Design and analysis of discriminant pattern classifiers

This item was submitted to Loughborough University's Institutional Repository by the/an author.

Additional Information:

• A Doctoral Thesis. Submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of Loughborough University.

Metadata Record: [https://dspace.lboro.ac.uk/2134/14143](https://dspace.lboro.ac.uk/2134/14143)

Publisher: © S. Semnani

Please cite the published version.
This item was submitted to Loughborough University as a PhD thesis by the author and is made available in the Institutional Repository (https://dspace.lboro.ac.uk/) under the following Creative Commons Licence conditions.

For the full text of this licence, please go to: http://creativecommons.org/licenses/by-nc-nd/2.5/
<table>
<thead>
<tr>
<th>AUTHOR/FILING TITLE</th>
<th>SEMNALL, S.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESSION/COPY NO.</td>
<td>O3472799</td>
</tr>
<tr>
<td>VOL. NO.</td>
<td></td>
</tr>
<tr>
<td>CLASS MARK</td>
<td></td>
</tr>
<tr>
<td>1 JUL 1994</td>
<td></td>
</tr>
<tr>
<td>30 JUN 1995</td>
<td></td>
</tr>
<tr>
<td>06 SEP 1994</td>
<td></td>
</tr>
<tr>
<td>17/2/95</td>
<td></td>
</tr>
<tr>
<td>30 JUN 1995</td>
<td></td>
</tr>
<tr>
<td>28 JUN 1996</td>
<td></td>
</tr>
</tbody>
</table>
Design and Analysis of Discriminant Pattern Classifiers

by

Shahram Semnani

A doctoral thesis submitted in partial fulfilment of the
requirements for the award of
Doctor of Philosophy of the Loughborough University of Technology

Supervisor: Dr. M.J.J. Holt

March 1993

© S. Semnani
Dedication

This thesis is dedicated to my mother and late father for all their love and encouragement over the years it was they who made it possible.
Abstract

In recent years pattern recognition has evolved to a mature discipline and has been successfully applied to various problems. A fundamental part of an automatic pattern recognition system is classification, where a pattern vector is assigned to one of a finite number of classes. This thesis reports on the development and design of pattern classifier algorithms, with particular emphasis on statistical algorithms which employ discriminant functions.

The first part of this research work investigates the use of linear discriminant functions as pattern classifiers. A comparison of some well known methods, including Perceptron, Widrow-Hoff and Ho-Kashyap, is presented.

Using generalised linear modelling a new method of training discriminant functions is developed. In this method the linear discriminant function is transformed by a non-linear link function which associates with each pattern vector a measure which is bounded in the range of 0 to 1 according to the class membership of the pattern. In simulations the GLM approach is applied both to synthetic data and to experimental data from a binary pattern matching problem. It is seen that GLM exhibits faster and more reliable convergence than existing linear discriminant approaches.

Extensions of this method to Piecewise linear discriminant functions and to polynomial discriminant functions are explored. Application of self-organising methods for efficient generation of polynomial discriminant functions is also investigated.

In the second part of the work a review of neural networks is presented, followed by an analysis and formulation of a popular neural network training algorithm, namely Back-propagation (BP). The capabilities and deficiencies of BP and its variations are experimentally evaluated by computer simulations.

An alternative formulation based on Empirical Maximum Likelihood (EML) is also proposed. This approach is shown to have a simpler error landscape in comparison to the original BP based on mean square error. Simulations show that the EML approach generally provides faster convergence, involves fewer calculations per iteration than conventional BP, and results in equally good classification performance.
Acknowledgement

I would like to express my sincere gratitude to Dr. Murray Holt for his help, guidance and encouragement throughout this research work. I am grateful to Dr. S. Datta for his suggestions and interest.

Special thanks to all my friends and colleagues in the Speech and Image processing group for their willingness to help.

Finally and most importantly, grateful thanks are due to my wife, Fatemeh for her help, support, patience and understanding when I needed it most.

S. Semnani
Chapter 1

Introduction

1.1 Introduction

As computers and electronic machines enter our lives in ever increasing numbers, the need for better and faster communication between human and machines becomes more evident. The need for machines that can be commanded by voice, machines that can read and print handwriting, machines that translate from one language to another and machines that can aid diagnosis of diseases are ever increasing. As machines grow in numbers and increase in speed their requirements for input data see no bounds. Neither does there seem to be a limit on the quantity of information that machines generate that humans must digest, comprehend and on which they must base their decision.

Advances of modern science and technology has produced solutions to more and more complex problems. By and large these tasks are realised as long as there exist a specified set of instructions, stating how the task must be performed. However, there are problems of long standing interest that have eluded solution because of lack of complete theory and understanding of the problem.

Recognition of a spoken word independent of the speaker, recognition of a writer
regardless of the written text and the problem of distinguishing the sound of one musical instrument from that of another are only a few of the problems that have so far remained largely unsolved for the reason mentioned above. And yet all these problems are solved with ease by humans in their daily life. The ever expanding field of pattern recognition concerns itself with the solution of these problems.

By its very nature, pattern recognition is a very broad field of activity with very fuzzy boundaries. However it is generally defined as a process of decision making in which a new input is recognised as a member of a given class by a comparison of its attributes with the already known pattern of common attributes[1].

1.2 Pattern Classification

A pattern recognition system can be viewed as an automatic decision rule, it transforms measurements from a pattern known as features into class assignments. Those pattern classified into the same pattern class usually have the same common properties. A pattern classifier is at the front end of a pattern recognition system, where it performs the task of decision making. Pattern classification methods are employed to partition the feature space in an optimal manner so to minimise the expected probability of misclassifications[2].

One of the first classification methods, the Perceptron[3] was based on the behaviour of the central nervous system in which the individual neurons in the Perceptron were modelled as a threshold logic unit and by means of an adaptive mechanism producing linear classification. When the Perceptron was still in its very infancy, it was realised that statistical and probability theory had much to offer to the pattern classification. The attraction of these approaches were mainly due to sound mathematical modelling of the classifier. This led to the development of numerous classification techniques such as Least Mean Square of Widrow and Hoff[4], Bayes classifier[5], Fishers discriminant function[6], Generalised inverse of Wee[7] and Ho and Kashyap solution to linear inequalities[8].

Effective though they were, these approaches had built-in limitations. Therefore some other approaches had been proposed. For instance linearity of these approaches posed a sever limitation to the performance of such classifiers. Naturally non-linear approaches to classification which allowed for generation of complex decision boundaries were suggested[9].
Nilsson's Learning Machines[10] had a system architecture consisting of layers of Perceptron. Theoretically this system was capable of producing highly complex decision boundaries. However these types of approaches were partially successful. Their major drawback was due to inefficient adaptive mechanism of generating decision boundaries.

An Artificial Neural Network proposed by Rumelhart, Hinton and Williams[11] overcame this inefficiency. Like the Perceptron, the Neural Network proposed by Rumelhart et al, was also based on the central nervous system. The discovery of the new adaptive mechanism for training such classifiers led to the rebirth of the Neural Network research and since its discovery it has found application in numerous research areas.

The scope of this thesis is to design and develop highly efficient pattern classification algorithms with particular emphasis on statistical approaches. The common characteristic considered in designing such classifiers is the application of a technique known as supervised learning[12]. In supervised learning a data set known as training data (containing pattern measurements and their correct classification) is used to teach the classifier producing decision boundaries so to minimise the expected probability of misclassification.

Initially Linear Discriminant Functions (LDF)[13] as pattern classifiers are examined. A comparison of some well known methods is made. Further more a new method of LDF based on Generalised Linear Modelling (GLM)[14] is developed and the performance of the resulting LDF is compared to that obtained from conventional methods. Piecewise Linear Function and Polynomial Discriminant Function are introduced as possible extensions to LDF. Although these are natural extensions of LDF and suggested improvements to some of the weakness of LDF, consideration is given to a more powerful and effective way of designing classifiers through Neural Networks.

The capabilities and deficiencies of a popular Neural Network classifier, namely the Backpropagation (BP), are evaluated by computer simulations. An alternative formulation based on Empirical Maximum Likelihood (EML) is proposed and its performance/complexity is compared to that obtained from the conventional BP.

The comparison of these classification methods can help the classifier designer to determine which algorithm would better suit the complexity/performance requirements of a particular application.
1.3 Thesis Overview

In Chapter 2 some fundamental issues in Pattern Recognition are addressed. Two main approaches to pattern classification, namely the syntactic and statistical approach, are introduced and their suitability for pattern classifications is discussed. This is followed by a discussion of various design strategies for the statistical approach including parametric verses non-parametric and supervised verses non-supervised methods. Also presented is the development and analysis of some well known pattern classification methods, namely the Perceptron[3], Widrow-Hoff[4], and Ho-Kashyap[8] algorithms.

In Chapter 3 a new method of pattern classification based on Generalised Linear Modelling[14] is proposed. The formulation and description of the GLM method is presented through an introduction to linear regression methods. A number of computer simulations using synthetic and experimental data were carried out to compare the performance of GLM with that obtained from conventional LDFs. The convergence rate and misclassification percentage of GLM is proved experimentally, to be superior than that of LDF types described in Chapter 2.

Possible extensions to LDF which attempt to overcome some of the drawbacks of LDF are examined in Chapter 4. These extensions generate non-linear decision boundaries, and are divided into two categories. In the first category LDFs are extended to include second order terms hence producing quadratic decision boundaries. It is shown that high order Discriminant Function can be generated by Group Method of Data handling (GMDH). This method is only suggested for its theoretical interest and is usually suited to very complex problems.

The philosophy behind the second category is to use the simple principles of LDF as a basic building block to produce a multilayer architecture generating a non-linear (Piecewise) classifier. These types of classifiers were considered to be inefficient[11], until the recent discovery of a training method in neural networks[11]. The neural network training methods are examined in more detail in Chapter 5 and 6.

Chapter 5 starts with a brief background to neural networks and their recent rebirth. The great interest surrounding neural network research is mainly due to the discovery of a powerful technique for training multilayer networks. The formulation of the well known Backpropagation (BP)[11] training algorithm as applied to such networks is
described. This is followed by a discussion on the generalisation capability of such networks and of the design issues involved in developing an optimal pattern classifier network.

Since the discovery of BP many variations have been proposed by researchers. Some of these variations that attempt to improve the BP performance are also described in Chapter 5.

Computer simulations of multilayer networks trained with BP illustrate the effectiveness of this type of classifier for pattern recognition problems which are difficult or impossible to solve using conventional LDF methods.

A more rigorous study of the BP learning and generalisation issues is carried out in Chapter 6, where these performances are experimentally compared with an alternative training method. The proposed training method is based on Empirical Maximum Likelihood (EML). The principle of this approach lies on reformulation of BP in terms of probabilities and the introduction of an alternative cost function.

Using computer simulations various properties of EML are assessed, including speed of training, generalisation capability, sensitivity to initial weights, and the shape of error surfaces of the two training methods. These simulations are carried out on both synthetic and experimental data. It is demonstrated that EML outperforms the conventional BP in all the simulations without any increase in computational complexity and cost.

Finally, Chapter 7 provides general conclusions relating to selection and design of supervised pattern classifiers to suit the performance/complexity requirement of a particular problem. Suggestions are made for further research arising from the work presented in this thesis.
References

Chapter 2

Methodologies in Pattern Recognition

2.1 Introduction

The basic function of pattern recognition (PR) is to assign a physical object or an event to one of several prespecified categories. The patterns themselves could range from characters obtained from a scanned document to speech waveforms of utterances from an individual, the associated recognition problems are to label a character as being a specific letter of an alphanumeric set or to identify the spoken word.

The field of pattern recognition has grown enormously in recent years and a wide variety of techniques have been developed for various applications. Traditionally these methods are classified into deterministic and syntactic methods.

The fundamental idea in the syntactic approach is the utilisation of structural information. The recognition is done by matching these structural descriptions. In this approach to PR, first the patterns are described in terms of simpler sub-patterns
(sub-structures) known as primitives. For instance in character recognition typical primitives include straight line segments, curves, loops and branches. Primitives are then appropriately labelled, and a relationship between the primitives is formulated in terms of a predefined linguistic grammar. Finally the recognition process is achieved by a searching method which matches the formulation to the prototype pattern description.

Syntactic methods are often effective when the patterns under consideration are quite complex or when the structural information contained in a pattern is important for the recognition process[1]. However, syntactic methods suffer from two major disadvantages. Firstly, identifying pattern primitives necessitates complex algorithms such as curve fitting and polygonal approximation[2] which require a significant computational effort. The other drawback is that the recognition of noisy patterns has not yet been solved[3].

Despite these disadvantages, syntactic methods have been successfully applied to many pattern recognition problems including 3-D shape recognition[4].

In the deterministic approach to PR, patterns under consideration are represented by a set of measurable quantities known as features. As an illustration, for character recognition[5], these measurements may include height, width, area and perimeter of a character. The recognition task can be accomplished by comparing features of an unknown pattern with the corresponding features of the prototype patterns. Various statistical decision theory techniques have been developed to perform this process.

This chapter surveys deterministic methods and examines the capability and performance of some well known algorithms through computer simulations.

2.2 Statistical Approach

Statistical approaches have a long tradition in pattern recognition and are based on well founded mathematical theory that have been proven useful in numerous applications. The algorithms are usually computationally inexpensive when compared with syntactic (structural) methods[6].
A simplified block diagram for a statistical PR system is shown in Fig.(2.1). In order to recognise a pattern it is necessary to obtain through appropriate sensory channels (for example TV camera, scanner, analogue to digital converter) information from the environment. This implies making some form of measurements in order to define the pattern data. A pattern is defined by a pattern vector, and a pattern vector is represented by a point in an N-dimensional pattern space whose co-ordinates correspond to measurable information of the pattern. The dimensionality of a pattern vector is typically quite large. Hence to design an economical PR system, it is desirable to reduce the dimensionality of these vectors by suitable selection of important attributes (or features) from a pattern vector. This process is carried out by the feature selector shown in Fig.(2.1), whose output information is known as a feature vector. This vector is passed through a classifier whose purpose is to categorise or make decision.

Based upon the information presented by a feature vector the designer of a statistical PR system is generally faced with two major problems. The designing problems of such a system are briefly described here.

The first problem concerns the determination of important attributes through the feature selector. Normally it is hoped that the selected features are such that patterns belonging to different categories will occupy different regions in the feature space. In most PR problems the determination of a complete set of discriminatory features is extremely difficult if not impossible. Although a great deal of effort has been undertaken in deriving optimal algorithms with respect to classifying the selected features, the features themselves are often chosen arbitrarily.

The second problem is the determination of the decision procedures which are needed in the classification process. The topic of classification theory in the context of mathematical PR is one which has evolved to a higher stage of refinement than that of feature selection and has found numerous applications in various fields of science eg. communications, medicine, psychology, etc.

2.2.1 Selection of Classifier

Designing a classifier in statistical PR depends on what kind of information is available about the distributions and densities of feature vectors. If it is assumed that
Fig. (2.1) Block diagram of simplified Pattern Recognition System
the form of the densities is known but some of the parameters of the densities are unknown then the problem is said to be a 'parametric' decision process. A common strategy for this kind of problem is to replace the unknown parameters in the density functions by their estimated values[7].

If the form of the densities is not known then the problem is said to be a 'non-parametric' decision process. In this case either the density function is estimated (eg using Parzen window[8]) or some form of non-parametric decision rule (eg discriminant functions) is applied to the problem. The estimation of parameters or the density function requires the availability of training samples.

Another distinction in statistical PR is between supervised learning (labelled training samples) and unsupervised learning (unlabelled training samples). The various subsections which appear in statistical PR are shown in the tree structure of Fig(2.2). In supervised learning the label on each pattern represents the category to which that pattern belongs.

Unsupervised learning refers to situations where samples are not labelled. This is a difficult problem and in practice one often relies on techniques of clustering analysis[9] to identify natural groupings (classes) presented in the data.

This approach in PR is particularly attractive in situations where the number of pattern classes are not precisely known. Cluster analysis attempts to assess the interaction among patterns by organising them into groups or clusters. A large number of clustering algorithms[10] have been developed and implemented. But there is no best clustering algorithm since the performance of a clustering algorithm is dependent on the nature of the data under consideration. This points to the difficulty of interpreting the results of a clustering algorithm, particularly in high dimensions[11]. Only a few statistical tests are available to assess cluster tendency and cluster validity in particular situations[12].

2.2.2 Parametric Versus Non-Parametric Approach in Supervised Learning

Most of the popular parametric techniques are optimal if the class densities happen to be Gaussian (Normal)[13].
Fig. (2.2) Breakdown of classifiers in a Pattern Recognition system
Even though statistics based on the Gaussian distribution are robust, it is wise to consider non-parametric techniques such as discriminant functions, if the data substantially departs from normality.

As an example visual assessment of feature distribution in patterns from the template matching problem had shown that they have a non-Gaussian type of distribution. Therefore selecting a parametric classifier could have a fatal effect on the overall recognition process.

Therefore, throughout this research work, due to the availability of labelled training data with an apparently non-Gaussian distribution, attention has been focused on supervised, non-parametric classifiers. Some classifiers of this type will be described.

2.3 Discriminant Functions

Discriminant functions are extensively used in the field of PR[14-17]. In using discriminant functions as pattern classifiers, it is assumed that the formulation of the functions are known, and the training samples are used to estimate only the parameters of the classifier. Linear discriminant functions (LDFs) are relatively easy to compute and they have a fixed structure which makes them attractive for implementation in special purpose computers.

Throughout this chapter, LDFs and various procedures of implementing these functions as classifiers will be examined.

2.3.1 LDF Classifier

A typical objective when using these functions is to classify a pattern, which is represented by a vector of N features. For the sake of simplicity and presentation the two-class case is considered. However, the techniques can easily be extended to multi-class problems.

