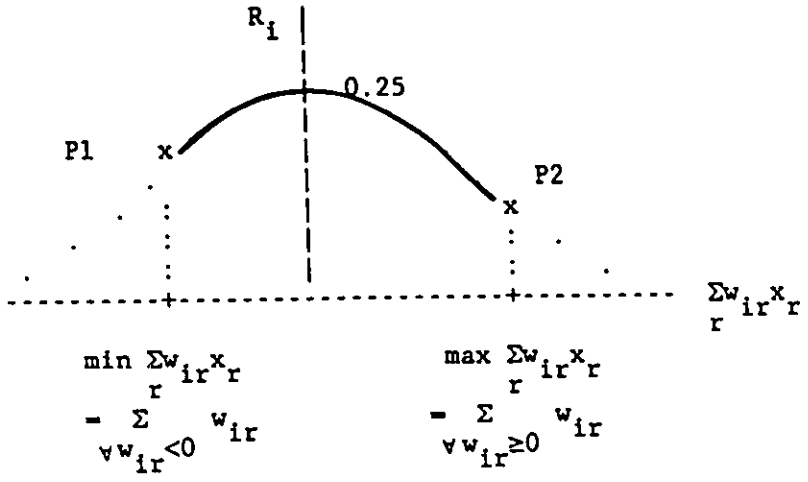


Fig. A.1. The Behavior of R_i as a Function of $\sum_{r=0}^{m} w_{ir} x_r$.

Since $0 \leq x_r \leq 1$, we have

$$0 \leq \sum_r w_{ir} x_r \leq \sum_r w_{ir} \qquad \forall\, w_{ir} \geq 0 \text{ , and}$$

$$\sum_r w_{ir} \leq \sum_r w_{ir} x_r \leq 0 \qquad \forall\, w_{ir} < 0.$$

The extreme points are P_1 (x_r=1 when $w_{ir}<0$, and x_r=0 when $w_{ir}\geq 0$) and

P_2 (x_r=1 when $w_{ir}\geq 0$, and x_r=0 when $w_{ir}<0$).

Note that the two extreme points P1 and P2 of the function R_i depend on the specific Σw_{ir} values, and either point could be the minimum.

Thus,

$$Q_2 = \min \sum_{i \notin I} w_{ij} v_i \exp(-U_i X')[(1+\exp(-U_i X'))]^{-2}$$

$$= \sum_{i \notin I} \min\, w_{ij} v_i \exp(-U_i X')[(1+\exp(-U_i X'))]^{-2}$$

(since each independent term ≥ 0)

$$= \Sigma\, w_{ij} v_i \,(\min \exp(-U_i X')[(1+\exp(-U_i X'))]^{-2})$$

$$= \Sigma\, w_{ij} v_i\, A_i$$

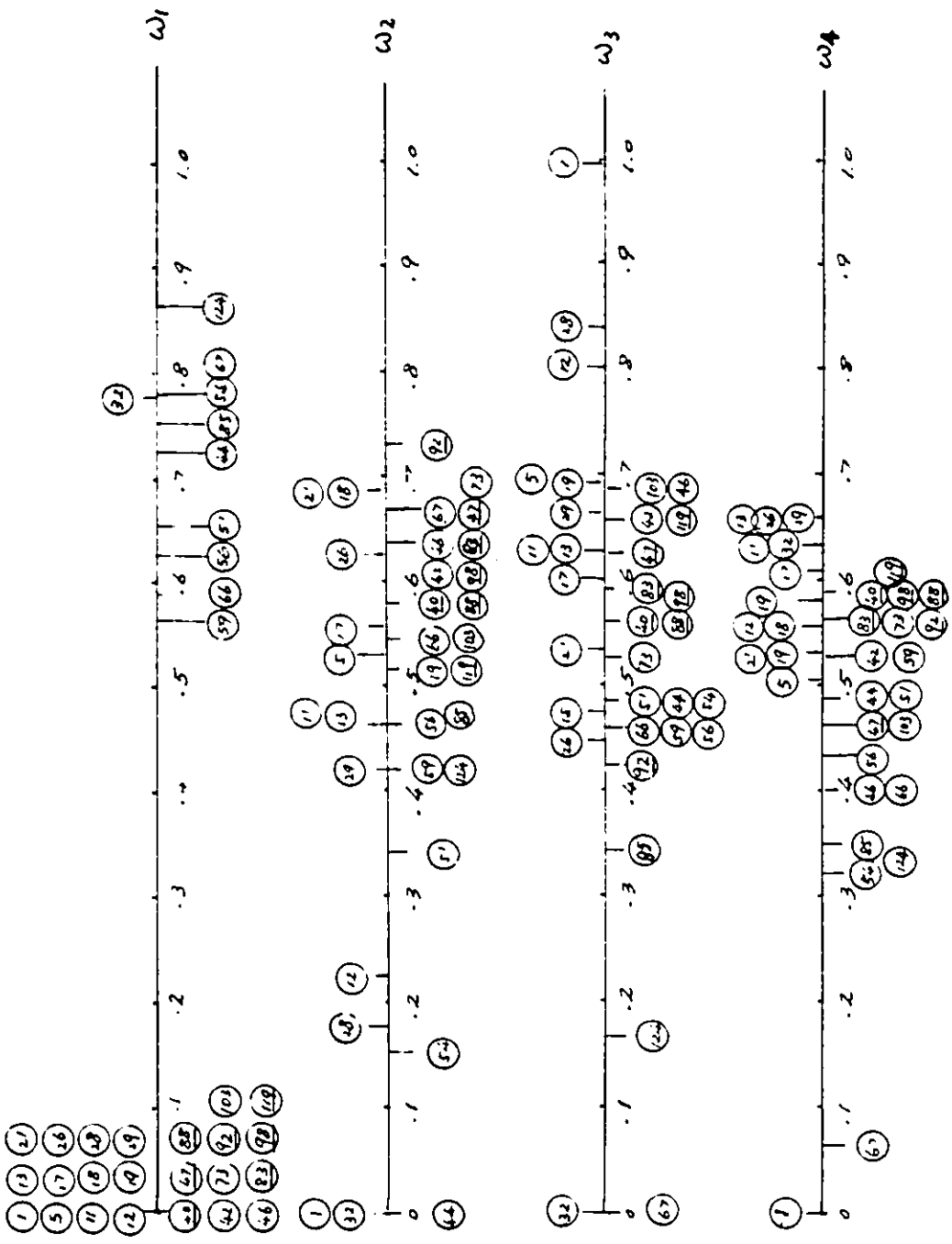
where A_i is the minimum value of P1 and P2.

(4) If $Q_1 + Q_2 \geq 0$ then $\sum_{i=1}^{h} w_{ij}\, v_i \exp(-U_i X')[(1+\exp(-U_i X'))]^{-2} \geq 0$,