Let us suppose that N is the number of features to be used in making a classification
of a pattern \( y_i \), \( y_i = \{y_1, y_2, \ldots, y_N\} \), where \( y_i \) is a \( N \)-dimensional point in feature space. A training set will consist of \( n \) such patterns \( y_i \), which will be used by the classifier to identify the two classes \( C_1 \) and \( C_2 \). A typical LDF which is a linear combination of the elements of \( y_i \) can be written as:

\[
d(y_i) = w_0 + \sum_{j=1}^{N} w_j y_j
\]

where

\[
w = (w_0, w_1, \ldots, w_N)
\]

\( w_0 \) = Threshold weight

\( y_i = \{y_1, y_2, \ldots, y_N\} \)

Feature vector

The common convention is to augment the feature vector by an additional constant unit-valued element, giving:

\[
y_i = \{1, y_1, y_2, \ldots, y_N\}
\]

so that equation (2.1) can be expressed in the form (dropping the subscripts):

\[
d(y) = w^T y
\]

If the patterns from different classes are linearly separable, (can be separated by a hyperplane in the feature space) as shown in Fig.(2.3) then, with correct values of the weights, in equation (2.2), a perfect recognition is possible to be achieved. Using the above assumption, the LDF classifier is designed to have the capability of estimating the best values of the weights (the weights which yield perfect recognition) from the training patterns. The basic idea is that by observing the input patterns with known classifications, the classifier can automatically adjust the weights in order to achieve best recognition rate. Performance of the classifier is suppose to improve as more and more patterns are applied. This process is known as training or learning.
Chapter 2: Methodologies in Pattern Recognition

Fig. (2.3) Example of linearly separable data set

Fig. (2.4) Example of Non-separable data set
Hence the main aim of LDF is to use the training set to determine the unknown weight vector in the LDF expression (2.2), such that:

\[
\begin{align*}
\text{d}(y) &= w^T y \\
&\begin{cases}
> 0 & \text{if } y \in C_1 \\
< 0 & \text{if } y \in C_2
\end{cases}
\end{align*}
\]

The equation \( d(y) = 0 \) defines a decision boundary which is a line for the two feature case \((N = 2)\) and is a plane or hyperplane for higher values of \( N \).

A sample \( y_i \) is classified correctly if \( w^T y_i > 0 \) and also \( y_i \) is labelled as \( C_1 \), or if \( w^T y_i < 0 \) and \( y_i \) is labelled as \( C_2 \).

In the latter case it can be seen that \( y_i \) is classified if \( w^T(-y_i) > 0 \). Hence a normalisation that simplifies the treatment of the two-category case is achieved by simple replacement of all samples labelled \( C_2 \) by their negative. With this normalisation one can forget the labels and look for a weight vector such that \( w^T y > 0 \) for all the samples.

From the above definition of LDF it can be seen that the problem of finding the unknown weight vectors is equivalent to solving a set of linear inequalities. If (from priori knowledge) there exists a solution which correctly classifies all the patterns in the training set, then the training set is said to be linearly separable. Many methods exist which guarantee a solution for separable cases.

However in most practical PR problems, due to various source of noise and distortions, often there exists a substantial amount of variability among the patterns belonging to the same class. This will cause the feature vectors from different classes to overlap, resulting in a non-separable data set shown in Fig.(2.4).

In such cases the methods which are based on the assumption of separability fail to produce a solution[18]. More complex algorithms exist which provide optimal or near-optimal LDF solutions to non-separable problems.
In the next section various popular LDF methods are reviewed. The performance and capability of these procedures are validated by means of computer simulations which have been applied to examples from the literature and also to the binary template matching problem[19-21].

2.4 The Perceptron Concept

One of the early significant results in the training of linear classifiers is Rosenblatt's Perceptron theorem[18]. This was developed at Cornell University and is based on a brain model. The basic block diagram of a Perceptron classifier is shown in Fig.(2.5).

The response of a model shown in Fig.(2.5) is a weighted sum of its input features followed by a hard limiting threshold element. If the weighted sum of the inputs exceeds a threshold then the output of the threshold element becomes +1 otherwise it will be -1. The output of the Perceptron is given by:

\[ P = f \left( \sum_{i=0}^{N} w_i y_i \right) = f(w^T y) \]  \hspace{1cm} (2.3)

where \( f \) is the threshold function. If \( P > 0 \) then the observed pattern belongs to \( C_1 \) and is \( P < 0 \) it belongs to \( C_2 \) (in order to be consistent with the original derivation of Perceptron, the input features are not augmented here). This description agrees with the concept of LDFs described earlier and as it can be seen the Perceptron is nothing more than an implementation of a linear discriminant function.

The Perceptron training algorithm involves adjusting the weights so that the decision surface produces an acceptable separation of the classes. This is stated as follows:

The training starts by a random assignment of the initial weight vector \( w_{(1)} \). At the \( k \)th training step the current value of \( w \) is updated according to: 
Chapter 2: Methodologies in Pattern Recognition

Fig. (2.5) The Perceptron
\[ w_{(k+1)} = \begin{cases} 
  w_{(k)} + \rho y_{(k)} & \text{if } y_{(k)} \in C_1 \text{ and } w^T_{(k)} y_{(k)} \leq 0 \\
  w_{(k)} - \rho y_{(k)} & \text{if } y_{(k)} \in C_2 \text{ and } w^T_{(k)} y_{(k)} \geq 0 \\
  w_{(k)} & \text{otherwise} 
\] 

Where \( \rho \) is a positive constant. The above rule updates \( w \) if and only if the pattern being considered at the \( k^{th} \) iteration (training step) is misclassified by the weight vector at this step.

According to the discussion in Section(2.4), it is possible to express the Perceptron algorithm in an equivalent form by multiplying the augmented vector of one class by -1. Thus the above expression can be written as:

\[ w_{(k+1)} = \begin{cases} 
  w_{(k)} + \rho y_{(k)} & \text{if } w^T_{(k)} y_{(k)} \leq 0 \\
  w_{(k)} & \text{otherwise} 
\] 

This equivalent formulation is used through the chapter unless otherwise stated.

This procedure is sometimes referred to as a 'reward and punishment' or 'error correcting'[22] procedure. That is, if a pattern is classified correctly the machine is 'rewarded' by the fact that no change is made in \( w \). On the other hand if a pattern is misclassified (in error) the machine is punished (corrected) by updating \( w \) by an appropriate amount.

The Perceptron algorithm is guaranteed to converge only if the training class are linearly separable[23].

Heuristic variations to the Perceptron algorithm have been suggested[24,25], where researchers have attempted to obtain acceptable performance on non-separable problems while maintaining the ability to find a separating vector for separable
A simple variation of the Perceptron method can be achieved by changing $\rho_k$ ($\rho_k = \rho/k$) at each iteration (k denotes the iteration step). The purpose of this variation is to decrease the effect of misclassified samples as the time progresses and hope for a stable solution for non-separable cases.

A more recent improvement to the Perceptron algorithm is due to Keller[24]. Keller has incorporated a fuzzy membership function[26] into the Perceptron procedure in order to stabilise erratic behaviour of the algorithm for non-separable cases.

Many other variations to the original Perceptron algorithm exists[8,22]. However some of these reported improvements sacrifice the guaranteed solution for the separable problem for obtaining stable solution for non-separable case, and almost all variations introduce a computational overhead[8,22]. Hence variations of Perceptron do not appear to offer an attractive solution to non-separable problems which arise in practice, since they either oscillate as far as they are allowed to execute or even if they produce a solution, the solution may not be optimal[27].

### 2.5 Least Mean Square Algorithm

In the Perceptron case the intention was to find a vector $w$ such that the inequality $w^Ty > 0$ is satisfied for all the samples in the training case. However in the LMS approach an attempt is made to make $w^Ty = b$, where $b$ is an arbitrary positive constant. Therefore the problem of finding a solution to a set of linear inequalities is transformed to one of finding a solution to a set of linear equations.

In this case the search is for a vector $w$ satisfying:

$$Yw = b \quad \cdots (2.4)$$
where

\[
Y = \begin{bmatrix}
y_1^T \\
y_2^T \\
\vdots \\
y_n^T
\end{bmatrix}
\]

\[
b^T = \{ b_1, b_2, \ldots, b_n \} \quad b_i \geq 0
\]

where \( Y \) is a matrix whose rows consist of the normalised training feature vectors. If \( Y \) is a non-singular matrix, then a unique solution exists in the form:

\[
w = Y^{-1} b
\]

But in almost all cases of PR the number of rows (training cases) greatly exceeds the number of columns (features), hence no exact solution exists. However it is possible to find a weight vector \( w \) such that the difference between \( Yw \) and a desired positive vector \( b \) is minimised. Hence the error vector is defined as:

\[
e = Yw - b
\]

One approach is to minimise the squared length of the error vector, which is equivalent to minimising the sum of squared errors[18], given by:

\[
E = \frac{1}{2} \sum_{i=1}^{n} (w^T y_i - b_i)^2 = \frac{1}{2} \| Yw - b \|_2^2
\]

where \( \| \cdot \| \) indicates the magnitude of the error vector. A well known approach to minimise expression (2.7) is indicated next.
2.5.1 The Widrow-Hoff Procedure

The Widrow-Hoff rule (W-H)[18] was developed in the early 1960’s and has found numerous applications in adaptive filter theory and adaptive pattern recognition[28,29]. Initially the W-H rule was implemented as an Adaptive Linear Neuron (ADALINE) to be used as a trainable pattern classifier. The Structure of such a classifier is shown in Fig.(2.6). The output response of such a structure is the weighted sum of input feature vectors. The weights of an ADALINE are updated by using an error signal between the output response and the actual response of the classifier.

Widrow and Hoff proposed an approach to minimise the sum of squared errors, equation (2.7), using the gradient descent procedure.

Generally this approach consists of incrementing $w$ in the direction of negative gradient of the error function, in order to seek the minimum of a function. In other words if $w(k)$ represents the value of $w$ at the $k^{th}$ training step, then the general gradient descent may be written as:

$$w(k + 1) = w(k) - \rho \left( \frac{\partial E}{\partial w} \right)$$

(2.8)

Where $\rho$ is a positive constant (usually $\rho < 1$) that controls the stability and the rate of convergence. It must be noted that no correction is made to $w$ when $(\partial E/\partial w) = 0$, which is the condition for a minimum.

Using the gradient descent equation (2.8) to minimise (2.7), a descent procedure can be written as:

$$w^{(1)} \quad \text{arbitrary}$$

$$w^{(k + 1)} = w^{(k)} - \rho Y^T(Yw - b)$$

(2.9)

Where in this case $b$ is a unity vector. This is known as the Widrow-Hoff rule[18].
Fig. (2.6) Structure of an LMS classifier based on the Widrow-Hoff rule
The computational task of processing and storing large matrices can be reduced by considering the samples sequentially. This is given by:

\[ w_{(1)} \text{ arbitrary} \]

\[ w_{(k+1)} = w_{(k)} - \rho (w^T_{(k)}y_{(k)} - b_{(k)}) y_{(k)} \]

As it can be seen from the above expression, W-H in contrast with the Perceptron updates the weights regardless of pattern classification. W-H rule adjusts the weights by gradient descent with the objective of reducing sum of squared errors. Both the W-H and Perceptron rules invoke similar training procedures. Because they are based on different objectives, however, they can have significantly different characteristics. This will be shown through computer simulations in later sections of this chapter.

2.6 The Ho-Kashyap Procedure

The LMS approach yields a weight vector both for separable and non-separable cases. However there is no guarantee that this vector is a separating vector in linearly separable cases[8].

Ho and Kashyap[20] proposed an iterative procedure which is a modified derivative of the LMS procedure, in which an improvement is made by an iterative adjustment of both \( w \) and \( b \). The underlying idea comes from the observation that if the samples are separable and if both \( w \) and \( b \) are allowed to vary in the error function:

\[ E = \frac{1}{2} || Yw - b ||^2 \] \hspace{1cm} (2.10)

Subject to the constraint that \( b > 0 \), then the minimum value of \( E \) is zero and the \( w \) that achieves that minimum is a separating vector[20].
2.6.1 Formulation of H-K Procedure

The gradient descent approach is used to minimise $E$. However in view of the fact that $E$ will be minimised with respect to both $w$ and $b$, a slight modification to the general gradient descent, equation (2.8), is required.

The gradients associated with this problem are:

$$\frac{\partial E}{\partial w} = Y^T (Yw - b) \quad \ldots \ldots (2.11)$$

and

$$\frac{\partial E}{\partial b} = -(Yw - b) \quad \ldots \ldots (2.12)$$

Since $w$ is not constrained in any way, the error gradient is brought to zero in one step as follows:

$$Y^T Yw = Y^T b$$

$$w = (Y^T Y)^{-1} Y^T b = Y^\dagger b \quad \ldots \ldots (2.13)$$

Where $Y^\dagger$ is known as generalised inverse of $Y$.

Since $b$ is constrained to be a positive vector, it must be varied in such a way that it never violates this constraint. One way of achieving this is to start with $b > 0$, and refuse to reduce any of its components. This is accomplished by:

$$b_{(k+1)} = b_{(k)} + \delta b_{(k)} \quad \ldots \ldots (2.14)$$

where a single component of $\delta b_{(k)}$ is given by:
Chapter 2: Methodologies in Pattern Recognition

\[ \delta b_i(k) = \begin{cases} \rho \left[ Yw(k) - b(k) \right]_i & \text{if } \left[ Yw(k) - b(k) \right]_i > 0 \\ 0 & \text{if } \left[ Yw(k) - b(k) \right]_i \leq 0 \end{cases} \]

\( \rho \) is a positive constant, \( i \) represents the \( i \)th component of \( b \) and \( k \) is the iteration index. The above expression can be written as:

\[ \delta b(k) = \frac{1}{2} \rho \left( Yw(k) - b(k) + \left| Yw(k) - b(k) \right| \right) \] \hspace{1cm} (2.15)

In this manner the components of \( b(k) \) are never reduced. The above expressions are a modification of the gradient descent procedure, and the iterative procedure defined by equations (2.13 - 2.15) is known as the Ho-Kashyap procedure. Hence the H-K procedure can be summarised as follows:

\[ \begin{align*}
    w(1) &= Y^\dagger b(1) \\
    b(0) &> 0 \\
    e(k) &= Yw(k) - b(k) \\
    b(k+1) &= b(k) + \frac{1}{2} \rho \left( e(k) + \left| e(k) \right| \right) \\
    w(k+1) &= Yb(k+1)
\end{align*} \] \hspace{1cm} (2.16)

From the above expression it can be seen that the initial weights to the H-K iterative procedure is obtained by using the generalised inverse of \( Y \) hence yielding a good starting point as compared with the random initial weights used in the other methods.
2.7 LDF Simulations

This section reports on software implementation of three linear classifiers and presents their relative performance on various types of data sets. The three classifiers simulated here are Perceptron (P), Widrow-Hoff (W-H) and Ho-Kashyap (H-K).

Four two-class data sets were used to evaluate the performance of these classifiers on the basis of speed of convergence and the ability to converge to an optimum solution.

Two separable and two non-separable data sets from the literature and data arising from binary template matching were used in these simulations. A brief description of each data set and the experimental results on LDFs are as follows:

1-Data Set (a)

This data set is obtained from Hand[30] who has made some studies on the performance of the Perceptron. The data set has been extracted from the experimental study of human social behaviour and contains a total of 26 patterns, each pattern has androsterone and eticoholanolone as the two components of the feature vector. This data set is linearly separable. The distribution of this data set and the decision boundaries obtained from the simulations is shown in graph(2.1).

For this data set none of the classifiers encountered any problem in producing the separating vector. The Perceptron classifier on average (over 10 different starting points) was the slowest in convergence; it produced zero error solution after 11 iterations in comparison with 8 for the W-H. Although H-K algorithm requires relatively more calculation per iteration, it converges to a separable solution only after a single iteration through the training data.

2-Data Set (b)

This data set was obtained from Keller, University of Missouri, who has studied and compared the performance of Perceptron with the fuzzy Perceptron[24]. The
Graph (2.1) Distribution of Hand’s Data and Decision Boundaries Generated by Various Simulations.
Graph (2.2) Distribution of Kellers's Data and Decision Boundaries
Generated by Various Simulations.
Graph (2.3) Graph of Misclassification against Number of Iterations for Keller's Data.

Graph (2.4) Graph of Misclassification against Number of Iterations for Keller's Data.
data set is generated from a multivariate normal distribution. This data set is non-separable and it contains a total of 242 samples, 121 per class.

The distribution of this data set and the decision boundaries obtained from simulations is shown in graph (2.2). Graph (2.3) and (2.4) illustrate the error rates against the number of iterations produced by simulations of the classifiers on this data set.

As shown in graph (2.2) the Perceptron has failed to produce a solution and it behaves erratically as the iterations progresses.

The W-H classifier produces 9% error rate after 100 iterations. A similar percentage of error rates 9.09% was produced by Keller’s fuzzy Perceptron, but the fuzzy Perceptron was reported to have converged to this error rate after twenty two iterations[24]. The superior performance of H-K over other classification techniques can be observed in graph (2.4). The H-K method converges to 7.43% error rate after 3 iterations.

3-Data Set (c)

This training data set is obtained from the easily measured feature of alphabetic characters arising in binary template matching. The measurements used in this data set are Combined Black Pixels and Weighted And-Not of the characters under consideration (the reader is referred to appendix A for further discussion on binary template matching). The total number of training samples in this data set is 600 which were obtained from a single document and they are linearly separable.

The distribution of this data set and the decision boundaries obtained from simulations is shown in graph (2.5). Graph (2.6) and graph (2.7) show the error rate curves (%misclassified samples) against the number of iterations. From graph (2.6) it can be seen that the Perceptron method converges to a separable solution after 370 iterations. The W-H method converges to a solution but it fails to produce a separable solution. This is because the W-H method is based on sum of squared
Graph (2.5) Distribution of Template Matching Data and Decision Boundaries Generated by Various Simulations.
Graph (2.6) Graph of Misclassification against Number of Iterations for Experiment Case c.

Graph (2.7) Graph of Misclassification against Number of Iterations for Experiment Case c.
errors and the correction to the weights are made, so the method seeks a solution minimising this sum, but this minimum point does not necessary correspond to the minimum misclassified error rate. For the case of Perceptron, the method is based on the error correction, that is the weights are updated only at the occurrence of misclassification. If an error free solution exists the method guarantees to find it after a finite number of iterations.

The H-K algorithm converges to zero error misclassification at 178 iterations which is significantly faster than the Perceptron.

4- Data Set (d)

This data set is obtained in the same manner as described above but the characters are from three different documents. This data set contains 2000 data points and because of character variability it forms a non separable data set.

Graph(2.8) and (2.9) show the error rate curves against the number of iterations for this data set. From this graph it can be seen that the Perceptron exhibits erratic behaviour and fails to reach a steady state solution. This is because the data is non-separable and the success of the Perceptron, being an error correcting method, is largely due to relentless search for a error free solution. For this case the Perceptron never ceases correction, since there is no weight vector which would correctly classify all the samples.

Although W-H failed to produce a separable solution for the previous simulation, for this non-separable data set it produces 14% error rate after 52 iterations and it does not exhibit any erratic behaviour as was shown by the Perceptron. This is illustrated in graph (2.8). The W-H procedure yields a solution irrespective of sample distribution but this is at the cost of losing the guarantee for consistently converging to a separating solution for the separable data sets.

The H-K algorithm produces an error rate of 4.2% at 100 iterations in comparison with 14% for the W-H method. However it requires a large number of iterations to reach this near optimum solution.

Comparing the classification methods simulated so far, the Perceptron was found to be an attractive method because of its relative simplicity of implementation, although
Graph (2.8) Graph of Misclassification against Number of Iterations for Experiment Case d.

Graph (2.9) Graph of Misclassification against Number of Iterations for Experiment Case d.
it lacks a fast convergence rate and also its performance degrades for non-separable cases.

On the other hand the W-H algorithm, being based on minimisation of sum of squared errors (MSSE), offers a solution for both separable and non-separable data sets. But this is achieved by loosing the guarantee to a error free solution in the separable cases.

The H-K also being based on the variation of MSSE not only converges to a zero-error solution for the separable cases, but also it provides a good solution for the non-separable problems with a relatively fast speed of convergence. The fast speed of convergence is due to the fact that adjustments towards the minimum are made with respect to both w and b, and take place at each iteration. Of course the price paid for this feature is the added complexity of the H-K method. However the only apparent disadvantage in complexity in applying the H-K algorithm is the inversion of the \((Y^TY)\) matrix. Unless the patterns are of very high dimensionality, however, this does not present a serious difficulty since the matrix needs to be inverted only once per problem. In addition the inverse matrix \((Y^TY)^{-1}\) can be recursively updated as new rows (ie patterns or data samples) are added to Y[31].

All the techniques which were reported are linear functions of input data. Although these algorithms are not exhaustive, they do represent the spectrum of available deterministic linear classification techniques. The main advantage of these is their relative simplicity. On the other hand their linearity severely limits their ability to recognise and separate classes when they are non-linearly separable (that is they can be separated by a curve or Piecewise lines, but not linearly separable).

### 2.8 Conclusions

In this chapter some of the methodologies in pattern recognition have been introduced. Generally the syntactic methods are used when the pattern under consideration are quite complex or the structural information contained in a pattern are important in the recognition process. Although these methods have been applied successfully to various problems in PR, the main disadvantages are in high computational cost and poor recognition in the presence of noise.
On the other hand statistical methods are based on well formulated mathematical theory and they are computationally inexpensive.

Statistical methods are fundamentally divided into two categories of supervised and non-supervised methods. For the cases where the distribution of the patterns are unknown and difficult to estimate LDFs can provide a relatively simple and computationally efficient solution to pattern classification.

The performance of three well known LDFs (Perceptron, Widrow-Hoff and Ho-Kashyap) over a number of training sets were presented. The results indicate that the Perceptron offers a simple solution, but it fails to perform well on the non-separable cases. Widrow-Hoff method, overcome the problem of non-separability but this is at the cost of guaranteeing a separable solution for separable cases. Ho-Kashyap method performs significantly better than the other two methods and produces a solution both of the separable and non separable cases.
References

Chapter 3

Fitting an LDF by Generalised Linear Modelling

3.1 Introduction

In the previous chapter, through computer simulations some of the properties of linear discriminant functions have been presented. The central problem in defining such functions is to find a solution vector using a training data set such that an error rate is minimised.

If a solution vector exists which correctly classifies all the samples (the separable case), simple iterative error-correcting methods such as Perceptron are guaranteed to converge to a separable solution. However in most practical problems the training data form a non-separable set, due to large feature variation within each class. In such cases methods such as Perceptron become unstable, in the sense that the iterative process fails to converge.

More sophisticated algorithms such as Widrow-Hoff and Ho-Kashyap, however can converge to a solution for both the separable and non-separable cases, but such
methods suffer from a slow convergence.

In this chapter an algorithm for linear pattern classification is proposed. It will be shown that this algorithm converges faster in most cases, and produces a smaller error rate in non-separable cases.

In the proposed algorithm the linear discriminant function is transformed by a 'link' function which associates with each feature vector a measure in the range 0 to 1 according to the class membership of the pattern. Thus instead of dealing with a linear discriminant function, which is unbounded and can take any value on the real line, the Generalised Linear Model (GLM) method uses the non-linear transformation of a linear function, together with an iterative algorithm which estimates the unknown parameters in the discriminant function. The coefficients of the linear discriminant function form the solution vector.

A computer simulation of the method has been applied to examples from the literature, and also to a binary template matching problem arising in digital facsimile coding. The performance of the GLM method is compared with the linear classification algorithms discussed in Chapter 2.

The GLM algorithm is presented by first introducing the linear regression model and then generalising it to a transformed linear model by means of a link function.

3.2 Linear Regression Model

A linear regression model is a model in which the parameters, ie the quantities to be estimated, appear linearly. An example of linear regression is:

$$y_i = \sum_{j=1}^{n} \beta_j x_{ij} + \varepsilon_i \quad i = 1, \ldots, m \quad \ldots \ldots (3.1)$$

The $\beta_j$ (j=1, ..., n) represent the unknown parameters, assumed to be constant for a given model, whereas $x_{ij}$ are values of n independent variables, j=1, ..., n (often called regressor or predictor variables) for m samples, i=1, ..., m. The $y_i$, i=1, ..., m are the observations or values of dependent variable. Each value of the $y_i$ deviates
from the expected value by an amount $\varepsilon_i$, which is a random error term whose distribution is unknown but which is assumed to have mean value of zero.

The dependent variable can also be considered to be the sum of a systematic component and one random variable:

$$y_i = \mu_i + \varepsilon_i$$

where in this case the systematic component is the linear predictor:

$$\mu_i = \sum_{i}^{n} \beta_i x_{ij}$$

or in the vector form:

$$\mu = X\beta$$

Where $\mu$ is $(m \times 1)$, $X$ is $(m \times n)$ and $\beta$ is $(n \times 1)$. If $(m=n)$ and $X$ is of full rank, then there is a unique solution satisfying $\varepsilon = 0$, given by:

$$\beta = X^{-1}y$$

If $(m > n)$ there will not generally be a solution satisfying $\varepsilon = 0$. Instead $\beta$ is calculated so that the magnitude of the error $\varepsilon$ is minimised in some sense.

### 3.2.1 Method of Maximum Likelihood

Essentially this involves finding a vector $\beta$ which minimises the likelihood or probability of obtaining the observations $y = X\beta + \varepsilon$ under certain assumptions about the distribution of $\varepsilon$. 

42
A common assumption in linear regression is that the $e_i$ are independent Gaussian random variables with zero mean and constant variance. In this case the maximum likelihood estimate of $\beta$ is identical to the least square estimate, which minimises the sum of the squares of the residuals $e_i$:

$$
E = \sum_{i=1}^{m} e_i^2 = \sum_{i=1}^{m} \left( y_i - \hat{y}_i \right)^2 
= \sum_{i=1}^{m} \left( y_i - \sum_{j=1}^{n} \beta_j x_{ij} \right)^2
$$

The minimisation of $E$ is effectively what the LMS algorithm performs by gradient descent. There is also a direct non-iterative solution available as follows:

Considering $X$ is ($m \times n$) and $y$ is ($n \times 1$);

$$X^TX \beta = X^Ty$$

and provided the information matrix $X^TX$ has an inverse, the least squares estimate of $\beta$ follows from:

$$\beta = (X^TX)^{-1} X^Ty$$

This approach will directly yield the solution which minimises (3.4), and is therefore identical to the limiting solution of the LMS algorithm, since both methods are minimising the same cost function.

### 3.3 The Generalised Linear Model

Generalised Linear Models (GLM) in the context of regression analysis were originally defined by Nelder and Wedderburn. The essential components of a
GLM are as follows:

a) The random component: \( y_i \) having independent Normal distribution with \( E(y_i) = \mu_i \) and constant variance. This is also known as the observation or the response variable.

b) The systematic component \( x_{i1} \ldots x_{in} \) and an associated linear predictor \( \eta_i \) given by:

\[
\eta_i = \sum_i^n \beta_i x_{ij}
\]

c) The link function between random and systematic component:

\[
\eta_i = g(\mu_i)
\]

In this type of formulation classical linear models are such that the response variable has Normal distribution and the link function is simply the identity function so that \( \mu_i \) is equal to \( \eta_i \).

Generalised Linear Models allow two extensions. Firstly the distribution of the random response component may come from an exponential family other than the Normal distribution and secondly the link function may become any monotonic differentiable function[4].

3.4 GLM Applied to LDF

In the application of GLM to LDF, the independent variables are the pattern features, and the outputs (dependent variable) \( y_i \) are coded according to the pattern class. Confining the discussion for the moment to the two-class case, the LMS algorithm codes \( y_i \) as \( b \) if pattern \( i \) belongs to class \( C1 \), and as \(-b\) if pattern \( i \) belongs to class \( C2 \), where \( b \) is an arbitrary positive constant. The decision boundary is defined as the hyperplane in the feature space consisting of all points where the linear predictor, \( \eta_i \), evaluates to zero.

In more general terms, the output could be coded, according to pattern class, as any
two distinct constants, with the decision boundary defined as the hyperplane over which the linear predictor evaluates to midway between the two constants. Regardless of choice of the two constants, the LMS method will be equivalent, subject to a linear adjustment to the parameters[5].

In the following discussion, binary coded outputs are assumed, i.e.

\[
y_j = \begin{cases} 
1 & \text{for class } C1 \\
0 & \text{for class } C2
\end{cases}
\]

so that the decision boundary is formed at \( \eta_i = 0.5 \).

There are now two reasons why it may be considered inappropriate to fit an LDF to such data by the method of least squares.

Firstly, the linear predictor \( \eta \) is effectively unbounded, given arbitrary continuous-valued features. It is therefore possible for the residuals \( |e_i| = |y_i - \eta_i| \) to become arbitrarily large, even for correctly classified samples.

Consider a simple example, where there is a single feature, the two classes are linearly separable, but the patterns of one class have more widely dispersed feature values than those of the other class.

Such a case is illustrated in Fig (3.1), which shows that the least squares estimate of the LDF fails to provide a separating solution. Hence both the LMS algorithm and the linear regression method outlined above would yield a solution which is non-optimal, in the sense that it does not minimise the classification errors in the training set.

The second point relates to the distribution of the response variable \( y_i \). Since the only possible values are 0 and 1, the only applicable probabilistic model for such a variable is a binomial distribution of order 1 and with an undetermined probability \( \mu_i \) (0 \( \leq \mu_i \leq 1 \)). The mean and variance of this distribution are \( \mu_i \) and \( \mu_i (1 - \mu_i) \) respectively. Under the assumptions of maximum likelihood, the linear predictor
Fig. (3.1) Example of Least Squares LDF and LMS Decision Boundary
corresponds to the mean of such a distribution which maximises the probability of realising the observations $y_i[6]$. But as mentioned previously, the linear predictor $\eta_i$ is not necessarily bounded in the unit interval $[0,1]$, and can not therefore equate to the mean $\mu_i$ of such a distribution. Moreover, since the assumed distribution is not Normal, the maximum likelihood and least squares estimates can no longer be assumed equivalent.

A solution to the first problem is now proposed by passing the LDF through a non-linear ‘link’ function prior to fitting[7]. This is not new to pattern recognition, the function proposed below is identical to the one used in the backpropagation formulae for multilayer Perceptron (see Chapter 5).

The second problem is solved by adapting a cost function whose minimum corresponds to the maximum likelihood solution.

It will be shown how with the above reformulations the solution is achievable by the method of GLM. Although this method is well documented in the general case, its application to discriminant pattern recognition is believed to be original in this work.

3.4.1 Link Function

It has been shown above that the mean $\mu_i$ of the response to a particular feature $x_i$ is bounded in the interval $[0,1]$. The assumption here is that the dependency of $\mu_i$ on $x_i$ is through the linear combination:

$$\eta_i = \sum_{j=1}^{n} \beta_j x_{ij}$$

for some unknown coefficients $\beta_1, ..., \beta_n$. Unless restrictions are imposed on these coefficients we have $-\infty < \eta_i < \infty$, so that to express $\mu_i$ directly as such a linear combination would be inconsistent with the laws of probability. A simple way of avoiding this problem is to use a transformation, or link function, $g(\mu)$ which maps
the unit interval [0,1] onto the real line (-∞, ∞). This leads to an instance of
Generalised Linear Models with

\[ g(\mu_i) = \eta_i = \sum_{j=1}^{n} \beta_j x_{ij} \]  

\[ \text{---(3.7)} \]

a commonly used link function is the empirical logistic or logit function[8]

\[ \eta = g(\mu) = \ln \left( \frac{\mu}{1-\mu} \right) \]  

\[ \text{---(3.8)} \]

whose inverse is

\[ \mu = g^{-1}(\eta) = \frac{1}{1 + e^{-\eta}} \]  

\[ \text{---(3.9)} \]

This is illustrated in Fig (3.2). A decision boundary is then obtained by solving for
the case \( \mu = 0.5 \), which corresponds to \( \eta = 0 \).

The solution will give a hyperplane which divides the feature space into two regions,
corresponding to the two classes of patterns. Unknown patterns may then be
assigned as follows:

- **C1** if \( \mu > 0.5 \) i.e. if \( \eta > 0 \)
- **C2** if \( \mu < 0.5 \) i.e. if \( \eta < 0 \)
- Undecided if \( \mu = 0.5 \) i.e. if \( \eta = 0 \)

### 3.5 Maximum Likelihood Cost Function

The probability of observing \( y_i \) is \( \mu_i \) if \( y_i = 1 \) and \( (1 - \mu_i) \) if \( y_i = 0 \). Thus for a
sequence of independent observations \( y_i (1 \leq i \leq m) \), the log likelihood is[9]:

\[ \text{---(3.5)} \]
Fig.(3.2) Typical Sigmoid Function
\[
L = \sum_{i=1}^{m} \left\{ y_i \ln(\mu_i) + (1 - y_i) \ln(1 - \mu_i) \right\} \quad \ldots\ldots (3.10)
\]

The Maximum likelihood estimate of \( \beta \) is therefore that which generates via the link function a vector \( \mu \) which maximises \( L \) in equation (3.10)

### 3.6 Obtaining the Maximum Likelihood Estimate

A cost function such as equation (3.10) can be maximised by a gradient method as it will be shown in Chapter 6. However a more elegant method has been reported in connection with GLM[5].

Based on the method of maximum likelihood the parameter \( \beta \) in the linear predictor \( \eta \) can be estimated. McCullagh and Nelder[5] show how for generalised linear models, this method may be formulated in terms of an iterative weighed least squares. In this approach, the dependent variable is not \( y \) but \( z \), a linearised form of the link function applied to \( y \), and the weights \( W \) are functions of the fitted values \( \mu \).

\[
z = \hat{\eta} + (y - \hat{\mu}) \left( \frac{d\hat{\eta}}{d\hat{\mu}} \right) \quad \ldots\ldots (3.11)
\]

\[
W = \frac{1}{\text{var}(y)} \left( \frac{d\hat{\eta}}{d\hat{\mu}} \right)^{-2} \quad \ldots\ldots (3.12)
\]

where \( \hat{\eta} \) is the current estimate of the linear predictor, with corresponding fitted values \( \hat{\mu} \).

The process is iterative because both the adjusted dependent variable \( z \) and the weight \( W \) depend on the fitted values, for which only current estimates are available. The procedure, as applied with the logistic link function to the maximisation of \( L \) (equation (3.10)) is as follows:
Form the adjusted dependent variable $z$ with typical value:

$$ z_i = \hat{\eta}_i + \frac{y_i - \hat{\mu}_i}{\hat{\mu}_i (1 - \hat{\mu}_i)} $$

and the diagonal weight matrix $W$ with diagonal element

$$ w_{ii} = \hat{\mu}_i (1 - \hat{\mu}_i) $$

A new estimate of $\beta$ can then be obtained by solving the linear regression equation:

$$ X^T WX \hat{\beta} = X^T Wz $$

The process underlying the iteration is summarised as follows:

1) Initially setting

$$ \mu_i = 0.5 \quad 1 \leq i \leq m $$

from which the other variables become

$$ \hat{\eta}_i = 0 $$

$$ z_i = \begin{cases} 2 & \text{if } y_i = 1 \\ -2 & \text{if } y_i = 0 \end{cases} $$

$$ w_{ii} = 0.25 $$

2) Calculate:

$$ r = X^T Wz \text{ and } A = X^T WX. $$

3) Solve for $\hat{\beta}$ ;

$$ \hat{\beta} = A^{-1} r $$
4) If convergence detected stop, else continue.
5) Update estimates:
   \[ \hat{\eta}_i = X \hat{\beta} \]
   \[ \hat{\mu}_i = \frac{1}{1 + e^{-\hat{\eta}_i}} \]
   \[ w_{ii} = \hat{\mu}_i (1 - \hat{\mu}_i) \]
   \[ z_i = \hat{\eta}_i + \frac{y_i - \hat{\mu}_i}{\hat{\mu}_i (1 - \hat{\mu}_i)} \]

6) Repeat step 2 to 5.

At each iteration the matrix inversion \((X^T w X)^{-1}\) is performed. The iterations through the data set under consideration is terminated either by reaching a perfect classification (for separable data sets) or reaching a predetermined (maximum 100) number of iterations (for the non-separable cases).

3.7 GLM Simulations

This section reports on a software simulation of the GLM pattern classifier. It compares its performance with the LDF classifiers discussed in the previous chapter specifically with the H-K classifier. For purpose of comparison the same four data sets used in LDF simulations of Chapter 2 are also considered here.

1. Data set (a)

This is a linearly separable data set and its description can be found in Chapter 2. For this data set GLM like the H-K required only one iteration through the data set to produce a separable decision boundary. The decision boundary generated by GLM at
Graph (3.1) Distribution of Hand's Data and Decision Boundary Generated by GLM.
convergence is identical to H-K decision boundary and is shown in graph(3.1).

2- Data set (b)

This is a non-separable data set used in the study of fuzzy Perceptron by Keller[10]. The distribution of this data set and the decision boundary generated by GLM and H-K are shown in graph(3.2). Graph(3.3) illustrates the error rate against the number of iterations produced by GLM and H-K. As shown in this graph GLM outperforms the H-K and other LDF classifiers discussed in Chapter 2 including the fuzzy Perceptron. GLM produced an error rate of 7.43% only after two iterations through the data set which illustrates its superior speed of convergence over the other LDF classifiers.

3- Data set (c)

The decision boundary generated by GLM for the binary template matching data set is shown in graph(3.4). This graph shows that GLM has successfully separated the data set into two distinct regions. Graph(3.5) shows the error rate against the number of iterations produced by GLM and H-K. This graph once again shows the superior speed of convergence over H-K. The GLM method only required 7 iterations to produce a separable solution as compared with 178 for the H-K method.