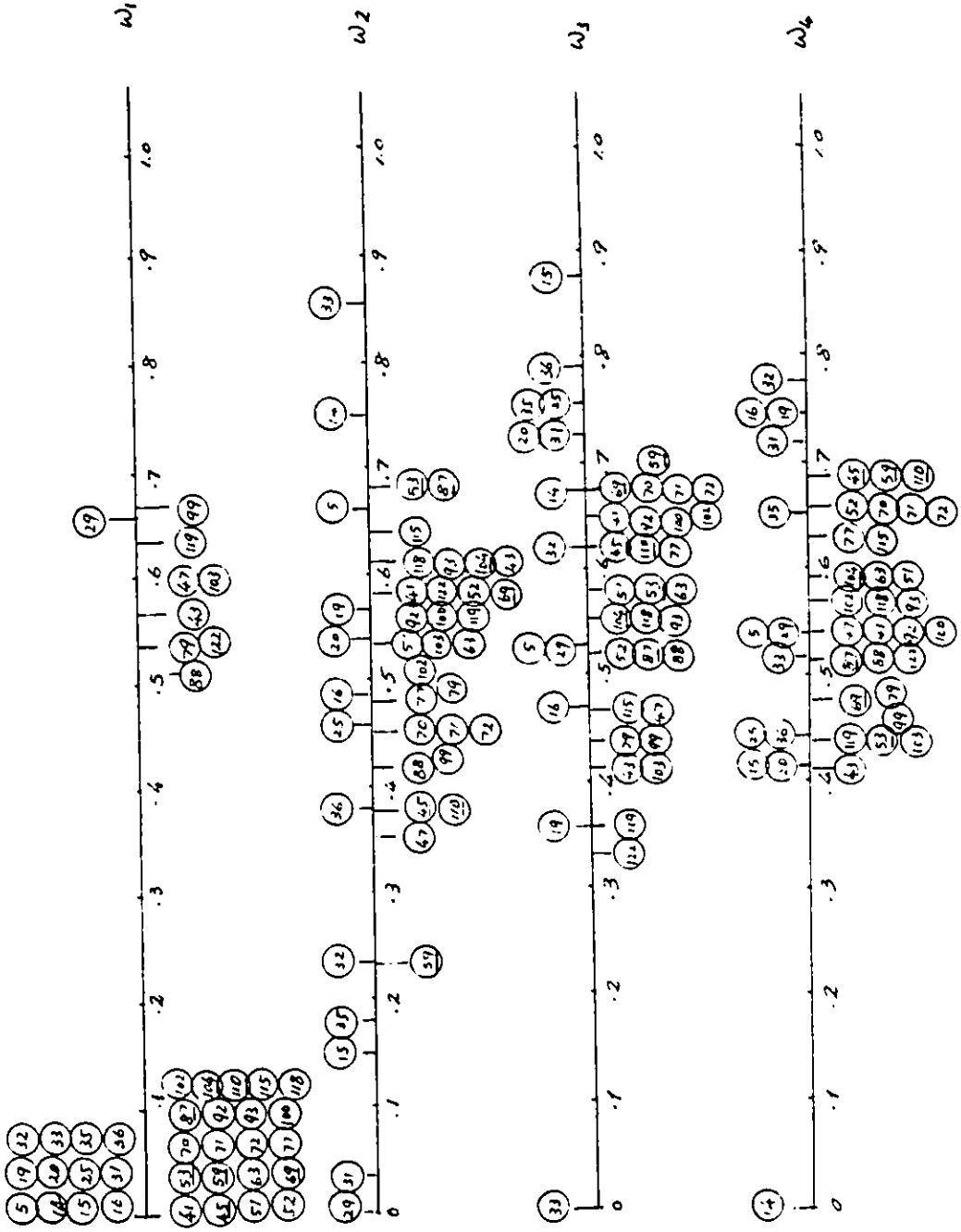
that is, y is monotonic; else monotonicity is violated.

APPENDIX IV.

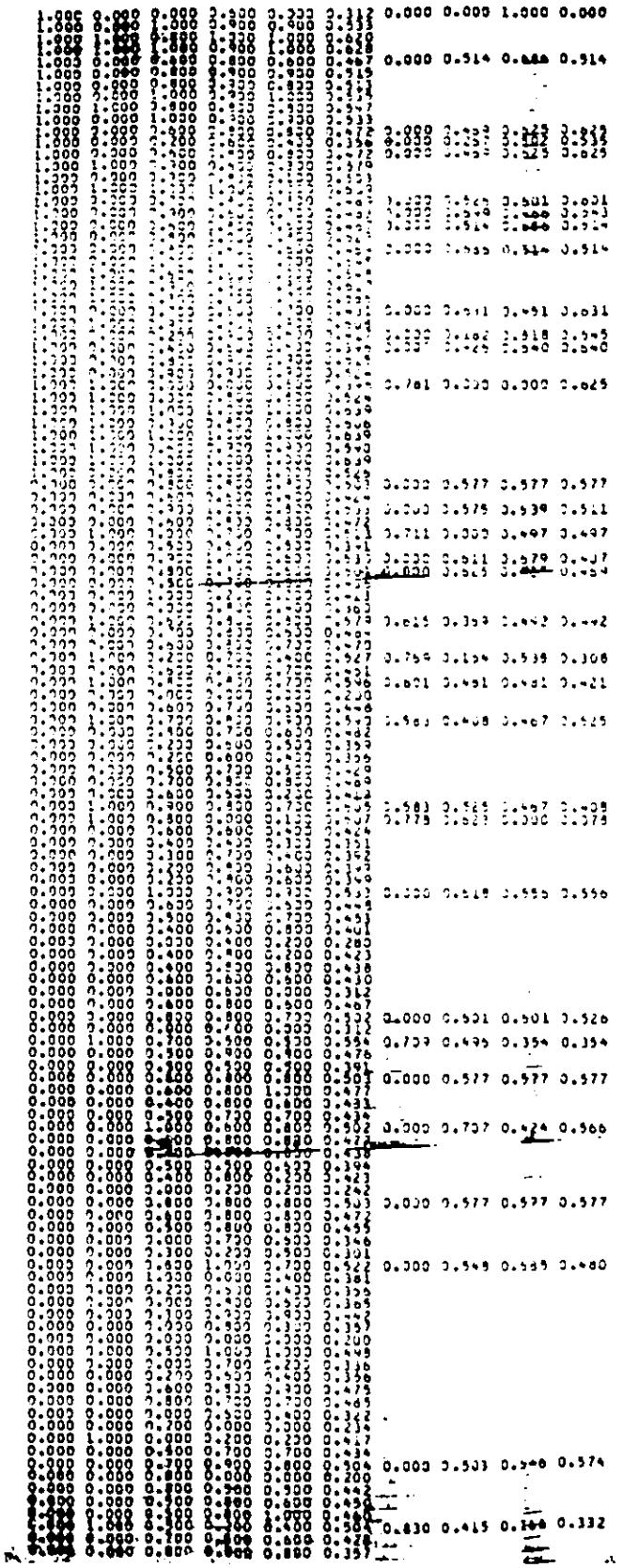
AN EXPERIMENT ON GREEN'S DATA [Green 1978]



Appendix IV (1) - 1. Vector analysis on subset S_1
(The points not underlined were ignored in further learning)