4- Data set (d)

This is a non-separable template matching data set. Graph(3.6) shows the error rate against the number of iterations. This graph illustrates that the significantly faster speed of convergence of GLM over H-K for the non-separable data set. GLM required only 7 iterations through the data set to reach a steady state error rate of 3.75% as compared with 4.2% at 100 iterations for the H-K.

Table(3.1) summarises the performance of GLM and the other LDF methods of Chapter 2. Although GLM method requires more calculations per iteration, but as it can be seen from table (3.1) both separable and non-separable
Graph (3.2) Distribution of Keller's Data and Decision Boundary Generated by GLM.

Graph (3.3) Graph of Misclassification against Number of Iterations for Experiment Case b.
Graph (3.4) Distribution of a Template Matching Data and Decision Boundary Generated by GLM.

Graph (3.5) Graph of Misclassification against Number of Iterations for Experiment Case c.
Graph (3.6) Graph of Misclassification Against Number of Iterations for Experiment Case d.
Table (3.1) Summary of LDF Simulations

<table>
<thead>
<tr>
<th>Separable</th>
<th>Number of iteration required to reach zero classification error.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GLM</td>
</tr>
<tr>
<td>Data Set a</td>
<td>1</td>
</tr>
<tr>
<td>Data Set c</td>
<td>7</td>
</tr>
</tbody>
</table>

(a) Separable Cases

<table>
<thead>
<tr>
<th>Non-Separable</th>
<th>GLM</th>
<th>H-K</th>
<th>W-H</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Iterations to convergence</td>
<td>% Error rate</td>
<td>Iterations to convergence</td>
</tr>
<tr>
<td>Data Set b</td>
<td>2</td>
<td>7.4</td>
<td>4</td>
</tr>
<tr>
<td>Data Set d</td>
<td>7</td>
<td>3.75</td>
<td>100</td>
</tr>
</tbody>
</table>

(b) Non-separable Cases
cases GLM reaches a solution at a significantly faster speed of convergence. For the non-separable cases (data set B and D) GLM not only converges faster than other methods but also it produces a lower error rate.

3.8 Conclusion

The application of GLM to pattern classification problems has been developed and through computer simulations its performance has been compared with some of the existing methods.

Simulation results demonstrate that the GLM method always yields a separating vector for separable training data. For the non-separable cases, the GLM method proves to be stable and consistently converges to a solution giving the same or fewer misclassification errors that are obtained from the HK and WH methods. Although the GLM algorithm involves more computation per iteration (inverse of an m by m matrix where m is the number of features per iteration) than the other two methods, its overall convergence rate is shown to be significantly faster than the other methods tested.
References

Chapter 4

Extensions of Linear Discriminant Functions

4.1 Introduction

In the previous chapters methods for obtaining linear discriminant functions have been discussed. The ability of the LDFs to perform classifications degrades on problems where the input patterns are not linearly separable [1,2]. This is one of the serious drawbacks of LDFs.

In this chapter possible extensions of LDF's which attempt to overcome the linearity problem are introduced. These extensions generate non-linear decision boundaries and are divided into two classes (a) Quadratic (Polynomial) Discriminant Functions (QDFs) (b) Piecewise Linear (Multilayer) Discriminant Functions (PLDFs).

For the QDFs second order terms are introduced into the discriminant function. The second order terms and other terms are linearly related hence the methods described in earlier chapters can be transformed to include the second order terms and hence generate appropriate QDFs.
The second types of non-linear discriminant functions is Piecewise Linear Discriminant functions (PLDFs). They are consist of LDF layers and are known as Multilayer Decision Functions (MDFs). Although it is possible to obtain the weights of QDFs using the already existing methods for LDFs, MDFs do not lend themselves to the same analysis[3,4,5].

In this chapter some of the capabilities and difficulties involved in generating non-linear discriminants are presented.

4.2 Quadratic and Higher Order Polynomial

One method of extending LDFs that can generate non-linear discriminant function is through QDF[6]. The general form of QDF for an N-dimensional feature is given by:

\[
d(y) = w_0 + \sum_{j=1}^{N} w_j y_j^2 + \sum_{j=1}^{N} \sum_{k=j+1}^{N} w_{jk} y_j y_k + \sum_{j=1}^{N} w_j y_j \quad \quad (4.1)
\]

In this equation, the first summation on the right hand side consists of N-terms, the second summation of N(N-1)/2 terms and the third summation of N terms. Hence the total number of coefficients or weights to be be estimated is (N +1)(N +2)/2.

In spite of the fact that equation (4.1) represents a quadratic decision function, it is possible to treat this function as if it was linear by virtue of straightforward transformation. For simplicity and clear presentation considering a two-feature case \(y_1, y_2\) (N=2), equation (4.1) can be written as:

\[
d(y) = w^T y \quad \quad (4.2)
\]
where
\[ w = (w_0, w_1, w_2, w_{11}, w_{12}, w_{22}) \]
\[ y^* = (1, y_1^2, y_2^2, y_1 y_2, y_1, y_2) \]

Although equation (4.2) contains quadratic terms, all these terms are combined linearly and as it can be seen, equation (4.2) is equivalent to equation of LDF defined in Chapter 2. Therefore methods described in Chapter 2 and Chapter 3 can be extended so they will be capable of generating quadratic decision boundaries.

### 4.2.1 Simulation of QDF

In this section computer simulation results illustrating the performance of QDFs are presented. In this simulation GLM method (described in Chapter 3) was extended as described above, to QGLM (Quadratic GLM) generating quadratic decision boundaries. The description of the data set used in the is simulation is as follows:

**Chang data set:**

The data for this test is extracted from Chang[7] and its distribution is shown in graph (4.1). Clearly this data is a non-linearly separable case consisting of 15 samples belonging to Class 1 and 15 samples to Class 2. This graph shows a separable decision boundary generated by QGLM, requiring only one iteration through the data set.

Clearly QDFs are capable of producing non-linear decision boundaries overcoming the linearity problem of LDFs, but this is achieved at the cost of more computational time.
Chapter 4: Extensions of Linear Discriminant Functions

Graph (4.1) Distribution of Chang's Data and a Quadratic Decision Boundary Generated by QGLM.
4.3 Polynomial Discriminant Function

The QDF described in previous section suggests a general scheme for generation of polynomial Discriminant function (PDF)[8] of any finite degree. One way of expressing PDF in a general recursive form is:

$$d^r(y) = \left( \sum_{p_1=1}^{N} \sum_{p_2=p_1}^{N} \cdots \sum_{p_r=p_{r-1}}^{N} w_{p_1 p_2 \cdots p_r} y_{p_1} y_{p_2} \cdots y_{p_r} \right) + d^{r-1}(y) \quad \text{(4.3)}$$

where $r$ indicates the degree of polynomial, $N$ is the number of features and $d^0(y) = w_0$. This relationship provides a convenient method of generating decision functions of any finite degree. Once again methods described in Chapter 2 and Chapter 3 can be extended to evaluate the weights of any order PDF.

As might be expected, the number of terms needed to describe a PDF grows quite rapidly as a function of the order of polynomial and the feature size. In general an $r^{th}$ order PDF with N-dimensional feature size will have:

$$\text{Number of Weights} = C_r^{N+r} = \frac{(N+r)!}{r! \cdot N!}$$

This leads to a high dimensional space. For example, a case where $r = 8$ and $N = 10$, requires 43,758 weights to be evaluated. Although PDF method has the potential of generating considerably more complex decision boundaries using linear methods, it has the disadvantage of requiring an expansion of dimensionality in most cases. However in some cases the weights in a PDF do not contribute equally to the shape of the decision boundary and hence there would be some redundant weights. Therefore by locating and removing the redundant weights it is possible to reduce the number of terms in a PDF without a significant effect to the overall shape of the decision boundary[9].

64
4.4 Group Method of Data Handling (GMDH)

Originally this approach was introduced by Ivakhnenko\cite{10,11}, and used for modelling the input-output relationship of a complex system. This approach can be used as an efficient way of generating high order polynomials.

GMDH has a multilayer network structure as shown in Fig (4.1), each element in the network implements a non-linear function of its inputs. The function implemented is usually a second order polynomial of the inputs, since each element generally accepts two inputs, the function implemented by an element can be expressed as:

\[
z = \alpha + \beta y_i + \gamma y_j + \delta y_i^2 + \epsilon y_j^2 + \phi y_i y_j
\]  

where \(y_i\) and \(y_j\) denote the input variables, \(z\) is the output and the Greek characters indicate the weights.

Starting at the input level, for each pair of variables \(y_i\) and \(y_j\), the output of the first level generates \(N(N - 1)/2\) new quadratic variables. (\(N\) is the dimensionality of the input feature). This step is followed by a selection of what Ivakhnenko\cite{9} called the ‘useful’ terms from the ‘harmful’ ones. These ‘useful’ terms which are quadratic, are combined in exactly the same manner as the previous layer so to produce new higher order variables for the next layer (degree 4). This is repeated until the output layer, where the output of the elements represent the class membership of the input patterns.

Comparing GMDH with PDF, if one needs an \(8^{th}\) order polynomial for the input features having 10-dimensions, one requires 43,758 weights in the PDF. To find these weights by the PDF equations necessitates a large data set size, since the number of patterns in the data set must be >> 43,758 to obtain a good estimate of the weights. At any rate finding 43,758 weights in a PDF requires handling large matrices and effectively large computational time. GMDH on the other hand is able to produce a similar PDF by repeatedly solving quadratic equations which contain only 6 weights. For the above example, if all the quadratic variables generated at the first regression level were kept (that is keeping both ‘useful’ and ‘harmful’ variables) and only keeping two variables from the second regression level, in total GMDH would require solution to 1036 quadratic equations (45 for the first level +
Second generation
variables like \( z \) are quadratics in variables of the previous generation.

First generation
variables like \( u \) and \( v \) are quadratics in
\( y_i, y_j, y_k, y_l \)

\[
z = \alpha + \beta u + \gamma v + \delta u^2 + \epsilon v^2 + \phi u v
\]

Second generation
\[
v = \alpha + \beta y_i + \gamma y_j + \delta y_i^2 + \epsilon y_j^2 + \phi y_i y_j
\]

Zereth generation
variables are the original inputs
\( y_i, y_j, y_k, y_l \)

\[
u = \alpha + \beta y_i + \gamma y_j + \delta y_i^2 + \epsilon y_j^2 + \phi y_i y_j
\]

Fig 4.1- Basic block diagram of GMDH
990 for the second level + 1 for the final level). Of course in practice this number is far less since one drops the 'harmful' terms.

As it was described GMDH can be a very powerful method for generating very complex PDFs with relatively small computational effort. However GMDH method introduces extra storage (storing the newly generated high order variables) and the computational effort required by GMDH is only suitable for very complex problems.

4.5 Piecewise Linear Discriminant Functions

A Piecewise Linear Discriminant Function (PLDF) is a function which is linear over sub-regions of feature space. To introduce this class of DFs, a simple two class problem as shown in Fig (4.2) is considered.

A pattern \( y \) is assigned to \( C_2 \) if it lies on the positive side of either \( d_1(y) \) or \( d_2(y) \) and \( y \) is assigned to \( C_1 \) if it lies on the negative side of \( d_1(y) \) and \( d_2(y) \).

In this example the two classes are not linearly separable, but perfect separation is achieved by using two hyperplanes instead of one, which illustrates the general advantage of PLDF.

While it is possible to efficiently estimate the weights of LDF using techniques described in Chapter 2 and Chapter 3, PLDFs do not lend themselves to the same analysis[12]. Nilsson[13] and Fu[14,15] have both talked about 'layered machines' which can be described in terms of Piecewise linear decision surfaces.

An example of such a machine is shown in Fig.(4.3). This is a two layer machine which is known as a 'committee machine'. The first layer consists of an odd number of linear DFs and the 2\(^{nd}\) layer weights are fixed and is usually a unity vector[16]. The machine is 'committee machine' because it takes a majority vote over the output of each LDF to determine the classification of an input pattern.

The 'committee machine' produces an output of 1 if the majority of the inputs to the second layer are 1 and output of -1 if the majority inputs are -1. Only the weights in the first layer are adaptive, they are adapted in a way so to increase the number of
Fig.(4.2) Piecewise Linear Separation of Group $C_1$ and $C_2$. 
Fig.(4.3) The Committee Machine
Nilsson[12] describes an intuitive training procedure, similar to an error-correcting procedure (The Perceptron). The suggested procedure makes appropriate adjustments to the first layer weights if they contribute to an incorrect majority vote. This procedure was not an efficient way of obtaining the PLDF weights and it also did not allow for second layer weight adaptation.

Another approach for the implementation of PLDF was initiated at Stanford by Widrow[17] in the early 1960s. Their implementation of PLDF had similar structure to Nilsson 'committee machine', that is having an adaptive first layer and a fixed second layer. In this case the inputs were applied to an Adaptive Linear Neuron (ADALINE), using LMS procedure in the first layer, and their outputs in turn were connected to a fixed logic device providing the PLDF output. An example of such a system is shown in Fig.(4.4).

In this example the two ADALINES are connected to an AND gate to provide an output decision. Widrow[18] has called systems of this type as MADALINES (Many or Multilayer ADALINES). Researchers at Stanford[18] have also shown that MADALINES can be constructed with various fixed logic devices in the second layer (eg OR, NOT).

The fundamental difficulty in these types of systems was the inefficient adaptation of weights in all layers of a multilayer system. However recently a more effective type of weight adaptation for layered machines was discovered in 1980s and gave rebirth to the field of Neural Networks, which will be discussed in the later chapters.

4.6 Conclusion

A simple extension of LDF for generating non-linear decision boundary is through QDF. The advantage of using QDF is in the usage of the existing LDF methods. This is done by introducing second order terms which appear linear in the discriminant function. Higher order discriminant functions (polynomial DFs) can also be produced using the same principle. However this is computationally very expensive way of producing more complex decision boundaries, since PDF would require many more weights to be evaluated. Moreover not all the weights contribute equally
Fig. (4.4) A MADALINE System
to the final decision functions. Hence in order to have an efficient classifier one needs to identify and remove these redundant weights therefore introducing more computational overhead.

A powerful and efficient way of producing non-linear decision boundary is through GMDH. GMDH requires relatively fewer weights to be evaluated, but it introduces extra storage and is only suitable for very complex problems.

A third way of producing non-linear DFs is through PLDF, which is linear over a sub-region of the feature space. In PLDF layers of LDFs are used to produce Piecewise decision boundaries. The major difficulty in designing a PLDF classifier is in the efficient way of evaluating the weights in various LDF layers. A more effective way of weight adaption is through Backpropagation neural networks which are introduced in the next chapter.
References

Chapter 5

Neural Networks

5.1 Introduction

This section presents a brief review, merely setting the stage for later discussion on neural networks. Due to the sheer size and complexity of neural networks literature, this review has been very selective.

5.1.1 What Is A Neural Network?

Neural Networks are artificial structures which are constructed in such a way as to make use of some of the organisation principles that are thought to be used in the human brain. An artificial neural network uses a large number of simple processing nodes, analogous to neurons in a brain, each connected to some number of other nodes in a network. These nodes are capable of only a few simple actions such as Boolean and/or arithmetic operations on the incoming signals, and sending signals
In biological neurons the distinctive processing ability of each neuron is supposed to reside in the electrochemical characteristics of the inter-neuron connections or synapses[1]. In many artificial neuron systems this is accomplished by altering the pattern of interconnections among the nodes or by modifying a quantity called the connection strength (or synaptic weight) associated with each connection. Hence over a long term a network of simple processing nodes embodies knowledge which is stored in the connection strengths[2]. The manner in which this information is attained by a network is known as a learning process.

Another interesting aspect of these networks is their parallel functionality. All of the connections can carry signals simultaneously, and all of the neurons can act in parallel to process their incoming signals from other neurons[2].

Numerous types of neural networks have been investigated by researchers in this area[3]. In the next few sections the development of neural networks and descriptions of some popular networks will be presented.

5.1.2 A Brief History of Neural Networks: The Early Years

In recent years there has been a great interest in the field of artificial neural nets. Neural nets have been proposed as a possible solution to a wide range of computational problems, particularly in the area of speech and vision. However the study of neural nets dates back as long as 40 years. But in those days neural nets were only a small subset of pattern recognition rather than a self-contained subject.

One of the earliest works reported in this field is due to McCulloch and Pitts in 1943[4]. The McCulloch-Pitts neuron (or threshold logic unit) is a binary device, that is it can be in one of only two states. Each neuron has a fixed threshold as shown in Fig (5.1). The central result of their work was that any finite logical expression can be realised by McCulloch-Pitts neurons.

This was an exiting result, since it showed that simple nodes connected in a network could have immense computational power. It must be stressed that the crucial factor was not the complexity of nodes, but rather the way in which they were connected.
Fig (5.1) (a) The general representation of McCulloch-Pitts neuron.
(b) The OR gate: \( x_1 + x_2 \geq 1 \) so long as either input is 1.
(c) The AND gate: \( x_1 \cdot x_2 \geq 2 \) only if both inputs are 1.
together to achieve highly complex computation. Since the nodes were derived from neurophysiology, it presented a powerful model for the logic of the brain.

In 1949 Donald Hebb\(^5\) indicated that the weights on an input to a neuron should be varied to reflect the correlation between input and the output neurons. Hebb's neural model is famous among neural models because it was the first explicit statement of the physiological learning algorithm for synaptic weight modification that became known as the Hebbian rule. Hebb did not provide a mathematical formulation of his rule, but the weight update using his rule is simply carried out by adding a positive proportion of the input signal to the previous values of the synaptic weights. The problem with the Hebbian rule is that the weight change is always positive hence there is a chance of synaptic weight saturation\(^6\).

Normalisation and the use of inhibitory synapses\(^7\) were proposed to prevent synaptic weight saturation associated with a Hebbian learning rule.

The next milestone was the invention of the Perceptron by Rosenblatt in 1958\(^8\). Rosenblatt did not make the assumption that the function of a neuron is fixed for all time. Instead he allowed the weights on each neuron to change with time. The purpose of this was to allow the neural network to change itself with time in such a way to as to learn from experience. One may think of the Perceptron as a pattern recognition device that is not built to recognise a specific set of patterns, but rather has some ability to learn to recognise the patterns of a set after a finite number of trials.

One of the most significant results presented by Rosenblatt\(^9\) was the proof that a simple training procedure will converge to a solution to the problem if such exists.

The major difference between the Hebbian and Perceptron algorithms is that the Perceptron requires specification of the desired response for each neuron, so it can compare it with the actual response of a neuron, and hence form an error signal that will be used for modifying the weights.