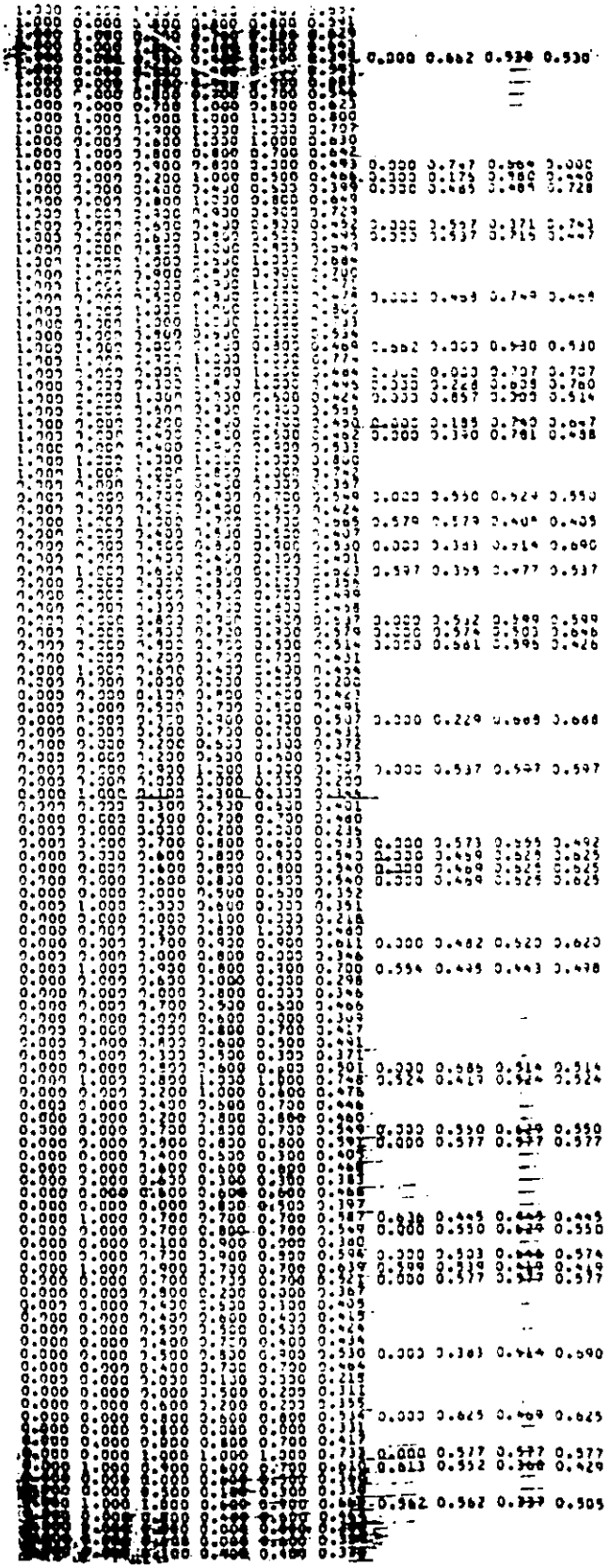


Appendix IV (1) - 2. Vector analysis on subset S_2
(The points not underlined were ignored in further learning)



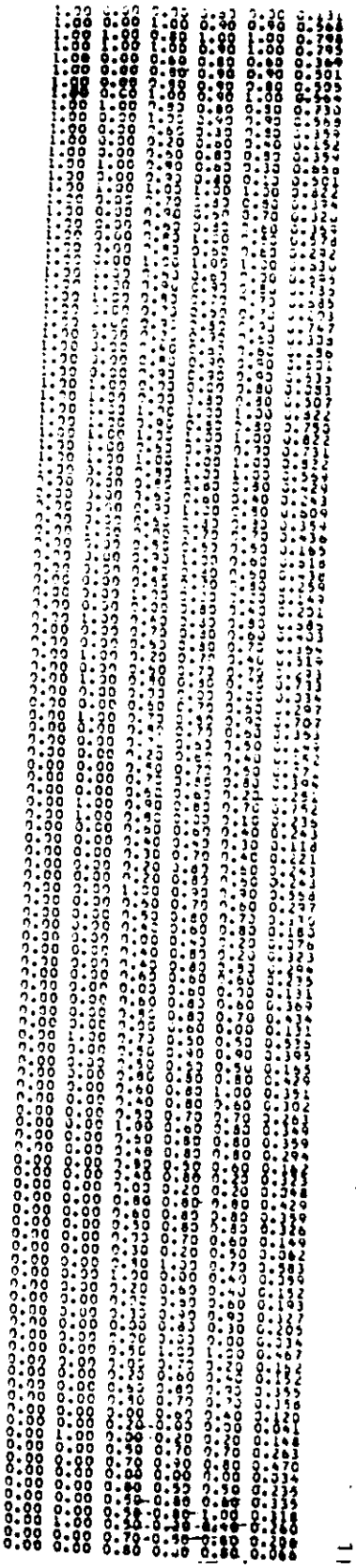
Appendix IV (2)
1. Data subset S_1

Column coding:
1: class i/0;
2-5: pattern vector (normalized)
6: normalized classification score of LDA
7-10: $\cos\omega_i$ values for the misclassified points



Appendix IV (2)
2. Data subset S_2

Column coding:
1: class i/0;
2-5: pattern vector (normalized)
6: normalized classification score of LDA
7-10: $\cos\omega_i$ values for the misclassified points



Appendix IV (2)

3. The classification results of the neural network trained on S_1 .

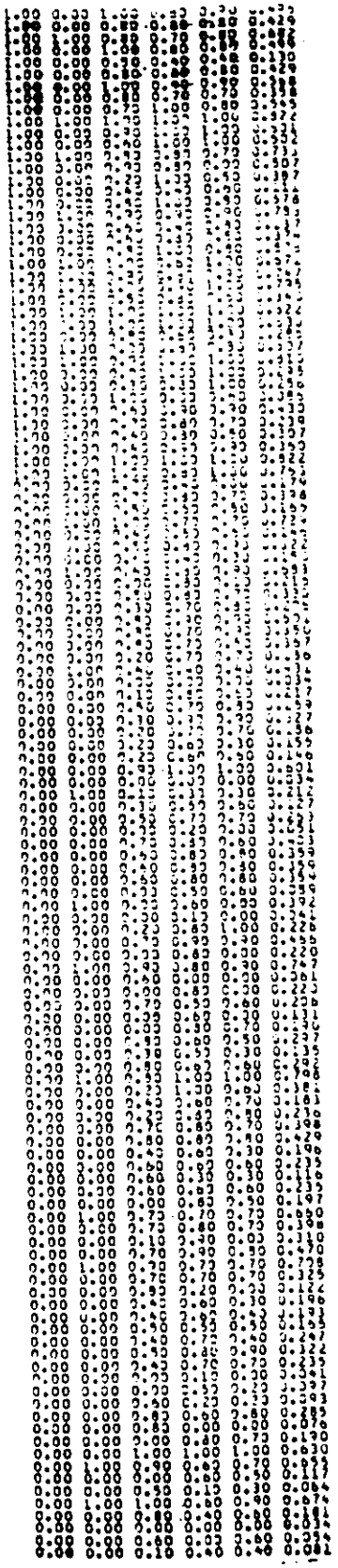
and tested on S_1

Column coding:

1: class 1/0

2-5: pattern vector (normalized)

6: the neural network classification results



Appendix IV (2)
4. The classification results of the neural network trained on S_1 , and tested on S_2

Column coding:
1: class 1/0
2-5: pattern vector (normalized)
6: the neural network classification results

0.041505
-0.206648
0.959130
1.187891
0.895159
1.190988
1.106281
-1.004076
-2.291144
-0.223252
1.069635
1.008359
1.253073
-1.396963
0.089747
0.661672
0.443358
0.530773
0.537712
0.101149
0.337741
0.113039
0.913428
0.999945
0.611745
0.419201
0.345126
0.810114
0.809940
0.374824
0.423078
0.621911
0.113926
0.481202
0.446421
0.553636
-0.127093
1.036595
0.743204
0.292444
0.104411
0.391057
0.535443
1.020679
0.491627
0.170568
1.007422
0.144200
0.193162
0.394537
0.058990
0.095114
0.923935
2.329717
1.071485
-0.216754
0.371797
-0.066052
1.490784
0.714552
0.974670
0.028819
0.436919
0.244005
0.176113
0.226018
0.280360
0.199812
0.003891
0.239089
0.874375
0.494486
0.701226
-0.195875
0.067954
-2.695841
0.838822
-0.361881
-0.489346
-0.160528
-0.526825
-0.532903
1.460353
2.927351
0.952803
-0.700359
-0.277374
-0.710496
2.054515
0.615965

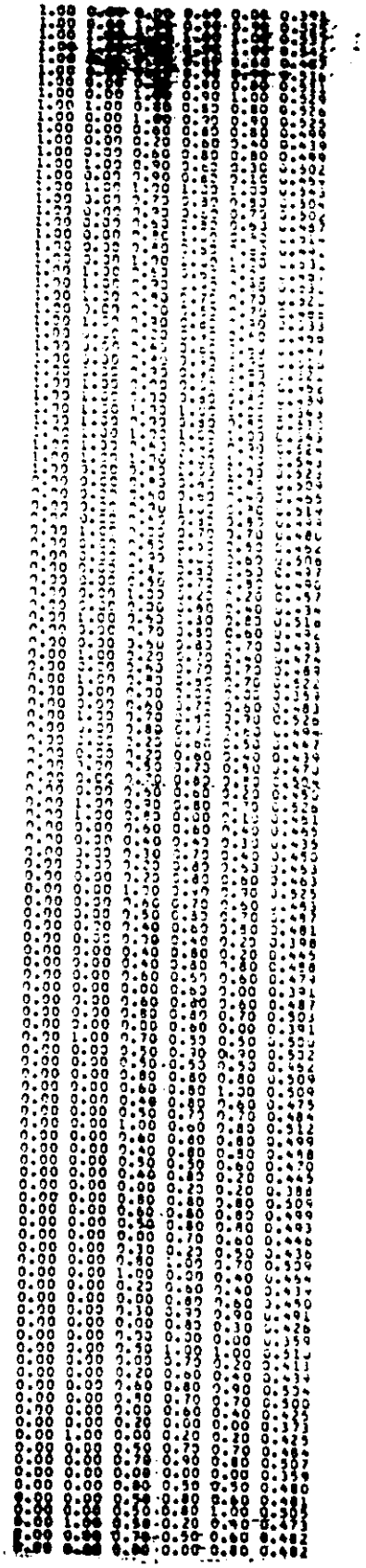
Appendix IV (2)

5. The neural network weights after training on data set S_1

order:

w_{ij} , j=0...4,
i=0...14;

v_i , i=0...14.



Appendix IV (2)

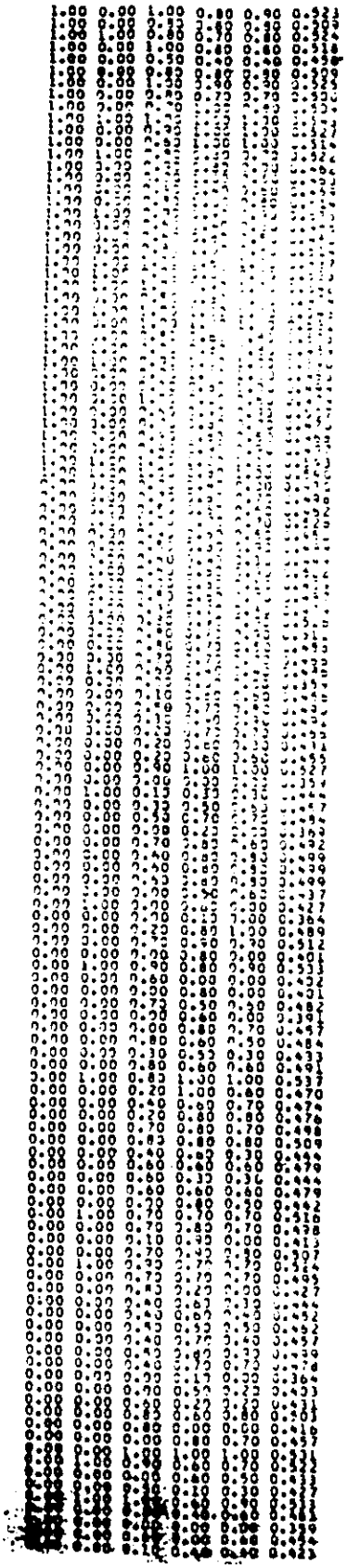
6. The classification results of the neural network trained on S_2, and tested on S_1

Column coding:

1: class 1/0

2-5: pattern vector (normalized)

6: the neural network classification results



Appendix IV (2)

7. The classification results of the neural network trained on S_2 , and tested on S_2

Column coding:

1: class 1/0

2-5: pattern vector (normalized)

6: the neural network classification results

0.041505
0.285218
0.826567
0.871881
0.760103
0.923806
0.904049
0.094966
-0.079980
0.509979
0.861933
0.876666
0.956659
0.162547
0.477847
0.561672
0.598754
0.548484
0.718688
0.113342
0.405173
0.121647
0.670941
0.476085
0.491339
0.563706
0.344246
0.867950
0.265292
0.180649
0.423078
0.699285
0.122002
0.403858
0.637674
0.561142
0.005952
0.859179
0.444947
0.419943
0.368609
0.377141
0.522729
0.945934
0.387682
0.170568
0.818598
0.279585
0.297133
0.405294
0.228173
0.420429
0.053423
0.499210
0.886702
-0.003620
0.405122
0.064802
0.152007
0.299413
0.974670
0.320273
0.392295
0.106156
0.193367
0.177031
0.292663
0.753098
1.014128
0.962582
0.729408
0.475730
0.488539
0.195336
0.230386
-1.273931
0.063622
-0.150634
-0.373976
0.012941
-0.207106
-0.088583
0.431034
0.847362
0.606875
-0.083948
0.008981
-0.271553
0.614385
0.213654

Appendix IV (2)

8. The neural network weights after training on data set S_2

order:

w_{ij} , j=0...4,
i=0...14;

v_i , i=0...14.

APPENDIX V.

AN EXPERIMENT ON FISHER'S DATA [Fisher 1936]

Appendix V (1) - The Iris Data [Fisher 1936]