The Hebbian rule was the first non-supervised learning procedure and Rosenblatt's Perceptron algorithm was the first supervised learning procedure.

In 1960 Widrow and Hoff\(^10\) proposed an adaptive system that could learn quickly and accurately. It was called Adaptive Linear Neuron (ADALINE) which was a
threshold logic unit with variable connection weights. They assumed that the system is able to form a continuous error signal between the desired response and the computed output. Using the error signal, the connection weights are adjusted and the output is recomputed until the error signal becomes zero or close to zero.

Minsky and Papert[11] analysed in some detail the capabilities of Rosenblatt’s Perceptron. They showed that there is an interesting class of problems (eg Xor discussed later in this chapter) that a single layer Perceptron can not solve, and they held out little hope for the training of more complex (multilayer) Perceptron systems.

This proved to be a near fatal blow to the whole field of neural networks. However, much work continued in what had become an unfashionable area, living in the shadow of symbolic Artificial Intelligence and expert systems.

5.1.3 Rebirth of Neural Networks

In 1982 Hopfield[12] showed that a highly interconnected network of threshold logic units could be analysed by considering it to be a physical dynamic system possessing an ‘energy’. Hopfield’s network starts in an initial random state and goes on to a stable final state. This process parallels the action of the system falling into a state of minimum energy.

Criticism has been made by some researchers[13] in the neural network field, that there was nothing fundamentally new in the model proposed by Hopfield. However the one thing that attracted a lot of attention to Hopfield’s work was the successful development of neural network chips based on his proposals by AT and T[14].

The idea of introducing hidden layers (that is layers of neurons that are neither input nor output) for a Perceptron emerged during 1960’s by Rosenblatt[15] and Nilsson[16]. Unfortunately no efficient practical learning algorithm was available for such a network at that time.

However, a breakthrough lead to the discovery of back error propagation or simply backpropagation (BP), which is a learning procedure for working with networks containing hidden layers (multilayer).
As noted by Grossberg[17], BP has claims to multiple authorship. It was discovered by Werbos[18], rediscovered by Parker[19], Le Cun[20], and discovered again and made popular by Rumelhart, Hinton and Williams[2].

The BP learning algorithm is an error correcting learning procedure that generalises the Widrow-Hoff procedure. It's implementation involves two passes through a multilayer network: a forward pass through the layers in which an error signal is estimated, and a backward pass modifying the synaptic weights in order to reduce the error. This process is repeated until the network has settled down to a solution.

Since its discovery, BP has been the most popular learning algorithm for working with multilayer networks.

The remainder of this chapter is given to a discussion and development of BP, the advantages and disadvantages of this learning algorithm, and possible improvements to speed up the learning process. Finally the effectiveness of BP to learn complex tasks is presented using computer simulations.

5.2 Back Error Propagation (BP)

BP is a supervised learning algorithm which is a generalisation of the Widrow-Hoff procedure. BP is applicable to multilayer networks, that is networks having layers of hidden nodes between their input and output.

Given a set of labelled training data, the Widrow-Hoff technique forms an error signal which is the difference between the actual and desired output. The synaptic weights are adjusted in proportion to the error signal, which progressively diminishes as a result of these adjustments. This algorithm is easily applied to networks that only possess input and output nodes. However in multilayer networks the problem of adjusting the weights is more complicated.

In multilayer networks the error signal can be formed as before, but in this case many (or all) hidden nodes contribute to the error, not just the ones at the output layer. Since the actual and desired states of hidden nodes are not specified by the task, one can not directly compute the error signal for hidden nodes.
This is achieved by backward flow of error information to the hidden nodes contributing to the error.

Hence the BP algorithm aims at driving the error function towards its global minimum by suitable adjustments to the network weights. This essentially constitutes a minimisation problem that the BP algorithm attempts to solve by a gradient descent technique.

5.3 Node Structure of a Multilayer Network

In general a multilayer network is made up of nodes arranged in layers. A system architecture for such a network is shown in Fig (5.2). Except for the input layer nodes, the input to each node is the weighted sum of outputs from the nodes in the prior layer. The total input to a node \( j \) in either a hidden layer or the output layer is

\[
x_j = \sum_i w_{ij} y_i + w_{0j}
\]  

(5.1)

Where \( x_j \) is the weighted sum of the inputs to the node \( j \), \( w_{ij} \) is the weight associated with the connection from node \( i \) in the previous layer, and \( w_{0j} \) is a bias term.

The output from node \( j \) is (as shown in Fig (5.2)):

\[
y_j = f(x_j)
\]  

(5.2)

where \( y_j \) is the output of node \( j \).

The activation function \( f \) is a non-linear monotonically increasing function which is usually a smooth approximation to the classic step function of a threshold logic unit.
Fig (5.2) Schematic description of a multilayer network with one hidden layer
Throughout this research work the logistic activation function has been used:

\[ y_j = f(x_j) = \frac{1}{1 + e^{-x_j}} \]  

(5. 3)

This is a sigmoidal function bounded in the interval \((0, 1)\) as shown in Fig(5.3b).

Other sigmoidal monotonic functions can also be used. However, as will be shown in the next two sections, the learning algorithm requires the activation function to be continuous and differentiable so to allow for appropriate gradient calculation.

5.4 Training with BP

Network training starts by randomising all the weights to small numbers. Randomising the initial weights overcomes the problem of weight symmetry. That is if all the weights start with equal values and the solution requires unequal weights to be developed, the network will never learn. This is because the back-propagated error will have the same response for all the hidden nodes and since the weights change depends on the error signal, therefore all the weights will remain equal and the system never learns.

The application of the BP learning algorithm to training a network involves two passes. The first phase of the training starts by presenting a pattern to the input layer of the network, which is initialised with small random starting weights.

The nodes of each layer produce a weighted sum of their input, which is passed through a sigmoid non-linearity. Finally the output of the non-linearity is transmitted to the nodes in the next layer. This process is repeated until reaching the output layer where each node produces an estimate value of its desired state. This is the ‘forward pass’ of the algorithm.

The second pass involves a backward pass through the network (analogous to the forward pass) during which an error signal for each node in a layer is calculated and then propagated to the previous layer.
Fig.(5.3) Examples of activation functions.
BP provides an algorithm for recursive calculation of error signals for all nodes in a hidden layer.

The first step is to calculate the error signal at the output layer, from which changes to the weights in the output nodes are then calculated. The error signal at the top layer is then propagated back to the previous layer, allowing the error signal and thus weight changes for all nodes in this layer to be computed. Changes to the weights in the hidden layer nodes can then be calculated. The formulae for updating the connection weights are discussed in section 5.6. This process continues for all the layers in the network. This is the 'backward pass' of the algorithm.

These two passes are repeated for all the patterns in a training set, which constitutes a training cycle. The training cycle is repeated until the network converges to a satisfactory performance.

5.4.1 Online and Batch Training

In the original version of BP algorithm formulation by Rumelhart et al[2], the network weights are updated in a manner to reduce the error for a single pattern, as rapidly as possible. In general, different results are obtained depending on whether one carries out gradient descent in weight space on a single pattern presentation or on total pattern presentation. In the former case, the corrections to the weights are made sequentially, that is after presentation of a single pattern (one at a time). This is known as the 'online' version of the procedure.

A true gradient descent procedure for minimising network error should be based on minimisation of total error (error arising from all training patterns). In this case the weight updates are found by summing the weight changes for each pattern in the training set and, only when all patterns have been included, updating the weights. This variation of BP is known as the 'batch' version. This version is feasible for adaptive pattern recognition, but is unlikely to correspond to processes carried out in biological neural networks[21].
5.5 The Error Function

Training the network is achieved by minimising an error function which is calculated as follows:

Each pattern vector in the training set is presented to the input nodes of a network, and a forward pass is carried out. At the output layer the final outputs are subtracted from the desired or target outputs to obtain an error signal. The error for a single pattern using sum of squared error (SSE) is given by:

\[ E = \frac{1}{2} \sum_j (y_{jk} - d_{jk})^2 \]  \hspace{1cm} (5.4)

and the total error for all the training data is:

\[ E_{\text{Total}} = \frac{1}{2} \sum_k \sum_j (y_{jk} - d_{jk})^2 \] \hspace{1cm} (5.5)

where the factor one half is inserted for mathematical convenience, \( k \) represents summation over all training patterns, \( j \) represents summation over the output nodes, \( y_{jk} \) is the actual value of an output node and \( d_{jk} \) is its desired or target value.

5.6 Computing the Weight Change

As described earlier, the weights in a network are adjusted using gradient descent method. Hence the changes in the network weights for a single pattern presentation are as follows:

\[ \Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} \] \hspace{1cm} (5.6)
where $\eta$ is a positive scalar which controls the learning rate. The partial derivative $\partial E/\partial w_{ij}$ can be evaluated using the chain rule:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial x_j} \cdot \frac{\partial x_j}{\partial w_{ij}}$$

Using expression (5.1)

$$\frac{\partial x_j}{\partial w_{ij}} = y_i$$

Let

$$\delta_j = \frac{\partial E}{\partial x_j}$$

hence using expression (5.4)

$$\Delta w_{ij} = -\eta \delta_j y_i \quad \ldots (5.7)$$

The above expression is similar in form to the delta rule defined by Widrow-Hoff as shown in section (2.5.1). To compute $\delta_j$, the chain rule is applied once more giving:

$$\delta_j = \frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial x_j}$$

Therefore for an output node $j$, $\delta_j$ is expressed as:

$$\frac{\partial E}{\partial y_j} = (y_j - d_j)$$

$$\frac{\partial y_j}{\partial x_j} = y_j (1 - y_j)$$

$$\delta_j = (y_j - d_j) y_j (1 - y_j) \quad \ldots (5.8)$$
For a hidden node $i$:

$$
\delta_i = \frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial y_i} \cdot \frac{\partial y_i}{\partial x_i}
$$

$$
\frac{\partial y_i}{\partial x_i} = y_i \left(1 - y_i\right)
$$

$$
\frac{\partial E}{\partial y_i} = \sum_j \left\{ \frac{\partial E}{\partial x_j} \cdot \frac{\partial x_j}{\partial y_i} \right\}
$$

$$
= \sum_j \left\{ \frac{\partial E}{\partial x_j} \cdot w_{ij} \right\}
$$

$$
= \sum_j \delta_j w_{ij}
$$

where the summation index $j$ represents all the outputs emerging from the node $i$.

Hence for the hidden node $i$:

$$
\delta_i = y_i \left(1 - y_i\right) \sum_j \delta_j w_{ij} \quad \text{(5.9)}
$$

From the above expression it can be seen that the $\delta$'s at a hidden node are evaluated in terms of the $\delta$'s at the upper layer. Thus starting at the output layer, $\delta$'s are evaluated and then propagated through until the input layer, and hence the name error backpropagation.

It must be noted that the bias terms $w_{0j}$ are adjusted in the same manner as are the other weights. It is simply considered that $w_{0j}$ are the weights from an imaginary node that always has an output value of unity.

In the 'batch' version, implemented in this research work, it is necessary to compute the partial derivative of $E_{\text{Total}}$ with respect to each weight in the network. This is
simply the sum of the partial derivatives for each training pattern $k$, that is:

$$\frac{\partial E_{\text{Total}}}{\partial w_{ij}} = \sum_k \frac{\partial E_k}{\partial w_{ij}}$$

Hence after a forward and a backward pass the weight changes are accumulated and only after all the patterns have been seen, the weights are updated:

$$\Delta_{\text{Total}} w_{ij} = \sum_k \Delta_k w_{ij} \quad \cdots (5.10)$$

The network processing cycle is continued until the network reaches a satisfactory level of performance (convergence).

The main steps of BP batch training algorithm can be summarised as follows:

1) Initialise all $w_{ij}$ to small random values (usually between ± 0.5)

2) For each pattern in training set calculate:

$$x_j = \sum_i w_{ij} y_i + w_{0j}$$

$$y_j = f(x_j) = \frac{1}{1 + e^{-x_j}}$$

3) For each pattern compute $\delta_j$:

  for an output node $\delta_j$ is

$$\delta_j = (y_j - d_j) y_j (1 - y_j)$$

  for a hidden layer node $\delta_i$ is

$$\delta_i = y_i (1 - y_i) \sum_j \delta_j w_{ij}$$
where \( j \) is over all nodes in the layer above node \( i \).

4) Adjust each weight by:

\[
    w_{ij}(n+1) = w_{ij}(n) - \eta \delta_j y_i
\]

where \( n \) denotes the iteration index.

5) Repeat steps 2 to 5 until a convergence criterion is satisfied.

There is no general way of detecting convergence, however one or more of the following rules can be used to detect convergence. These are:

a) Checking for correct classification of all samples in the training data,

b) Reaching a predetermined number of iterations,

c) Falling below some prespecified mean square error,

d) Checking for very small reduction in error over a fixed number of iterations.

5.7 Capabilities of a Multilayer network

Capabilities of a multilayer network are usually assessed by the complexity of the decision boundaries that can be produced. Fig (5.4) extracted from Lippmann[3] illustrates such capabilities for one, two and three layer networks that use a hard limiting non-linearity as their activation function. The second column of Fig (5.4) shows the type of decision boundaries that can be formed by different networks. The next two columns illustrate examples of decision boundaries which could be formed for the exclusive OR and a meshed region respectively.

From this figure it can be seen that a one layer network (ordinary Perceptron) can produce a linear decision boundary, a two layer network can produce any convex decision boundary, and a three-layer network can produce any arbitrary complex decision region and can therefore separate samples of patterns even though the pattern distribution might form a meshed region in pattern space.

Lippmann[3] demonstrates that no more than three layers in a network are necessary to create any type of decision boundary required in a classifier.
<table>
<thead>
<tr>
<th>Structure</th>
<th>Exclusive OR problem</th>
<th>Classes with meshed region</th>
<th>Types of decision region</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single layer</td>
<td></td>
<td></td>
<td>Hyperplane (A line for two dimensional case)</td>
</tr>
<tr>
<td>Two layer</td>
<td></td>
<td></td>
<td>Convex region</td>
</tr>
<tr>
<td>Three layer</td>
<td></td>
<td></td>
<td>Arbitrary (Complexity limited by number of nodes)</td>
</tr>
</tbody>
</table>

![Diagram](image)

Fig (5.4) Types of decision regions that can be produced by single and multilayer networks.
However, recently it has been demonstrated that two layer networks can also form decision boundaries that are not simply convex[22, 23]. Fig (5.5) is an example which illustrates how disjoint decision boundaries can be generated using a two layer network. The two disjoint shaded areas in Fig (5.5) represent decision regions for class $C_1$ (where output node has high output, $y = 1$). The remaining area represents the decision region for class $C_2$ (output node has low output, $y = 0$). The topology of a network that can create such decision boundaries is shown in Fig (5.6). The network uses the hard limiting non-linearity for its activation function. The connection weights and bias are indicated in the diagram.

In this type of network, each first layer node forms a hyperplane (a line in two dimensional space) in pattern space and the second layer nodes act to combine (in this example they perform an AND operation) these hyperplanes to form the appropriate decision boundaries shown in Fig (5.5).

The discussion of this section was centred on multilayer networks that have one output and use a hard limiting non-linearity for their activation function.

Similar behaviour is exhibited by multilayer networks with multiple output nodes when a sigmoidal non-linearity is used, and where the decision rule would be to select the class corresponding to the output node with the largest value[22].

The behaviour of these networks is more complex because decision boundaries are smooth curves instead of straight line segments and analysis is thus more difficult.

This has been the subject of comparative studies and extensive experiments carried out by Huang and Lippmann[22]. They show that there is no significant performance difference between two-layer and three-layer networks using BP training. However they have concluded that both types of network take an excessively long time to be trained in order to form complex decision boundaries such as shown in Fig (5.5).

5.8 Generalisation in MLN

Commonly people have very good ability to make accurate generalisation from a few scattered facts or discover patterns in a chaotic collection of observations. One of the
Fig (5.5) Disjoint decision regions formed by the network shown in Fig 5.6.

Fig (5.6) An example of two layer network topology that forms disjoint decision region
strength of neural networks is also their ability to generalise. A network can be said to generalise well when the input to output relationship is nearly correct for patterns never used in the significantly large set of typical cases in a training set.

Learning in neural networks may be viewed as curve fitting[24]. This analogy allows for generalisation to be seen as the effect of a good non-linear interpolation of the data.

Fig (5.7a) shows the input and output of a hypothetical network, and how generalisation can occur. A network that generalises well will produce a correct input to output relationship even when the input is slightly different from the exemplars used in the training. Therefore it is not desirable to have the network 'memorise' the training set.

'Memorisation' in a neural network corresponds to learning many specific input-to-output relationships without generalising between similar input/output pairs. This usually occurs when the network consists of many more nodes than the number required to solve a specific problem, coupled with availability of limited training patterns. In this type of situation the network behaves as a table look-up and implies that the input-to-output relationship of the network is not very smooth[24, 25].

Fig (5.7b) shows an example of how memorisation might take place for the same problem shown in Fig (5.7a).

Since no theoretical formulation for definition of generalisation exists, in practice a generalisation test usually consists of training the network until it performs well on the training set, and then measuring its generalisation performance on a set of examples that were not included in the training set.

5.9 Issues Involved in Designing MLN Using BP

Multilayer networks (MLNs) trained with BP have been successfully applied to various problems such as speech recognition[26, 27], sonar discrimination[28], to form text to phoneme rules[29] and many non-linear signal processing tasks[30,31]. Successful performance of MLNs on these problems depends on the appropriate design.

In this section some fundamental issues involved in designing MLNs
Fig. (5.7) Examples of generalisation
trained with BP are discussed.

5.9.1 Selection of Hidden Nodes

In designing an MLN an important question which arises is the size of the network needed to solve a particular problem. That is to say how many layers are required for the learning and discrimination task, and how many nodes are required in each layer.

In the previous section, capabilities of one-layer, two-layer and three-layer networks have been discussed. It is generally acknowledged that in comparison to a three-layer network a two-layer network is capable of producing equally good performance for most complex classification tasks.

For obtaining a satisfactory performance from a two layer network, the number of hidden nodes must be large enough to form a decision boundary that is as complex as is required by a given problem. If the number of hidden nodes is too small then no learning can occur. This is because the network will not be able to adequately model the class boundaries required by the problem. However the number must not be so large that too many weights would be involved and hence it would be difficult to reliably estimate them from the limited training data.