	Group I				Group II					Group III				
ID	x_1	x_2	x_3	x_4	ID	x_1	x_2	x_3	x_4	ID	x_1	x_2	x_3	x_4
1	5.1	3.5	1.4	0.2	51	7.0	3.2	4.7	1.4	101	6.3	3.3	6.0	2.5
2	4.9	3.0	1.4	0.2	52	6.4	3.2	4.5	1.5	102	5.8	2.7	5.1	1.9
3	4.7	3.2	1.3	0.2	53	6.9	3.1	4.9	1.5	103	7.1	3.0	5.9	2.1
4	4.6	3.1	1.5	0.2	54	5.5	2.3	4.0	1.3	104	6.3	2.9	5.6	1.8
5	5.0	3.6	1.4	0.2	55	6.5	2.8	4.6	1.5	105	6.5	3.0	5.8	2.2
6	5.4	3.9	1.7	0.4	56	5.7	2.8	4.5	1.3	106	7.6	3.0	6.6	2.1
7	4.6	3.4	1.4	0.3	57	6.3	3.3	4.7	1.6	107	4.9	2.5	4.5	1.7
8	5.0	3.4	1.5	0.2	58	4.9	2.4	3.3	1.0	108	7.3	2.9	6.3	1.8
9	4.4	2.9	1.4	0.2	59	6.6	2.9	4.6	1.3	109	6.7	2.5	5.8	1.8
10	4.9	3.1	1.5	0.1	60	5.2	2.7	3.9	1.4	110	7.2	3.6	6.1	2.5
11	5.4	3.7	1.5	0.2	61	5.0	2.0	3.5	1.0	111	6.5	3.2	5.1	2.0
12	4.8	3.4	1.6	0.2	62	5.9	3.0	4.2	1.5	112	6.4	2.7	5.3	1.9
13	4.8	3.0	1.4	0.1	63	6.0	2.2	4.0	1.0	113	6.8	3.0	5.5	2.1
14	4.3	3.0	1.1	0.1	64	6.1	2.9	4.7	1.4	114	5.7	2.5	5.0	2.0
15	5.8	4.0	1.2	0.2	65	5.6	2.9	3.9	1.3	115	5.8	2.8	5.1	2.4
16	5.7	4.4	1.5	0.4	66	6.7	3.1	4.4	1.4	116	6.4	3.2	5.3	2.3
17	5.4	3.9	1.3	0.4	67	5.6	3.0	4.5	1.5	117	6.5	3.0	5.5	1.8
18	5.1	3.5	1.4	0.3	68	5.8	2.7	4.1	1.0	118	7.7	3.8	6.7	2.2
19	5.7	3.8	1.7	0.3	69	6.2	2.2	4.5	1.5	119	7.7	2.6	6.9	2.3
20	5.1	3.8	1.5	0.3	70	5.6	2.5	3.9	1.1	120	6.0	2.2	5.0	1.5
21	5.4	3.4	1.7	0.2	71	5.9	3.2	4.8	1.8	121	6.9	3.2	5.7	2.3
22	5.1	3.7	1.5	0.4	72	6.1	2.8	4.0	1.3	122	5.6	2.8	4.9	2.0
23	4.6	3.6	1.0	0.2	73	6.3	2.5	4.9	1.5	123	7.7	2.8	6.7	2.0
24	5.1	3.3	1.7	0.5	74	6.1	2.8	4.7	1.2	124	6.3	2.7	4.9	1.8
25	4.8	3.4	1.9	0.2	75	6.4	2.9	4.3	1.3	125	6.7	3.3	5.7	2.1
26	5.0	3.0	1.6	0.2	76	6.6	3.0	4.4	1.4	126	7.2	3.2	6.0	1.8
27	5.0	3.4	1.6	0.4	77	6.8	2.8	4.8	1.4	127	6.2	2.8	4.8	1.8
28	5.2	3.5	1.5	0.2	78	6.7	3.0	5.0	1.7	128	6.1	3.0	4.9	1.8
29	5.2	3.4	1.4	0.2	79	6.0	2.9	4.5	1.5	129	6.4	2.8	5.6	2.1
30	4.7	3.2	1.6	0.2	80	5.7	2.6	3.5	1.0	130	7.2	3.0	5.8	1.6
31	4.8	3.1	1.6	0.2	81	5.5	2.4	3.8	1.1	131	7.4	2.8	6.1	1.9
32	5.4	3.4	1.5	0.4	82	5.5	2.4	3.7	1.0	132	7.9	3.8	6.4	2.0
33	5.2	4.1	1.5	0.1	83	5.8	2.7	3.9	1.2	133	6.4	2.8	5.6	2.2
34	5.5	4.2	1.4	0.2	84	6.0	2.7	5.1	1.6	134	6.3	2.8	5.1	1.5
35	4.9	3.1	1.5	0.2	85	5.4	3.0	4.5	1.5	135	6.1	2.6	5.6	1.4
36	5.0	3.2	1.2	0.2	86	6.0	3.4	4.5	1.6	136	7.7	3.0	6.1	2.3
37	5.5	3.5	1.3	0.2	87	6.7	3.1	4.7	1.5	137	6.3	3.4	5.6	2.4
38	4.9	3.6	1.4	0.1	88	6.3	2.3	4.4	1.3	138	6.4	3.1	5.5	1.8
39	4.4	3.0	1.3	0.2	89	5.6	3.0	4.1	1.3	139	6.0	3.0	4.8	1.8
40	5.1	3.4	1.5	0.2	90	5.5	2.5	5.0	1.3	140	6.9	3.1	5.4	2.1
41	5.0	3.5	1.3	0.3	91	5.5	2.6	4.4	1.2	141	6.7	3.1	5.6	2.4
42	4.5	2.3	1.3	0.3	92	6.1	3.0	4.6	1.4	142	6.9	3.1	5.1	2.3
43	4.4	3.2	1.3	0.2	93	5.8	2.6	4.0	1.2	143	5.8	2.7	5.1	1.9
44	5.0	3.5	1.6	0.6	94	5.0	2.3	3.3	1.0	144	6.8	3.2	5.9	2.3
45	5.1	3.8	1.9	0.4	95	5.6	2.7	4.2	1.3	145	6.7	3.3	5.7	2.5
46	4.8	3.0	1.4	0.3	96	5.7	3.0	4.2	1.2	146	6.7	3.0	5.2	2.3
47	5.1	3.8	1.6	0.2	97	5.7	2.9	4.2	1.3	147	6.3	2.5	5.0	1.9
48	4.6	3.2	1.4	0.2	98	6.2	2.9	4.3	1.3	148	6.5	3.0	5.2	2.0
49	5.3	3.7	1.5	0.2	99	5.1	2.5	3.0	1.1	149	6.2	3.4	5.4	2.3
50	5.0	3.3	1.4	0.2	100	5.7	2.8	4.1	1.3	150	5.9	3.0	5.1	1.8

0.949919
0.108495
0.469895
-0.022973
0.731969
0.555677
0.104969
0.085311
0.940956
0.685537
0.808064
0.813634
1.005982
0.209493
0.430443
0.914637
0.872377
0.852356
0.810907
0.342118
0.608924
0.571155
0.609655
0.533156
0.745569
0.053022
0.472193
0.121433
0.761244
0.463151
0.579194
0.544544
0.358888
0.824111
0.327948
0.082881
0.423078
0.785788
0.153577
0.626891
0.627510
0.638884
-0.039123
0.851283
0.175274
0.248649
-0.535197
0.086066
-0.465913
0.145553
-0.026466
0.116675
-0.358488
0.570267

Appendix V (2)

1. The result weights of the neural network corresponding to the linear function classifier for Group I vs. Group II and Group III.
   Order: w_{ij} , j=0...4, i=0...8; v_i , i=0...8.

0.949919
-14.148875
10.927569
-4.906080
-2.183318
-2.665772
21.376488
-13.277008
39.825768
0.685537
5.245275
-4.164439
3.583759
9.528214
3.005856
-5.642430
5.887071
7.509141
0.810907
-6.914767
5.717214
-3.149457
2.907146
-1.828136
9.796270
-7.168872
15.750071
0.121433
10.433985
-8.467744
4.313279
12.111095
3.059761
-15.719791
9.342008
-36.867817
0.423078
14.342949
-11.784514
6.302492
-13.029404
4.034373
-21.035074
13.870870
-40.861168
5.222240
-5.179410
4.766443
-3.508483
9.138285
-2.784471
6.484387
-5.151093
16.165234

Appendix V (2)

2. The result weights of the neural network corresponding to the linear function classifier for Group II vs. Group III. Order: w_{ij} , j=0...4, i=0...8; v_i , i=0...8.

REFERENCES

Ackley, David H., "A Connectionist Machine for Genetic Hillclimbing," Boston: Kluwer Academic Publishers, 1987.

Altman, Edward I., "Financial Ratios, Discriminant Analysis and The Prediction of Corporate Bankruptcy," The Journal of Finance, Volume 28, Number 4, September 1968, 589-609.

Anderson, Dana Z., "Neural Information Processing Systems," New York: American Institute of Physics, 1988.

Balachandra, R., "An Expert System for R&D Projects," in Applied Expert Systems, Turban and Watins, Eds., Amsterdam: North-Holland, 1988, Part II, 107-120.

Barnard, Etienne, and Casasent, David, "A Comparison between Criterion Functions for Linear Classifiers, with an Application to Neural Nets," IEEE Transactions on Systems, Man, and Cybernetics, Volume 19, Number 5, September/October 1989, 1030-1041.

Bellman, R. E., and Zadeh, L. A., "Decision-Making in A Fuzzy Environment," Management Science, Volume 17, Number 4, December 1970, B-141-164.

Bellman, Richard, and Giertz, Magnus, "On the Analytic Formalism of the Theory of Fuzzy Sets," Information Sciences, Volume 5, 1973, 149-156.

Bossel, Hartmut, Klaczko, Salomon, and Muller, Norbert, "Systems Theory in the Social Sciences," Basel: Birkhauser Verlag, 1976.