The two ad hoc ways of arriving at the required number of hidden nodes in a layer are:

To initially start training with a small number of hidden nodes, then go through a slow process of deciding whether any learning is taking place. Usually this is achieved by testing the network on the data which has not been used during the training, and monitoring the error produced by the network. If no learning takes place (no reduction or an increase in error on the test data) increase the number of hidden nodes and start training again. The process of adding a hidden node and retraining the network is repeated until satisfactory level of performance on the test data is obtained. Through this process the network size is determined, which is large enough for the specified task.

The second way is to start training with a very large network, then reduce its hidden nodes and start training again. The process of removing a hidden node and retraining
is repeated until the performance of the network starts to deteriorate.

Both of the above are very time consuming ways of determining the optimal (or near optimal) number of nodes in a hidden layer. Nevertheless they can be effective when the problem under consideration is relatively simple.

Other experimental research work in pattern classification appears to indicate that, the number of hidden nodes is a function of the input training patterns. For example, with a two dimensional input and a two class output, Lippmann[3] used 8 hidden nodes to obtain good results with 200 input training patterns. In another experiment, with a 60 dimensional input, a two class output and 192 sonar return patterns, Gorman and Sejnowski[28] obtained no improvement in performance as the number of hidden nodes exceeded 8. It was noted[32] that in both experiments an approximate relationship exists between $H$, the number of hidden nodes and $T$ the number of input training patterns:

$$\log_2 T = H$$

The above approximation to the optimal number of hidden nodes, was shown to be a poor approximation since it does not take into effect the dimensionality of input and output patterns.

Kung et al[33] observed that for a 10-dimensional input and 10-dimensional output the optimal number of hidden nodes for obtaining satisfactory performance was approximately $(M - 1)$ where $M$ is the number of training pattern pairs.

Although selecting optimal nodes using the number of training patterns was shown to be successful in some cases[32], at any rate as it is shown from above discussion, there exists no unique way of obtaining an optimal number of hidden nodes by relating it only to the size of the available training data.

Another approach to this problem is to start with a large number of hidden nodes and then systematically remove the nodes and their connection weights such that the overall performance of the network (or error level) remains the same. This method is known as 'network pruning'.

Several researchers have proposed methods to accomplish this reduction[34, 35]. One way of realising this is to add an extra term known as 'weight decay' to the
weight change formula[36]. The update formula, equation (5.7) can be modified to:

\[ \Delta w_{ij}(n) = -\eta \delta_j y_i - \beta w_{ij}(n-1) \quad \text{(5.12)} \]

Where \( \beta \) is a positive constant less than one and \( n \) indicates the \( n^{th} \) training cycle. The idea is that the weights that do not have much influence on decreasing the error while learning, i.e., which have \( \partial E/\partial w_{ij} = 0 \), will experience an exponential time decay:

\[ \Delta w_{ij}(n) = \beta^n w_{ij}(0) \quad \text{(5.13)} \]

When a redundant node and its associated weights is removed from the network this method does not require the restart of the training algorithm, hence the method of network pruning can arrive at the required number of hidden nodes faster than the ad hoc methods described earlier.

The weight decay allows redundant nodes to be removed from the network and thus yields a subset of the network which is efficient and has the same performance (i.e., no increase in error) as the complete network (original network), for a given training data. Unfortunately, this procedure does not give a complete answer to the question since the size of the initial network is still largely based on guess work. Despite this, it is reported that this method gives a very useful partial answer to networks that are not so large, but applying this technique to larger networks and many training patterns will result in a prohibitively long training process[34].

Although several ways of arriving at the required number of hidden nodes exist, the question of 'how many hidden nodes does a network need?' seems not to have a simple unique answer. Most of the research work in this area indicates that the size of the network should vary dynamically as more and more training patterns are provided. Therefore by allowing the network to shrink or expand during the training phase, the network not only learns to adapt its weight from the training data but also it attempts to adapt its size for a specific task through training. As yet, no results have emerged from the above proposal[37].

97
5.9.2 Selecting the Learning Constant

Apart from the appropriate size of a network, the success of BP algorithm significantly depends on the value of learning constant $\eta$. The optimum value for $\eta$ depends on the problem being solved[38].

In various parts of error surface the magnitude of the partial derivative of error with respect to a weight may be such that modifying a weight by a constant proportion of that derivative will yield a minor reduction in the error measure. This can occur in situations when the error surface is fairly flat along a weight dimension. Thus the value of the weight is adjusted by a small amount and many steps will be required to achieve a significant reduction in error. Hence for error surfaces with shallow gradient one requires a large value of $\eta$ to ensure rapid convergence.

Alternatively, where the error surface is steep along a weight dimension, the derivative of the weight will be large in magnitude. Thus if the weights are adjusted by a large amount, this will cause the weights to overshoot and oscillate along the minimum of the error surface. Hence, for areas with steep gradient a small value of $\eta$ must be chosen to avoid overshooting of the solution. This makes clear the fact that $\eta$ must either be chosen experimentally for each problem or be allowed to vary during training time.

Experimental observation made by Huang et al[27] and Peeling et al[39] suggest that it is usually best to start training with value of $\eta$ at mid range (i.e 0.5) and decide to stop training either when the learning taking place is very slow (very small reductions in error over a large number of iterations) or when oscillations in the error measurements are observed. If either case occurs, the value of $\eta$ is increased or decreased appropriately and the network is retrained.

More effective ways of selecting and adjusting $\eta$ for faster convergence have been proposed by other researchers. These methods and other variations to the original BP algorithm are discussed in later sections of this chapter.
5.10 Backpropagation Error Surfaces

The error surface of a BP network is defined in an (n+1) dimensional vector space where n is the number of weights used in a network.

As discussed earlier BP training used with MLNs has the property that, given any starting point on the error surface that is not the minimum, the learning rule will modify the weight vectors so that the sum of squared errors will decrease, and eventually drive the network to a minimum of the task.

The shape of BP error surfaces is largely a mystery. However from experience and rigorous experimental studies three basic problems have emerged[37].

First, because of the combinatorial permutation of the weights that leave the network input-output relation unchanged, effectively there can exist a large number of global minima in the error surface (which may lie at infinity for some problems)[37]. This causes the error surface to be highly degenerate and have numerous troughs.

Secondly, it has been shown that many BP error surfaces are dominated by flat areas or have a multitude of areas with shallow gradient in multiple dimensions simultaneously (also known as plateaus). These typically occur because particular combinations of weights cause the weighted sum of inputs (prior to sigmoid output) to be very large in magnitude. When this occurs the output of a node (or nodes) and hence the value of the error would be insensitive to small weight changes, since these simply move the weighted sum value back and forth along one of the shallow tails of the sigmoid function.

In these circumstances BP algorithm results in extremely small adjustments to the weights to reduce the error, hence often a great deal of numerical precision and large number of iterations must be employed to make significant progress.

The third problem is the existence of local minima. These are true of minima on the error surface where the level of error is higher than that of global minima. Since BP is based on gradient descent, the network would fail to move out to a global solution and hence stagnate in the local minima where the error level is high.

Beyond these three facts, little is known. For example, the number of local minima,
compared to the number of global minima, and whether local minima are excluded from regions near the global minima, are some of the unanswered questions regarding the shape and nature of error surfaces produced by BP. Clearly more research into error surfaces is needed.

5.11 Variations of BP

Since the discovery of BP, variations that attempt to improve its performance have been proposed by many researchers. These variations mainly address the problem of avoiding getting trapped in poor local minima and suggest ways to speed up the learning rate in multilayer network training. Many variations to BP exist, but due to the sheer size of the literature only a few popular methods have been selected and discussed here.

5.11.1 The Momentum Method

In the BP learning procedure, the speed of convergence is governed by the learning rate \( \eta \). In practical purposes one must choose a learning rate that is as large as possible without leading to oscillations. This offers the most rapid learning.

One way to increase the learning rate without oscillations is suggested by Rumelhart et al[2]. In this variation the expression for weight update is modified to include what is known as a momentum term. This is accomplished by the following rule:

\[
\Delta w_{ij}(n) = -\eta \delta_j y_i + \alpha \Delta w_{ij}(n-1)
\]

where \( n \) denotes the \( n^{th} \) iterating cycle and \( \alpha \) is the momentum constant \( (0 \leq \alpha < 1) \) which determines the effect of past weight changes on the current direction of movement in the weight space[2]. Thus the second term in the above expression indicates that the change in weights at the \( n^{th} \) cycle should be somewhat similar to change undertaken at the \( (n-1)^{th} \) cycle. In this way some inertia is built in, and
momentum in the rate of change is conserved to some degree[2]. This is useful in error surfaces containing long ravines that are characterised by sharp curvature across the ravine and a gently sloping floor. The sharp curvature tends to cause oscillation across the ravine. To prevent oscillation it is necessary to reduce the learning rate, but this causes very slow progress along the ravine. In such cases the momentum filters out the effect of high curvature and thus allows the effective learning rate to be bigger[2].

The momentum version of BP has been frequently used due to its simplicity and effectiveness, and so far it has been rigorously tested and has proved to be successful by many researchers in various applications.

5.11.2 Adaptive Learning Rate

As discussed in earlier sections, BP error surfaces contain many plateaus and ravines. This indicates that with a constant learning rate it will be very time consuming to move across these areas. Thus these properties indicate the usefulness of the variable learning rate to ensure rapid movements through various parts of the error surface.

Jacob[40] has studied these types of error surfaces and suggested four heuristics to improve the speed of learning. These are:

1) Every weight should have its own individual learning rate.
2) Every Learning rate should be allowed to vary over time.
3) When a weight derivative possesses the same sign for several consecutive steps (along a flat error region eg along a ravine) the learning rate for that weight must be increased.
4) When the sign of a weight derivative alternates for several consecutive steps, (along a highly curved error surface, eg across a ravine) the learning rate for that parameter must be decreased.

Simulation studies carried out by Jacob on simple binary problems, have shown improvement in the convergence rate by a factor of 1.2 to 6 (depending on the problem) in comparison to BP with momentum. However it was noted that out of 25 simulation attempts, for the Xor task, only 23 times, the solution was reached.
with comparison to 25 for BP with momentum. Other drawbacks of Jacob’s proposal are that it requires more computation for weight update and also it introduces more parameters (excluding the learning rates) to be optimised.

5.11.3 Application of Simulated Annealing to BP

Simulated annealing is a function minimisation algorithm\[41\] which allows for the possibility of increasing the energy (error) during the optimisation procedure. This is called simulated annealing since it copies the physical procedure called annealing. For example in a metal raised to a temperature above its melting point, the atoms are in rapid random motion. As with all physical systems, the atoms tend towards minimum energy state, but at high temperature the vigour of the atomic motion prevents this. As a metal is gradually cooled, lower and lower energy states are assumed until finally the lowest state, a global minimum, is achieved.

This gradual cooling process is analogous to a function minimisation algorithm which allows escape from local minima and attempts to find a global minimum of an error function.

The algorithm constructs a search path in the weight space. At each optimisation step, random deviations from the last point of the search path are generated, and these deviations are considered as possible weight adjustments. If these adjustments produce a reduction in the error function the weight adjustments would be accepted, a new point would be added to the search path and the process continues by considering random deviations from the new point. If however the adjustments to the weights cause an increase to the error function, the weight adjustments would be accepted according to a probability derived from a distribution (commonly the Boltzman distribution).

The probability of acceptance of increase in the error function is governed by the ‘cooling’ schedule (or temperature) which has to be reduced very slowly (reducing the acceptance probability) for the algorithm to converge to the global minimum.

In general, simulated annealing converges to a global minimum, but this is achieved at a high computational cost.
There have been attempts to modify the neural network algorithm to combine the hill-climbing benefits of simulated annealing with the relatively quick convergence of BP. This general approach has been proposed and investigated by Hoptroff[42], and is usually referred to as the diffusion method.

The diffusion method involves the combination of random fluctuations and gradient descent for the calculation of adjustments to the weights. Such a process in this context can be expressed as:

\[ \Delta w_{ij}(n) = -\eta \delta_j y_i + \sqrt{2T} \ R \]  

Where T is temperature and R represents the random fluctuations.

With the diffusion method one hopes that the early and large random fluctuations will allow the system to quickly escape from local minima, whereas the later behaviour, as temperature approaches zero, will be essentially a gradient descent into a prominent minimum.

Hoptroff has experimentally compared diffusion method to BP and simulated annealing for a binary task (shift register). Preliminary results reported by Hoptroff[42] indicate that diffusion method is robust and more computationally efficient than both BP and simulated annealing. However, the success of the diffusion learning method is mostly dependent on the appropriate selection of the initial temperature. If the initial temperature is too low, then, diffusion would be equivalent to BP. If the temperature is too high, the diffusion may take a long time to converge. Only at ‘intermediate’ temperatures one can obtain possible improvements.

The reported work by Hoptroff[42] was only a preliminary study which needs to be rigorously tested further for more practical and complex classification tasks.

5.11.4 Improvement by Detection of Learning Standstill

It is known that the BP error surface contains many regions with shallow gradients. In such areas adjustments to the weights do not significantly reduce the overall system error, hence many iterations would be required to pass through these types of
regions. In such cases the network stays in a learning standstill. By observing the transition of correction weights and that of output values on every layer, Yamada et al[43] found that a learning standstill occurs when all output values on one layer are nearly 1 or 0, that is the weighted sum of the input is lying on either edge of the sigmoid function. In order to solve this problem Yamada et al[43] have introduced a variable slant parameter in their sigmoid function:

\[
f(x) = \frac{1}{2} \left( 1 + \tanh \frac{x}{u_0} \right)
\]

where \( f(x) \) is the differentiable sigmoid function, \( x \) is the weighted sum of inputs and \( u_0 \) is controlled slant parameter.

In this version of BP algorithm a learning standstill is first detected, and then the slant parameter \( u_0 \) is enlarged so that output values on any layer are far from 0 and 1. Effect of varying the slant parameter is shown in Fig (5.8).

Yamada et al[43] have tested their algorithm on recognition of handwritten numerals and have reported 1% improvements in comparison to BP with momentum. This algorithm introduces extra computational effort since it requires to detect the network standstill by monitoring the outputs and weight adjustments.

### 5.11.5 High Order Correlation

Several efforts have been made to incorporate high-order correlation into this already inherently nonlinear function optimising of BP, in order to speed up the learning process. One approach is due to Rumelhart et al[2], which consists of incorporating so called Sigma-pi nodes (neuron) in the multilayer networks.

With Sigma-pi nodes the function of neuron is extended, so that rather than just summing over input terms, it can include higher-order terms that are multiples of input from previous layer:
\[ y_i = f \left( \sum_j w_{ij} \prod_k y_{jk} \right) \]  \hspace{1cm} (5.17)

where \( f \) is a sigmoid function, \( j \) varies over the set of conjuncts feeding into node \( i \) and \( k \) varies over the nodes of the \( j \)th conjunct[2]. Rumelhart et al [2] have applied the BP learning rule to train networks containing Sigma-pi nodes, but they have not reported any experimental applications regarding the Sigma-pi nodes.

An alternative approach is to form an \( n \)th order polynomial of input data, and use this form of higher dimensional input (the high order terms in the polynomial are treated as extra input to the network) to train a single layer network. A second order example of such network for a two input and two class training data is shown in Fig (5.9).

The output of an \( n \)th order network is given by:

\[ y = f \left( w_0 + \sum_j w_j x_j + \sum_j \sum_k w_{kj} x_j x_k + \cdots \right) \]  \hspace{1cm} (5.18)

Where \( y \) is the output, \( f \) is the sigmoid function, \( w_0 \) and \( w_j \) are the linear weights and \( w_{kj} \) is the higher order weight capturing the high order correlations.

In this type of network, since there are no hidden nodes to be trained, very fast convergence can be obtained. Experiments on the binary Xor problem carried out by Pao[21] show an improvement factor of 4 in the convergence rate with comparison to BP. However the order of the polynomial network must be matched by the order of the problem. If high order polynomials are used (higher than is necessary to solve a problem) then this can lead to an excessive number of parameter estimations and hence poor generalisation, and the solution is very sensitive to the size of the network.

5.11.6 Minkowski-r BP

The standard BP is derived by minimising the sum of squared errors as a function of connection weights. The model presented by Hanson and Burr[44] is an alternative
Fig (5.8) Sigmoid function with different slant parameters

Fig (5.9) Architecture of a second order network
objective function that generalises BP by using Minkowski-\( r \) power metrics. For small \( r \)'s (\( r = 1 \)) a 'city block' error metric is approximated, for large \( r \)'s the maximum or 'supermum' metric is approached, while \( r = 2 \) results in the standard BP error function.

The error for Minkowski-\( r \) power metrics is given by:

\[
E = \frac{1}{r} \sum_{i} (|y_i - d_i|)^r
\]

Using the BP learning rule it can be shown that\([44] \) the expression for the gradients of an output node \( j \) in the general Minkowski-\( r \) case is given by:

\[
\delta_j = (|y_j - d_j|)^{r-1} y_j (1 - y_j) \text{sgn}(y_j - d_j)
\]

This gradient is used in the BP weight update rule, equation(5.8). Since the gradient computed for a hidden layer node is a function of the gradient for the output layer nodes, the hidden layer weight updating proceeds in the same way as in the original BP.

Hanson and Burr\([44] \) have applied this method to speech recognition problems. They observed how varying \( r \) effects the convergent properties of a network. As \( r \) is increased, convergence time tends to grow linearly and consequently decreasing \( r \) improves the convergence without much change in the nature of the solution\([44] \). Also they have shown that when dealing with small training set size with large \( r \)'s a network generalises better in comparison to BP, but this is at a cost of longer convergence time.

5.11.7 Other Variations

Many other variations to the original BP learning algorithm exist. For example Parker\([45] \) has investigated the use of second-order algorithms, that is to include the second derivative of the weights in the learning algorithm. He has shown that the second order algorithm implements an efficient network approximation to a
minimisation technique called Newton's method\[45\]. However Parker's second-order algorithm introduces twice as many arithmetic operations in comparison to the original BP.

Other researchers have studied the application of non-linear optimisation techniques to multilayer networks. Watrous\[46\], Webb et al\[47\] and Salas et al\[48\] have all investigated the stability, computational complexity and convergence properties of the conjugate gradient method.

On experiments with binary tasks Salas et al\[48\] have shown that, for moderately small tasks, such methods reduce the convergence rate considerably, but this again at the cost of more computational effort and time per iteration.

All the variations reported here so far can be classified into three groups. These are:

a) methods which modify the search for the minimum of the error function (eg, the Diffusion method\[42\]).  
b) methods which change the error function (eg, Minkowski-r method\[44\]).  
c) methods which change the node structure or formulation of the network (eg, High order correlation method\[2\]).