Bounds, David G., Lloyd, Paul J., Mathew, Bruce, and Waddell, Gordon, "A Multi Layer Perceptron Network for the Diagnosis of Low Back Pain," IEEE International Conference on Neural Networks, San Diego, California, 1988, II-481 - 489.

Breiman, Leo, Friedman, Jerome H., Olshen, Richard A., and Stone, Charles J., "Classification and Regression Trees," Belmont, California: Wadsworth International Group, 1984.

Bremermann, Hans, "Pattern Recognition," in Bossel, Hartmut, Klaczko, Salomon, and Muller, Norbert, "Systems Theory in the Social Sciences," Basel: Birkhauser Verlag, 1976, 116-159.

Bruck, Jehoshua, and Goodman, Joseph W., "A Generalized Convergence Theorem for Neural Networks and its Applications in Combinatorial Optimization," IEEE First International Conference on Neural Networks, San Diego, California, 1987, III-649 - 656.

Bruha, Ivan, and Madhavan, Gopal P., :"Need for a Knowledge-Based Subsystem in Evoked Potential Neural-Net Recognition System," submitted to 7th IEEE Workshop on Languages and Architectures for Automation, Madrid, 1989.

Bruha, Ivan, "Recognition of Waveforms : Neural Net Approach," Dept. of Computer Science and System and Dept. of Medicine, McMaster University, Hamilton, Ontario, Canada, 1989.

Cerveny, Robert P., Garrity, Edward J., and Sanders, Lawrence G., "The Application of Prototyping to System Development: A Rationale and Model," Journal of Management Information Systems, Fall 1986, 52-62.

Chien, Yi-Tzuu, "Interactive Pattern Recognition", New York: Marcel Dekker, Inc., 1978.

Chien, Y. T. and Killeen, T. J., "Computer and Statistical Considerations for Oil Spill Identification," in "Handbook of Statistics 2: Classification, Pattern Recognition and Reduction of Dimensionality", P. R. Krishnaiah and L. N. Kanel, Eds., New York: North-Holland, 1982, 651-671.

Churchill, Gilbert A. Jr., "Marketing Research: Methodological Foundations," Second Edition, Hinsdale, Illinois: The Dryden Press, 1979.

Cover, T. M., and Hart, P. E., "Nearest Neighbor Pattern Classification," IEEE Transactions on Information Theory, Volume IT-13, 1967, 21-27.

Creamer, Elliot M., "Equivalence of Two Methods of Computing Discriminant Function Coefficients," Biometrics, Volume 23, March 1967, 153.

Cybenko, G., "Approximation by Superpositions of A Sigmoidal Function," Mathematics of Control, Signals, and Systems, Volume 2, Number 4, 1989, 303-314.

Denker, John S., and Wittner, Ben S., "Network Generality, Training Required, and Precision Required," in Neural Information Processing Systems, D. Z. Anderson, Ed., New York: American Institute of Physics, 1988, 219-222.

Dubois, Didier, and Prade, Henri, "Fuzzy Sets and Systems: Theory and Applications," New York: Academic Press, Inc., 1980.

Dubois, Didier, and Prade, Henri, "Fuzzy Sets and Statistical Data," European Journal of Operational Research, Volume 25, 1986, 345-356.

Dubois, Didier, and Prade, Henri, "Possibility Theory: An Approach to Computerized Processing of Uncertainty," New York: Plenum Press, 1988.

Dubois, Didier, and Prade, Henri, "Fuzzy Sets, Probability and Measurement," European Journal of Operational Research, Volume 40, May 1989, 135-154.

Duda Richard O., and Hart, Peter E., "Pattern Classification and Scene Analysis," New York: John Wiley & Sons, 1973.

Efron, Bradley, "The Jackknife, the Bootstrap and Other Resampling Plans," Philadelphia, Pennsylvania: Society for Industrial and Applied Mathematics, 1982.

Eisenbeis, Robert A., "Pitfalls in the Application of Discriminant Analysis in Business, Finance, and Economics," The Journal of Finance, Volume 32, Number 3, June 1977, 875-900.

Fikes, Richard, and Kehler, Tom,: "The Role of Frame-Based Representation in Reasoning," Communications of the ACM, Volume 28, Number 9, September 1985, 904-920.

Fisher, R. A., "The Use of Multiple Measurements in Taxonomic Problems," Annals of Eugenics, Volume 7, 1936, 179-188.

Foley, Donald H., "Considerations of Sample and Feature Size," IEEE Transactions on Information Theory, Volume IT-18, Number 5, September 1972, 618-626.

Frank, Ronald E., Massy, William F., and Morrison, Donald G., "Bias in Multiple Discriminant Analysis," Journal of Marketing Research, Volume 2, August 1965, 250-258.

Gentry, James W., "Discriminate Analysis in the Credit Extension Decision," The Credit World, Volume 63, 1974, 25-28.

Gordon, A.D., "Classification," London: Chapman and Hall, 1981.

Gotsman, Craig, Shamir, Eli, and Lehman, Daniel, "Asynchronous Dynamics of Random Boolean Networks," IEEE International Conference on Neural Networks, San Diego, California, 1988, I-1 - 7.

Green, Paul E., "Analyzing Multivariate Data," Hinsdale, Illinois: The Dryden Press, 1978.

Greer, Carl C., "Deciding to Accept or Reject a Marginal Retail Credit Applicant," Journal of Retailing, Volume 43, Number 4, Winter 1968, 44-53.

Griffiths, Lloyd J., "Introduction to Special Section on Neural Networks," IEEE Transactions on Acoustics, Speech, and Signal Processing, Volume 36, Number 7, July 1988, 1107-1108.

Grometstein, Alan, and Schoendorf, William H., "Applications of Pattern Recognition in Radar," in Handbook of Statistics 2: Classification Pattern Recognition and Reduction of Dimensionality, P.R.Krishnaiah and L.N.Kanal, Eds.,1982, Chapter 26, 575-594.

Guyon, Isabelle, "Neural Network System," Proceedings of the INME Symposium, Lausanne, 1989, September 1989.

Guyon, I., Poujaud, L., Persnnaz, L., and Dreyfus, G., "Comparing Different Neural Network Architectures for Classifying Handwritten Digits," Proceedings of IJCNN89, Washington, D.C., June 1989.

Hand, D. J., "Discrimination and Classification," New York: John Willey & Sons, 1981.

Hayes-Roth, Frederick, Waterman, Donald A., and Lenat, Douglas B., Building Expert Systems," London: Addison-Wesley Publishing Company, Inc., 1983.

Healy, M. J. R., "Computing a Discriminant Function from Within-Sample Dispersions," Biometrics, Volume 21, December 1965, 1011-1012.

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

Hecht-Nielsen, Robert, "Kolmogorov's Mapping Neural Network Existence Theorem," IEEE First International Conference on Neural Networks, San Diego, California, 1987, III-11 - 14.

Hillier, Frederick S., and Lieberman, Gerald J., "Introduction to Operations Research," Fourth Edition, Oakland, California: Holden-Day Inc., 1986.

Hinton, G. E., and Sejnowski, T. J., "Learning and Relearning in Boltzman Machine," in "Parallel Distributed Processing : Explorations in the Microstructure of Cognition, Volume 1: Foundations," D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, Eds., Cambridge, Massachusetts: A Bradford Book, The MIT Press, 1986, Chapter 7, p282.

Hirsch, Morris W., "Convergence in Neural Nets," IEEE First International Conference on Neural Networks, San Diego, California, 1987, II-115 - 125.

Hogg, Robert V., and Craig, Allen T., "Introduction to Mathematical Statistics," Fourth Edition, New York: Macmillan Publishing Co., Inc., 1978.

Hopfield, J. J., "Neurons with Graded Response Have Collective Computational Properties Like Those of Two-State Neurons," Proceedings of the National Academy of Sciences, USA, 1984, 81.3088-3092.

Hopfield, J. J., and Tank D., "Neural Computations of Decisions in Optimization Problems," Biological Cybernetics, Volume 52, 1985, 141-152.

Huber, Peter J., "Robust Statistics," New York: John Wiley & Sons, 1981.

Hutchison, William R., and Stephen, Kenneth R., "Integration of Distributed and Symbolic Knowledge Representations," IEEE First International Conference on Neural Networks, San Diego, California, 1987, II-395 - 756.

James, Mike, "Classification Algorithms," New York: John Wiley & Sons, 1985.

Joy, O. Maurice, and Tollefson, John O., "On the Financial Applications of Discriminant Analysis," Journal of Financial and Quantitative Analysis, Volume 10, December 1975, 723-739.

Karplus, Walter J., "The Spectrum of Mathematical Models," Perspectives in Computing, Volume 3, Number 2, May 1983, 4-13.

Keeney, Ralph L., "Utility Functions for Multiattributed Consequences," Management Science, Volume 18, Number 5, January, Part I, 1972, 276-287.

Keeney, Ralph L., and Raiffa, Howard, "Decisions with Multiple Objectives: Preferences and Value Tradeoffs," New York: John Wiley & Sons, 1976.

Keller, James M., and Hunt, Douglas J., "Incorporating Fuzzy Membership Functions into the Perceptr· ı Algorithm," IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume PAMI-7, Number 6, November 1985, 693-699.

Klimasauskas, Casimir C., "The 1989 Neuro-Computing Bibliography," Second Edition, Cambridge, Massachusetts: The MIT Press, 1989.

Kohonen, Teuvo, "Associative Memory," New York: Springer-Verlag, 1977.

Kohonen, Teuvo, "State of the Art in Neural Computing," IEEE First International Conference on Neural Networks, San Diego, California, 1987a, I-79 - 90.

Kohonen, Teuvo, "Self-Learning Inference Rules by Dynamically Expanding Context," IEEE First International Conference on Neural Networks, San Diego, California, 1987b, II-3 - 9.

Kohonen, Teuvo, "Self-Organization and Associative Memory," Berlin: Springer-Verlag, 1988a.

Kohonen, Teuvo, Barna, Gyorgy, and Chrisley, Ronald, "Statistical Pattern Recognition with Neural Networks: Benchmarking Studies," IEEE International Conference on Neural Networks, San Diego, California, 1988b, I-61 - 68.

Kosko, Bart, "Adaptive Inference in Fuzzy Knowledge Networks," IEEE First International Conference on Neural Networks, San Diego, California, 1987, II-261 - 268.

Krishnaiah P.R., and Kanel L.N., "Classification, Pattern Recognition and Reduction of Dimensionality", New York: North-Holland, 1982.

Kung, S. Y., & Hwang, J. N., "An Algebraic Projection Analysis for Optimal Hidden Units Size and Learning Rates in Back-Propagation Learning," IEEE International Conference on Neural Networks, San Diego, California, 1988, I-363 - 370.

Lachenbruch, Peter A., "Discriminant Analysis," New York: Hafner Press, 1975.

Lachenbruch, Peter A., "An Almost Unbiased Method of Obtaining Confidence Intervals for the Probability of Misclassification in Discriminant Analysis," Biometrics, Volume 23, December 1967, 639-645.

Lachenbruch, Peter A., and Mickey, M. Ray, "Estimation of Error Rates in Discriminant Analysis," Technometrics, Volume 10, Number 1, February 1968, 1-11.

Law, Averill M., and Kelton, W. David, "Simulation Modeling and Analysis," New York: McGraw-Hill Book Company, 1982.

Lbov, G.S. "Logical Functions in the Problems of Empirical Prediction," in Classification, Pattern Recognition and Reduction of Dimensionality, Edited by P.R. Krishnaiah and L.N. Kanel, New York: North-Holland, 1982, Chapter 21, p479.

Lehar, Steve, and Weaver, John, "A Developmental Approach to Neural Network Design," IEEE First International Conference on Neural Networks, San Diego, California, 1987, II-97 - 105.

Lippmann, Richard P., "An Introduction to Computing with Neural Nets," IEEE ASSP Magazine, Volume 2, April 1987, 4-22.

Lippmann, Richard P., and Gold, Ben, "Neural-Net Classifier Useful for Speech Recognition," IEEE First International Conference on Neural Networks, San Diego, California, 1987, IV-417 - 425.

Lippmann, Richard P., "Pattern Classification Using Neural Networks," IEEE Communications Magazine, Volume 27, Number 11, November 1989, 47-64.

Mallows, C. L., "Sequential Discrimination," Sankhya: The India Journal of Statistics, Volume 12, Part 4, September, 1953, 321-338.

Martin, James, and Oxman, Steven, "Building Expert Systems: A Tutorial," Englewood Cliffs, New Jersey: Prentic Hall, 1988.

McClelland, James L., Rumelhart, David E., and the PDP Research Group, "Parallel Distributed Processing : Explorations in the Microstructure of Cognition, Volume 2: Foundations," Cambridge, Massachusetts: A Bradford Book, The MIT Press, 1986.

McClelland, J. L., Rumelhart, D. E., and Hinton, G. E., "The Appeal of Parallel Distributed Processing," in Parallel Distributed Processing : Explorations in the Microstructure of Cognition, Volume 1: Foundations," Cambridge, Massachusetts: A Bradford Book, The MIT Press, 1986, Chapter 1, p3.

McClelland, James L., and Rumelhart, David E., "Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises," Cambridge, Massachusetts: A Bradford Book, The MIT Press, 1988.

McCulloch, W. S., and Pitts, W. : "A Logical Calculus of the Ideas Immanent in Neural Activity," Bulletin of Mathematical Biophysics, Volume 5, 1943, 115-133

McGrath, James J., "Improving Credit Evaluation with a Weighted Application Blank," Journal of Applied Psychology, Volume 44, Number 5, 1960, 325-328.

Michalski, Ryszard S., Carbonell, Jaime, G., and Mitchell, Tom M., "Machine Learning: An Artificial Intelligence Approach," Volume I, Los Altos, California: Morgan Kaufman Publishers, Inc., 1983.

Michalski, Ryszard S., Carbonell, Jaime, G., and Mitchell, Tom M., "Machine Learning: An Artificial Intelligence Approach," Volume II, Los Altos, California: Morgan Kaufman Publishers, Inc., 1986.

Miller, Rupert G., "The Jackknife - A Review," Biometrika, Volume 61, Number 1, 1974, 1-15.

Minsky, Marvin, and Papert, Seymour, "Perceptrons : An Introduction to Computational Geometry," Cambridge, Massachusetts: The MIT Press, 1969.

Minsky, Marvin: "A Framework for Representing Knowledge," in The Psychology of Computer Vision, P. H. Winston Ed., New York: McGraw-Hill Book Company, 1975, 211-277.

Morrison, Donald G., "On the Interpretation of Discriminant Analysis," Journal of Marketing Research, Volume 6, May 1969, 156-163.

Mosteller, Frederick, "The Jackknife," Review of the International Statistical Institute, Volume 39, Number 3, September 1971, 363-368.

Myers, James H., and Forgy, Edward W., "The Development of Numerical Credit Evaluation Systems," Journal of the American Statistics Association, Volume 58, Number 303, September 1963, 799-806.

Nau, Dana, and Gray, Michael,: "Hierarchical Knowledge Clustering: A Way to Represent and Use Problem-solving Knowledge," in Expert Systems: The User Interface, edited by James A. Hendler, New Jersey: Ablex Publishing Co., 1988, 81-98.

Negoita, Constantin Virgil,: "Expert Systems and Fuzzy Systems," California: The Benjamin/Cummings Publishing Company,Inc., 1984.