Although the variations of BP reported so far reduce the convergence rate for small problems, nevertheless this is usually achieved by inclusion of more complex learning algorithms which effectively increase the computational time and in many cases introduce more learning parameters to be optimised (this also means extra storage).

Hence there is no easy way of recommending one technique over another. Many additional tests and research will have to be performed before one can get solid evidence that one technique is superior to others.

Despite this, among these modifications reported here, by far the most widely used version of BP is the BP with momentum which has been successfully applied by many researchers in various fields. This is mainly due to its simplicity and success rate reported in the literature\[3\].

Therefore BP with momentum has provided a common base of comparison for various types of learning procedures and hence it is implemented and compared with

108
the new learning algorithm proposed in a subsequent chapter of this thesis.

5.12 Simulations of BP Training Networks

The BP algorithm with momentum was implemented in software for investigation of its learning capabilities. The software was written in Pascal and run on Sun 3/60 workstations. Using this software experiments were conducted to show the effect of varying the learning rate \( \eta \) and the momentum term \( \alpha \) for classic problems existing in the neural networks literature and some synthetic data.

5.12.1 Experimental Data and Network Architecture

Four data sets were used in these simulations, details of which are as follows:

a) **Exclusive Or task:**

This is a classic problem where a simple Perceptron or a linear classifier fails to separate this non-linearly separable data set. The training data is a two class case consisting of four training patterns (two per pattern class). Table(5.1) shows the desired relationship between input and outputs, where those input combinations that should produce a -1 output are labelled as \( C_1 \), and those producing 1 are labelled as \( C_2 \). Fig(5.10) shows Xor data distribution and a typical linear decision boundary generated by a LDF. As illustrated in Fig(5.10), no combination of weight values for generating a linear decision boundary is capable of separating the Xor data. Introducing a hidden layer allows the network to yield an appropriate solution to this problem. The optimal network architecture for solving the Xor task is shown in Fig.(5.11) which consists of two input nodes, two hidden nodes and a single output node.

b) **Multiplexer task:**

For the multiplexer task, the network architecture consists of six input nodes, six hidden nodes and an output node. The input to the network is four data lines and two...
<table>
<thead>
<tr>
<th>Class</th>
<th>Input 1 $x_1$</th>
<th>Input 2 $x_2$</th>
<th>Output $d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1$</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>$C_2$</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>$C_2$</td>
<td>-1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$C_1$</td>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>

Table (5.1) Xor data

Fig.(5.10) Xor data distribution
address lines. The address lines indicate one of the data lines using a binary code. The desired output of the system is the input to the data line indicated by the address lines. This data set contains 64 training patterns and is a two-class non-linearly separable problem.

c) Uniform distribution:

This data set was generated using the Sun 3/60 random number generator. The data set is a two-class case containing 100 one-dimensional data points and is marginally separable. Class A spans from -1 to 1 and class B spans from 1 to 5. The distribution for this experiment is shown in Fig.(5.11). A simple network architecture, one input node and one output node, is sufficient for solving this problem, therefore it was not necessary to use a network containing hidden layers for this task.

d) Bimodal Uniform

This data set is a one-dimensional two-class uniform distribution that forms two overlapping regions. The data set contains 100 training patterns. Class A extends from -5 to -1, 1 to 5 and class B from -2 to 2. The data distribution for this experiment is shown in Fig.(5.12). The nature of the decision boundary required to solve this problem necessitates a network with minimum architecture of one input node, two hidden nodes and an output node.

5.12.2 Experiments on Learning Rate and Momentum Term

Experiments were conducted to investigate the effects of different values of the learning rate and the momentum term on the convergence property of a network.

In the experiments throughout this thesis, starting weights were generated using random numbers between ±0.5 so to avoid the weight symmetry problem. To reduce the dependency of a network on random starting weights, the networks are trained using ten different starting weights. The average results for the data sets described in previous section with 25 different pairs of η and α values are presented in the next subsection.
Fig. (5.11) Data distribution for the experiment case (c) with one input and two classes.

Fig. (5.12) Data distribution for the experiment case (d) with one input and two classes.
The termination criterion for the separable data sets was to detect for correct classification of all training patterns and for the non-separable case (d) it was decided to stop training after a predetermined number of iterations (1000).

5.12.3 Results

A summary of the results for the separable data sets (a-c) is shown in tables (5.2) to (5.4) and table (5.5) contains the results for the overlapping data set (d). Graphs (5.1) to (5.4) show typical learning curves for two different sets of \( \eta \) and \( \alpha \). The contents of tables (5.2) to (5.4) indicate the number of iterations required for the network to establish a separable decision boundary (i.e., correct classification of all data points). For the non-separable case, table (5.5), the contents of the table show the number of misclassified patterns after 1000 iterations.

From the results shown, it can be seen that for each problem the value of \( \eta \) and \( \alpha \) must be selected carefully in order to ensure rapid convergence of networks. However, it is evident from the results that \( \eta \) and \( \alpha \) must be chosen experimentally for each individual problem. But in general, a small value of \( \eta \) and a large value of \( \alpha \) leads to a fast convergence rate. This fact is supported by the learning graphs of (5.1) to (5.4), where in all the cases \( \alpha = 0.8 \) produced a faster convergence than \( \alpha = 0.5 \) for a fixed \( \eta \) (\( \eta = 0.5 \)).

The results reported here show the effect of \( \eta \) and \( \alpha \) for various types of data sets. A detailed discussion and performance evaluation of BP networks is presented in the next chapter where a new learning algorithm based on variation of the error function is proposed. It will be shown through computer simulations that the new algorithm converges significantly faster and also it requires fewer calculations per training cycle.
Table (5.2) Number of iterations required to perform perfect classification of Xor data.

<table>
<thead>
<tr>
<th>η</th>
<th>α</th>
<th>0.10</th>
<th>0.25</th>
<th>0.50</th>
<th>0.75</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.10</td>
<td>&gt;1000</td>
<td>&gt;1000</td>
<td>&gt;1000</td>
<td>&gt;1000</td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>0.25</td>
<td>&gt;1000</td>
<td>&gt;1000</td>
<td>&gt;1000</td>
<td></td>
<td>537</td>
<td>164</td>
</tr>
<tr>
<td>0.50</td>
<td></td>
<td>903</td>
<td>757</td>
<td>515</td>
<td>288</td>
<td>119</td>
</tr>
<tr>
<td>0.75</td>
<td></td>
<td>605</td>
<td>508</td>
<td>350</td>
<td>205</td>
<td>97</td>
</tr>
<tr>
<td>1.00</td>
<td></td>
<td>456</td>
<td>384</td>
<td>267</td>
<td>164</td>
<td>85</td>
</tr>
</tbody>
</table>

Graph (5.1) Typical learning curve for Xor task,
η = 0.5.

114
Table (5.3) Number of iterations required to perform perfect classification of Multiplexer data.

<table>
<thead>
<tr>
<th>$\eta$</th>
<th>0.10</th>
<th>0.25</th>
<th>0.50</th>
<th>0.75</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.10</td>
<td>240</td>
<td>202</td>
<td>143</td>
<td>87</td>
<td>&gt;1000</td>
</tr>
<tr>
<td>0.25</td>
<td>93</td>
<td>82</td>
<td>63</td>
<td>42</td>
<td>&gt;1000</td>
</tr>
<tr>
<td>0.50</td>
<td>246</td>
<td>81</td>
<td>40</td>
<td>34</td>
<td>&gt;1000</td>
</tr>
<tr>
<td>0.75</td>
<td>104</td>
<td>122</td>
<td>70</td>
<td>93</td>
<td>&gt;1000</td>
</tr>
<tr>
<td>1.00</td>
<td>106</td>
<td>93</td>
<td>376</td>
<td>&gt;1000</td>
<td>&gt;1000</td>
</tr>
</tbody>
</table>

Graph (5.2) Typical learning curve for the Multiplexer task
$\eta = 0.5$. 

$\alpha = 0.5$
$\alpha = 0.8$
Table (5.4) Number of iterations required to perform perfect classification of Separable Uniform data.

<table>
<thead>
<tr>
<th>$\eta$</th>
<th>$\alpha$</th>
<th>0.10</th>
<th>0.25</th>
<th>0.50</th>
<th>0.75</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.10</td>
<td></td>
<td>586</td>
<td>490</td>
<td>331</td>
<td>172</td>
<td>51</td>
</tr>
<tr>
<td>0.25</td>
<td></td>
<td>236</td>
<td>199</td>
<td>136</td>
<td>68</td>
<td>29</td>
</tr>
<tr>
<td>0.50</td>
<td>&gt;1000</td>
<td></td>
<td>&gt;1000</td>
<td>&gt;1000</td>
<td>99</td>
<td>24</td>
</tr>
<tr>
<td>0.75</td>
<td>&gt;1000</td>
<td>&gt;1000</td>
<td>&gt;1000</td>
<td>&gt;1000</td>
<td>&gt;1000</td>
<td></td>
</tr>
<tr>
<td>1.00</td>
<td>&gt;1000</td>
<td>355</td>
<td>23</td>
<td>417</td>
<td>&gt;1000</td>
<td></td>
</tr>
</tbody>
</table>

Graph (5.3) Typical learning curve for the Separable Uniform task, $\eta = 0.5$. 

Graph (5.3) Typical learning curve for the Separable Uniform task, $\eta = 0.5$. 

116
Table (5.5) Total errors from Bimodal Uniform data set after 1000 iterations.

<table>
<thead>
<tr>
<th>$\eta$</th>
<th>$\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.10</td>
</tr>
<tr>
<td>0.10</td>
<td>18</td>
</tr>
<tr>
<td>0.25</td>
<td>19</td>
</tr>
<tr>
<td>0.50</td>
<td>18</td>
</tr>
<tr>
<td>0.75</td>
<td>39</td>
</tr>
<tr>
<td>1.00</td>
<td>39</td>
</tr>
</tbody>
</table>

Graph (5.4) Typical learning curve for Bimodal Uniform data, $\eta = 0.5$. $\alpha = 0.5, \alpha = 0.8$.
5.13 Conclusion

A brief introduction to neural networks including the formulation and analysis of BP networks has been presented. It was shown that BP is a powerful learning method capable of producing good generalisation that can be used to train multilayer networks to generate complex decision surfaces. The success of BP partly depends on an appropriate architecture of the network (number of layers and number of hidden nodes within each layer) and the value of learning constant.

Despite its success BP suffers from slow convergence and a tendency to get trapped in local minima. Variation of BP addressing these problems have been presented. These variations improve the performance of BP but this is usually achieved by inclusion of more complex and computationally expensive learning rules.

Finally simulation studies of BP with momentum have been applied to problems such as binary Xor and multiplexer tasks. Because of their non-linearly separable nature these problems were impossible to solve using a single layer network or a linear discriminant function. The simulations also reported on the effect of the momentum and learning constants on the convergence of BP networks. It was shown that to ensure rapid convergence of a network $\eta$ and $\alpha$ must be selected with care.

These and other properties of BP network are investigated and compared with a new learning method in the next chapter.
References

17. S.Grossberg, 'Non-linear Neural Networks: Principles, Mechanism and

121
Chapter 6

Empirical Maximum Likelihood Learning

6.1 Introduction

In the previous chapter some of the capabilities of the backpropagated neural network training methods have been presented. Through computer simulations it was shown that problems which were difficult or impossible to solve by conventional classifiers can now be readily solved by the popular BP training method, applied to multilayer networks.

One of the major drawbacks of BP training is its long convergence time to reach a satisfactory solution[1,2]. Various modifications and improvements to BP have been reported, but as discussed in previous chapter, they either require more computation per iteration, due to their complexity, or extra storage since they introduce new parameters to be optimised.

In this chapter an alternative approach, based on a reformulation of the BP algorithm in terms of an ‘Empirical Maximum Likelihood’ (EML) criterion, is
proposed[3]. The word empirical is used since this method does not assume a prior form of distribution for each class of data (as is the case for a maximum likelihood Gaussian classifier, for example). The principle of this approach lies in reformulation of the training task in terms of probabilities, and hence the introduction of a new cost function to be optimised. In the following, first the formulation of the alternative learning rule is presented, followed by computer simulation comparisons with the conventional formulation.

It will be shown that the EML method increases the proportion of cases where the method converges to a near optimum solution, without increasing the computational complexity. Moreover the number of iterations required for convergence is often found to decrease and the amount of calculations in each iteration is also reduced.

Further computer simulations will also illustrate that the error landscape produced by this method is much simpler than BP, and also the generalisation ability of the network shows considerable improvement over BP.

6.2 Formulation of EML Algorithm

In BP, given a set of input vectors with their associated desired output values, the training method attempts to minimise the error function \( E \) by the gradient descent method. The most common choice of error function is the sum-of-squared distance between desired output value and the actual output value produced by the network.

In EML, the values of the output vectors are defined as probabilities. Thus instead of minimising an error function, EML attempts to maximise the likelihood or probability of the training sequence[4,5,6].

BP is equivalent to EML provided that the observed responses can be assumed to have approximately normal distribution[7]. However in the training of neural network classifiers this is hardly the case, since the response of a network usually takes just two values, corresponding to membership or otherwise of a given class.

For a given pattern \( k \), \( y_{jk} \) can be assumed to be the probability that the pattern belongs to the class \( j \), and \( 1 - y_{jk} \) is otherwise. \( d_{jk} \) and \( 1 - d_{jk} \) respectively are the probability target values associated with the input vectors. Hence, assuming
independence of the observations, the probability of all the training data being correct will be given by:

\[
P = \prod_k \prod_j \left\{ d_{jk} y_{jk} + (1 - d_{jk})(1 - y_{jk}) \right\} \tag{6. 1}
\]

The proposed method maximises the log likelihood given by:

\[
L = \log P = \sum_k \sum_j \log \left\{ d_{jk} y_{jk} + (1 - d_{jk})(1 - y_{jk}) \right\} \tag{6. 2}
\]

The above expression is known as the log likelihood function, which often arises in physics and information theory, and is generally interpreted as an entropy[8,9].

A maximum likelihood estimation of \( P \) is defined as the global minimum of the negative log likelihood function, i.e, \( L \) is negated to obtain an error function to be minimised given by:

\[
E = -L = - \sum_k \sum_j \left\{ d_{jk} \log y_{jk} + (1 - d_{jk}) \log (1 - y_{jk}) \right\} \tag{6. 3}
\]

where the rearrangement within the summation is possible because \( d_{jk} \) is always either 0 or 1.

The above cost function can now be minimised using the gradient descent method. Applying this method to the log-likelihood cost function leads to a revised backpropagation algorithm[10,11] which is now described.

For EML the feedforward formulae remain the same as for BP(section 5.6), omitted from this section since the classification function of the network is unchanged in the new formulation.

The calculation of error derivatives \( \delta_j \) (or the backpropagated errors) for the units at the output level explicitly depends on the form of the cost function. Using the
weight update, expression (5.7), and the log-likelihood cost function, $\delta_j$ for an output node $j$ for a single pattern can be calculated using the chain rule:

$$\delta_j = \frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial x_j}$$

$$\frac{\partial E}{\partial y_j} = \frac{y_j - d_j}{y_j (1 - y_j)}$$

$$\frac{\partial y_j}{\partial x_j} = y_j (1 - y_j)$$

Using the above $\delta_j$ can be expressed as:

$$\delta_j = y_j - d_j \quad (6.4)$$

using the same principle, for a hidden unit $i$:

$$\delta_i = \frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial y_i} \cdot \frac{\partial y_i}{\partial x_i}$$

$$\frac{\partial y_i}{\partial x_i} = y_i (1 - y_i)$$

$$\frac{\partial E}{\partial y_i} = \sum_j \left\{ \frac{\partial E}{\partial x_j} \cdot \frac{\partial x_j}{\partial y_i} \right\} = \sum_j \left\{ \frac{\partial E}{\partial x_j} \cdot w_{ij} \right\}$$

$$= \sum_j \delta_j w_{ij}$$

Where $j$ represents all outputs emerging from the unit $i$. Hence $\delta_i$ is given by:

$$\delta_i = y_i (1 - y_i) \sum_j \delta_j w_{ij} \quad (6.5)$$
The main steps of the EML batch training algorithm are summarised as follows:

1) Initialise all $w_{ij}$ to small random values (usually between ± 0.5)

2) For each pattern in training set calculate: (Forward pass)

$$x_j = \sum_i w_{ij} y_i + w_{0j}$$

$$y_j = f(x_j) = \frac{1}{1 + e^{-x_j}}$$

3) For each pattern compute $\delta_j$: (Backward pass)

   for an output node $\delta_j$ is
   $$\delta_j = y_j - d_j$$

   for a hidden layer node $\delta_j$ is
   $$\delta_j = y_j (1 - y_j) \sum_k \delta_k w_{jk}$$

   where $k$ is over all nodes in the layer above node $j$.

4) Adjust each weight by: (Weight update)

   $$w_{ij} (n + 1) = w_{ij} (n) - \eta \delta_j y_i + \alpha \Delta w_{ij} (n - 1)$$

   where $n$ denotes the iteration index.

5) Repeat steps 2 to 4 until a convergence criterion is satisfied.

It can be seen from expression (5.8), $\delta_j$ of BP, and expression (6.4), $\delta_j$ of EML, that the differences between the formulations lies solely in the evaluation of these terms at the output nodes. Since it is not necessary to calculate the cost function in the course of the backpropagated learning, the only computational difference is the
dropping of the term $y_j (1 - y_j)$ from the formula for the $\delta_j$ for the output nodes. The consequences of this modification are three fold:

Firstly, compared with the standard formulation, there is a saving of two floating point multiplications per training sample in each iteration, in the computations associated with the learning process for a two class problem. If there are more than two classes, the number of multiplications saved is multiplied by the number of outputs from the network.

Secondly, the magnitude of the error derivatives at the output layer is increased by a factor of at least 4 (see section 6.3.1), compared with the BP, so it is appropriate to reduce the learning rate $\eta$ by a similar factor when applying the EML.

Thirdly, the term $y_j (1 - y_j)$ in BP is effectively a bell-shaped weight function applied to each point's contribution to the gradient, which achieves its maximum for samples in the feature space which are on the decision boundary, and which tails off towards zero as the decision boundary moves away from the sample, even though the sample may become misclassified in the process.

This accounts for a phenomenon often observed in training multilayer networks, that there are large regions of the weight space far removed from the global minimum in which the error function is plateau-shaped. When the weight vector starts in, or wanders into, such a region, BP as discussed in previous chapter, may adjust the weight very slowly, or worse still stop completely because the gradients are so small as to be beyond the numerical range of the computer[12].