NeuroShell, Frederick, MD: Ward Systems Group, Inc., 1988.

Nicholson, Walter, "Microeconomic Theory," Hinsdale, Illinois: The Dryden Press, 1978.

Nie, Norman H., Hull C. Hadlai, Jenkins, Jean G., Steinbrenner, Karin, and Bent, Dale H., "SPSS: Statistical Package for the Social Sciences," Second edition, New York: McGraw-Hill Book Company, 1975.

Nilsson, Nils J., "Learning Machines," New York: McGraw-Hill Book Company, 1965.

Omidvar, Massoud Omid, "Analysis of Neuronal Learning Models and Their Application in Computer System Design," Ph.D. Dissertation, Norman, Oklahoma: Graduate College, The University of Oklahoma, 1987.

O'Reilly, Brian, "Computers That Think Like People," Fortune, Volume 119, February, 1989, 90-93.

Parkkinen, Jussi, Oja, Erkki, and Jaaskelainen, Timo, "Color Analysis by Learning Subspaces and Optical Processing," IEEE International Conference on Neural Networks, San Diego, California, 1988, II-421.

Parzen, E., "On Estimation of a Probability density Function and Mode," Annuals of Mathematical Statistics, Volume 33, 1962, 1065-1076.

Ramanujam, Vasudevan, Venkatraman, N., and Camillus, John C., "Multi-Objective Assessment of Effectiveness of Strategic Planning: A Discriminant Analysis Approach," Academy of Management Journal, Volume 29, Number 2, 1986, 347-372.

Rich, Elaine, "Artificial Intelligence," New York: McGraw-Hill Book Company, 1983.

Rosenblatt, Frank, "Principles of Neurodynamics: Perceptions and the Theory of Brain Mechanisms," Washington D.C.: Spartan Books, 1962.

Rudd, Andrew, and Clasing, Henry K. Jr., "Modern Portfolio Theory : The Principles of Investment Management," Homewood, Illinois: Dow Jones-Irwin, 1982, p11.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J., "Learning Internal Representations by Error Propagation," in "Parallel Distributed Processing : Explorations in the Microstructure of Cognition, Volume 1: Foundations," D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, Eds., Cambridge, Massachusetts: A Bradford Book, The MIT Press, 1986, 318-362.

Rumelhart, David E., McClelland, James L., and the PDP Research Group, "Parallel Distributed Processing : Explorations in the Microstructure of Cognition, Volume 1: Foundations," Cambridge, Massachusetts: A Bradford Book, The MIT Press, 1986.

Sammon, J. W. Jr., "A Nonlinear Mapping for Data Structure Analysis," IEEE Transactions on Computers, Volume c-18, May 1969, 401-409.

Scheuer, Ernest M., and Stoller, David S., "On the Generation of Normal Random Vectors," Technometrics, Volume 4, 1962, 278-281.

Shafer, Glenn, "A Mathematical Theory of Evidence," Princeton: Princeton University Press, 1976.

Shiue, L. C., and Grondin R. O., "On Designing Fuzzy Learning Neural-Automata," IEEE First International Conference on Neural Networks, San Diego, California, 1987, II-299-307.

Sietsma, J., and Dow, R. J. F., "Neural Net Pruning - Why and How," IEEE International Conference on Neural Networks, San Diego, California, 1988, I-325 - 333.

Soulie, F. Fogelman, Gallinari, P., Cun, Y. Le, and Thiria, S., "Evaluation of Network Architectures on Test Learning Tasks," IEEE First International Conference on Neural Networks, San Diego, California, 1987, II-653 - 660.

Specht, Donald F., "Probabilistic Neural Networks," Neural Networks, Volume 3, Number 1, 1990a, 109-118.

Specht, Donald F., "Probabilistic Neural Networks and the Polynomial Adaline as Complementary Techniques for Classification," IEEE Transactions on Neural Networks, Volume 1, Number 1, March 1990b, 111-121.

Sprecher, David A., "On the Structure of Continuous Functions of Several Variables," Transactions of The American Mathematics Society, Volume 115, March 1965, 340-355.

Sugeno, M., "Theory of Fuzzy Integral and Its Applications," Ph.D. Thesis, Tokyo: Tokyo Institution of Technology, 1974. (Source: [Dubois and Prade 1980]).

Sugeno, M., "Fuzzy Measures and Fuzzy Integrals: A Survey," in "Fuzzy Automata and Decision Processes," M.M.Gupta, G.N.Saridis, and B.R.Gaines, Eds., Amsterdam: North-Holland Publ., 1977, 89-102.

Tatsuoka, Maurice M., "Multivariate Analysis : Techniques for Educational and Psychological Research," New York: John Wiley & Sons, Inc. 1971.

Toussaint, Godfried T., "Bibliography on Estimation of Misclassification," IEEE Transactions on Information Theory, Volume IT-20, Number 4, July 1974, 472-479.

Utgoff, Paul E., "Shift of Bias for Inductive Concept Learning," in "Machine Learning: An Artificial Intelligence Approach," Volume II, R.S. Michalski, J.G.Carbonell, and T.M.Mitchell, Eds., Los Altos, California: Morgan Kaufman Publishers, Inc., 1986, Chapter 5, 107-148.

Wald, A., "Sequential Analysis," New York: John Wiley & Son, 1947.

Widrow, G., and Hoff, M. E., "Adaptive Switching Circuits," Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record, Part 4, 1960, 96-104.

Widrow, Bernard, Winter, Rodney G., and Baxter, Robert A., "Learning Phenomena in Layered Neural Networks," IEEE First International Conference on Neural Networks, San Diego, California, 1987, II-411 - 429.

Widrow, Bernard, Winter, Rodney G., and Baxter, Robert A., "Layered Neural Nets for Pattern Recognition," IEEE Transactions on Acoustics, Speech, and Signal Processing, Volume 36, Number 7, July 1988, 1109-1117.

Wieland, Alexis, and Leighton, Russell, "Geometric Analysis of Neural Network Capabilities," IEEE First International Conference on Neural Networks, San Diego, California, 1987, III-385 - 392.

Williams, R. J., "The Logic of Activation Function," in "Parallel Distributed Processing : Explorations in the Microstructure of Cognition, Volume 1: Foundations," D.E.Rumelhart, J.L.McClelland, and the PDP Research Group, Eds., Cambridge, Massachusetts: A Bradford Book, The MIT Press, 1986, Chapter 10, p423.

Wolfgram, Deborah D., Dear, Teresa, J., and Galbraith, Craig S., "Expert Systems for the Technical Professional," New York: John Wiley & Sons, 1987.

Young, Tzay Y., and Calvert, Thomas W., "Classification, Estimation and Pattern Recognition," New York: American Elsevier Publishing Company, Inc., 1974.

Zadeh, L. A., "Fuzzy Sets," Information And Control, Volume 8, 1965, 338-353.

Zadeh, L. A., "Probability Measures of Fuzzy Events," Journal of Mathematical Analysis And Applications, Volume 23, August 1968, 421-427.

Zadeh, L. A., "Fuzzy Sets As A Basis for A Theory of Possibility," International Journal of Fuzzy Sets And Systems, Volume 1, Number 1, 1978, 3-28.

Zadeh, L. A., "Fuzzy Sets and Commonsense Knowledge," Berkeley, California: Berkeley Cognitive Science Report No.21, 1984.

Zaltman, Gerald, and Burger, Philip C., "Marketing Research: Fundamentals and Dynamics," Hinsdale, Illinois: The Dryden Press, 1975.

Zimmermann, H. J., Zadeh, L. A., and Gaines, B. R., "Fuzzy Sets and Decision Analysis," New York: North-Holland, 1984.

********** E N D **********