In the EML formulation however, the unweighted gradient $y_j - d_j$ approaches unit magnitude as the decision boundary moves further away from a misclassified sample, and consequently such plateau regions should not occur[13].

These assertions will now be supported by various computer simulations. In these simulations the effects of sample size, speed of convergence and generalisation capability of both the learning methods are investigated.
6.3 Simulations

In this section the results of computer simulations comparing the properties of BP and EML are presented. The first test (section 6.3.1) compares the learning factors between the two methods on a well known binary task.

In section 6.3.2 a comparison between error surfaces generated by BP and EML cost functions will be presented.

Effects of random starting weights for both the algorithms are investigated in section (6.3.3).

This is followed by a performance assessment of BP and EML to produce a complex decision boundary.

The speed of convergence, generalisation and the effects of training set size are also investigated.

6.3.1 Multiplexer Task

This task is used to derive a heuristic relationship between BP and EML learning constants. As can be seen from equation (6.4) the difference between output derivatives lies in the factor \( y_j(1 - y_j) \). Since the sigmoid function constrains \( y_j \) to lie between 0 and 1, \( y_j(1 - y_j) \) is a non-negative bell-shaped function, whose maximum value is 0.25 (at \( y_j = 0.5 \)). Thus the magnitude of \( \delta_j \) is increased by a factor of at least 4 as a result of the EML. Since the gradient terms \( \partial E / \partial w_{ij} \) are linear combinations of \( \partial E / \partial x_j \), it would seem appropriate to reduce the EML learning constant by a similar factor, in order that typical adjustments to the weights should have a similar magnitude in the two weight update formulations so a fair comparison of learning speed can be conducted.

Experimental results will show that reducing the EML learning constant by a factor of 4, while retaining the value of the momentum term, consistently results in successful implementations of the EML formulation.
This is illustrated by using the multiplexer task[14] (the details of the data and the network architecture have already been described in section(5.12.1)). The multiplexer network was trained using $\eta = 0.4$ for BP and $\eta = 0.1$ for EML with $\alpha = 0.75$ for both cases, until convergence was reached (zero misclassifications).

A typical learning curve for this task is shown in graph(6.1). This graph clearly indicates the faster convergence of the EML over BP. The zero misclassification rate for BP occurs after 60 iterations as compared with 40 iterations for EML.

Further experimental results are shown in graph (6.2) to graph (6.5). These graphs illustrate the number of iterations required for convergence in repeated training of the multiplexer network. The same randomly selected starting weights were used each time, but the learning rates $\eta$ and momentum term $\alpha$ were adjusted systematically between trials. For the EML formulation the range and increment of $\eta$ was reduced by a factor of 4.

Comparing these graphs reveals that the fastest convergence is achieved when $\alpha = 0.85$ and $\eta = 0.4$ using BP, and $\alpha = 0.85$ and $\eta = 0.1$ using EML, thus the optimum value of $\eta$ is reduced by a factor of 4. This rule applies to other properties of $\eta$, for example, the ranges of $\eta$ giving convergence within 300 iterations, for all $\alpha$ tested, are 0.1 to 0.5 in graph(6.2), and 0.025 to 0.125 in graph(6.3), i.e the operational range of $\eta$ is reduced by a factor of 4 once again.

This example is a typical illustration of the relationship between the BP and EML learning constants. Other experiments conducted on various data sets revealed the same property. Hence for fair assessments of learning speeds between these two learning methods, in all the experiments throughout this chapter, the learning constant for the EML is divided by factor of 4.
Graph (6.1) Example of learning curves for the Multiplexer task, 
$\eta_{EML}=0.1$, $\eta_{BP}=0.4$ and $\alpha=0.75$ for both cases.
Graph (6.2) Convergence to perfect classification for the Multiplexer task using BP.

Graph (6.3) Convergence to perfect classification for the Multiplexer task using EML.
Graph (6.4) Convergence to perfect classification for the Multiplexer task using BP.

Graph (6.5) Convergence to perfect classification for the Multiplexer task using EML.
6.3.2 Comparison of Error Surfaces

This example is used to illustrate and compare the error surfaces generated by the BP and the EML cost functions. This is illustrated with a simple abstract example.

Consider a classification problem in which there is a single feature \( y \) and two classes (1 and 2). Let the training set for this problem consist of the two following six samples. Class1: \( y = 0.0, 0.1, 0.3 \) and Class2: \( y = 0.2, 0.4, 0.5 \). A single layer network used here consisted of a single node having two weights \( w_0 \) and \( w_1 \). Graph (6.6) is a contour plot of the BP cost function against the two weights. The surface consists of a deep ravine containing the global minimum, and surrounded by plateaus on two sides and a hill at one end.

Graph (6.7) shows the EML cost function plotted against the two weights over the same ranges. Again the global minimum lies in a deep ravine, but in this case the entire surface is an elongated bowl shape and contains no plateaus within the plotted range.

In this simple example a gradient descent algorithm converges relatively quickly to the minimum of the EML function regardless of the starting point, but for BP, starting points outside the ravine are liable to lead to extremely slow convergence when the weight vectors either starts in, or wanders into, the plateau region[15]. The relatively simpler error surface generated by the EML cost function results in an improvement in the learning speed. Further simulations with respect to speed of convergence and generalisations of the EML method are shown in the later sections of this chapter.

6.3.3 Assessment of initial weights

The success and speed of backpropagated learning are also dependent on the initial values assigned to the weights. The usual practice is to select these at random from a uniform distribution in range of typically \( \pm 0.3 \) or \( \pm 0.5 \)[16]. Within these ranges, the performance of the algorithm can vary considerably.

This is demonstrated by considering the repeated training of a network in which the
Graph (6.6) Contour plot of the BP cost function

Graph (6.7) Contour plot of the EML cost function
learning parameters are held constant but the initial weights are adjusted systematically between trials. The network used consisted of a single node with one input $y_1$ and one output $y_2$. Thus just two weights $w_0$ and $w_1$ are modified by the iterative procedure. Two cases of simulated training data are considered here; a) non-overlapping uniform distributions (marginally Separable case) b) overlapping triangular distribution. Both data sets consist of two classes whose inputs are randomly generated from the appropriate distributions. The distributions of these data sets are shown on Fig(6.1) and Fig(6.2). The NAG library random number generator with an appropriate transformation was used to produce these data sets.

For case (a), (Fig(6.1) non-overlapping uniform distribution), 160 samples with $y_1$ in the range -1 to 1 are coded with $d_2 = 1$, and 160 samples with $y_1$ in the range 1 to 5 are coded with $d_2 = 0$. The learning parameters were set to $\eta = 0.8$ for BP and $\eta = 0.2$ for the EML, and $\alpha = 0.85$ in both cases.

Table (6.1) gives the iterations required to obtain zero error in the training set. This shows a large region of the weight space where EML is an order of magnitude faster than its counterpart. In 14% of trials BP fails to converge in 999 iterations, whereas EML converges in all trials within maximum of 51 iterations.

For case (b), (Fig(6.2) overlapping triangular distribution), 160 samples with $y_1$ in the range -1 to 2 are coded with $d_2 = 1$, and 160 samples with $y_1$ in the range 0 to 6 are coded with $d_2 = 0$. The learning parameters were set to $\eta = 0.4$ for BP and $\eta = 0.1$ for EML, with $\alpha = 0.75$ in both cases. Table (6.2) to (6.4) shows the numbers of misclassifications at 20, 100 and 500 iterations. In the described training set the minimum number of misclassifications is 12 (3.75%).

At 20 iterations, (Table (6.2b)) EML has converged for 8 cases as compared with 1 for BP. In all the trials the EML has reduced the misclassifications to 16 (5%) or less, while with BP misclassification are 64 (20%) or more in 13 of 36 trials.

At 100 iterations, (Table (6.3)) the number of cases where EML has reached convergence is considerably larger than BP. EML has produced average of 3.84% misclassifications as compared with average of 10.38% for BP.
Fig. 6.1 Marginally Separable Uniform Distribution

Fig. 6.2 Overlapping triangular distribution
### Chapter 6: Empirical Maximum Likelihood Learning

<table>
<thead>
<tr>
<th>$w_1(0)$</th>
<th>$w_0(0)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-0.5</td>
</tr>
<tr>
<td>-0.5</td>
<td>39</td>
</tr>
<tr>
<td>-0.3</td>
<td>16</td>
</tr>
<tr>
<td>-0.1</td>
<td>23</td>
</tr>
<tr>
<td>0.1</td>
<td>237</td>
</tr>
<tr>
<td>0.3</td>
<td>972</td>
</tr>
<tr>
<td>0.5</td>
<td>381</td>
</tr>
</tbody>
</table>

Table (6.1a) Original BP with $\eta = 0.8$ and $\alpha = 0.85$

<table>
<thead>
<tr>
<th>$w_1(0)$</th>
<th>$w_0(0)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-0.5</td>
</tr>
<tr>
<td>-0.5</td>
<td>27</td>
</tr>
<tr>
<td>-0.3</td>
<td>16</td>
</tr>
<tr>
<td>-0.1</td>
<td>4</td>
</tr>
<tr>
<td>0.1</td>
<td>30</td>
</tr>
<tr>
<td>0.3</td>
<td>48</td>
</tr>
<tr>
<td>0.5</td>
<td>30</td>
</tr>
</tbody>
</table>

Table (6.1b) EML formulation with $\eta = 0.2$ and $\alpha = 0.85$

Table (6.1) Number of iterations required to reach zero error for the marginally separable uniform distribution.

137
<table>
<thead>
<tr>
<th>$w_1(0)$</th>
<th>$w_0(0)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-0.5</td>
</tr>
<tr>
<td>-0.5</td>
<td>28</td>
</tr>
<tr>
<td>-0.3</td>
<td>13</td>
</tr>
<tr>
<td>-0.1</td>
<td>38</td>
</tr>
<tr>
<td>0.1</td>
<td>55</td>
</tr>
<tr>
<td>0.3</td>
<td>65</td>
</tr>
<tr>
<td>0.5</td>
<td>53</td>
</tr>
</tbody>
</table>

Table (6.2a) Original BP with $\eta = 0.4$ and $\alpha = 0.75$

<table>
<thead>
<tr>
<th>$w_1(0)$</th>
<th>$w_0(0)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-0.5</td>
</tr>
<tr>
<td>-0.5</td>
<td>13</td>
</tr>
<tr>
<td>-0.3</td>
<td>13</td>
</tr>
<tr>
<td>-0.1</td>
<td>12</td>
</tr>
<tr>
<td>0.1</td>
<td>16</td>
</tr>
<tr>
<td>0.3</td>
<td>13</td>
</tr>
<tr>
<td>0.5</td>
<td>13</td>
</tr>
</tbody>
</table>

Table (6.2b) EML formulation with $\eta = 0.1$ and $\alpha = 0.75$

Table (6.2) Number of misclassified samples after 20 iterations for the overlapping triangular distribution.
<table>
<thead>
<tr>
<th>( w_1(0) )</th>
<th>( w_0(0) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-0.5)</td>
<td>20</td>
</tr>
<tr>
<td>(-0.3)</td>
<td>13</td>
</tr>
<tr>
<td>(-0.1)</td>
<td>20</td>
</tr>
<tr>
<td>(0.1)</td>
<td>51</td>
</tr>
<tr>
<td>(0.3)</td>
<td>53</td>
</tr>
<tr>
<td>(0.5)</td>
<td>42</td>
</tr>
</tbody>
</table>

Table (6.3a) Original BP with \( \eta = 0.4 \) and \( \alpha = 0.75 \)

<table>
<thead>
<tr>
<th>( w_1(0) )</th>
<th>( w_0(0) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-0.5)</td>
<td>12</td>
</tr>
<tr>
<td>(-0.3)</td>
<td>12</td>
</tr>
<tr>
<td>(-0.1)</td>
<td>12</td>
</tr>
<tr>
<td>(0.1)</td>
<td>13</td>
</tr>
<tr>
<td>(0.3)</td>
<td>13</td>
</tr>
<tr>
<td>(0.5)</td>
<td>12</td>
</tr>
</tbody>
</table>

Table (6.3b) EML formulation with \( \eta = 0.1 \) and \( \alpha = 0.75 \)

Table (6.3) Number of misclassified samples after 100 iterations for the overlapping triangular distribution.
Table (6.4a) Original BP with $\eta = 0.4$ and $\alpha = 0.75$

<table>
<thead>
<tr>
<th>$w_1(0)$</th>
<th>$w_0(0)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.5</td>
<td>-0.5</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>-0.3</td>
<td>13</td>
</tr>
<tr>
<td>-0.1</td>
<td>20</td>
</tr>
<tr>
<td>0.1</td>
<td>20</td>
</tr>
<tr>
<td>0.3</td>
<td>34</td>
</tr>
<tr>
<td>0.5</td>
<td>22</td>
</tr>
</tbody>
</table>

Table (6.4b) EML formulation with $\eta = 0.1$ and $\alpha = 0.75$

<table>
<thead>
<tr>
<th>$w_1(0)$</th>
<th>$w_0(0)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.5</td>
<td>-0.5</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>-0.3</td>
<td>12</td>
</tr>
<tr>
<td>-0.1</td>
<td>12</td>
</tr>
<tr>
<td>0.1</td>
<td>12</td>
</tr>
<tr>
<td>0.3</td>
<td>12</td>
</tr>
<tr>
<td>0.5</td>
<td>12</td>
</tr>
</tbody>
</table>

Table (6.4) Number of misclassified samples after 500 iterations for the overlapping triangular distribution.

140
At 500 iterations (Table (6.4)), the EML has converged to a minimum for all the 36 trials irrespective of the starting point in comparison with 2 in 36 trials for BP. In all the trials EML has produced optimum misclassification of 3.75%, while BP has reduced the misclassification to 9.38% or more in 7 of 36 trials.

These examples were chosen to minimise the complexity of the experiment and to enable a clear representation of the results. With more complicated networks, there will inevitably be regions of the weight space for which neither method achieves convergence to the global minima. However, experimental results presented in sections ahead will show that in such cases EML significantly increases the speed of convergence and the probability of reaching a solution compared with BP.

6.3.4 Generation of Complex Decision Boundary

In this section the capability of the two learning procedures for generating a complex decision boundary is demonstrated. The data for this test is extracted from Chang[17] and its distribution is shown in Fig(6.3). As can be seen from the Fig(6.3), this experimental data is a non-linearly separable case consisting of 15 samples belonging to Class 1 and 15 samples to Class 2.

The network architecture used in this experiment was a two layer network consisting of two input units, four hidden units and an output unit. The learning parameters were set to \( \eta = 0.04 \) for BP and \( \eta = 0.01 \) for EML, and \( \alpha = 0.8 \) for both cases.

Graph (6.8) shows learning curves averaged over 10 different starting points for both the learning methods. The training of the networks were terminated at 500 iterations, where both the learning rules produced a separating decision boundary (zero errors) before termination in all the 10 cases. On average, this was achieved after 360 iterations for EML and 452 for BP.

This experiment illustrates that speed with which EML produces a complex separating decision boundary outperforms the BP.
Fig. (6.3) Chang’s data distribution

Graph (6.8) Learning curves for Chang’s data.
6.3.5 Generalisation on Synthetic Data

One of the most important requirements of neural networks is the ability to generalise correctly to new cases after training on a sufficiently large set of typical cases from the same data distribution [18]. As discussed in the previous chapter, there is no formal definition of what is meant by 'generalise correctly.' However, generalisation of a network can be experimentally assessed by training the network on examples from a domain that the experimenters understand, and then judge to generalise correctly if its generalisation agrees with those of the experimenters. This is sufficient as an informal definition that the network can generalise, but however it does not give any predictions about when networks will generalise correctly and when they will fail [19,20].

In this part of the thesis, experiments on synthetic and real data arising from binary template matching problems were conducted to assess the speed of convergence and generalisation capability of BP and EML. The relative generalisation capability on different training set sizes is also tested.

6.3.6 Experiments on Synthetic Data

Three sets of two-class data are considered in these simulations. These data are generated using SUN 3/60 and the NAG library random number generators, and are then transformed appropriately.

The three types of distributions are (a) marginally separable uniform (b) overlapping triangular and (c) overlapping normal. For each distribution the feature space is one dimensional and the samples are drawn from the distributions listed above. For each experiment six training sets each containing 10, 20, 40, 80, 160 and 320 training patterns were produced, so the effects of training set size on the relative performance of the two learning rules can be assessed.

Data sets consist of equal number of training patterns from the two classes, giving the classes equal a priori probabilities. The network for these experiments consists of single node and was trained to find a decision boundary where the error rate is minimum.
As discussed in section 5.8 to assess the generalisation capability of a trained network, one could directly measure it by using a large number of test samples (samples that are not included in the training). However in experiments carried out in this section, it is simple to calculate generalisation from the known distribution, an exact figure which would be obtained in the limit as the test set size approaches infinity. An example of such case is shown in Fig(6.4). Assuming the single node network converges with weights \( w_0 \) and \( w_1 \), the decision threshold occurs at \( -\frac{w_0}{w_1} \). The generalisation error can then be obtained by calculating the areas on the wrong side of the decision boundary (the shaded area in Fig(6.4)).

(a) Marginally separable uniform

In this experiment referred to as marginally separable uniform distribution task, there is a one dimensional feature space and two classes (Fig(6.5a)). Class 1 extended from -1 to 1 and Class 2 from 1 to 5.

A single node network with \( \eta = 0.4 \) for BP, and \( \eta = 0.1 \) for EML with \( \alpha = 0.8 \) for both cases was trained using all the six data set sizes. The training of a network was terminated at 500 iterations.

Learning and generalisation curves for this experiment are shown in graphs (6.9a) to (6.9f). As can be seen from graph(6.9a) (10 data points), both the learning rules have successfully converged to zero error decision boundary within the same number of iterations on the training set. However both the learning rules fail to generalise correctly, that is the decision boundary produced by the networks at convergence does not correspond to the theoretical minimum error rate boundary; instead both the networks have generalised to 5.1% error rate. This due to the fact that 10 data points do not make a good representation of the underlying probability distribution.

The difference between speed of convergence starts to become apparent for 2D pattern data set, graphs (6.9b). The zero error convergence on training data for the EML learning rule occurs at 16\textsuperscript{th} iteration as compared with 33 for the conventional BP. However again both BP and EML rules have failed to produce error free generalisations; the generalisation error is reduced to 2.5% as compared with 5.1% for the 10 data point case.
Fig. 6.4 Example of directly measureable generalisation error rate.

Fig. 6.5 Synthetic data distributions used in the experiments.
Graph (6.9a) Learning and generalisation curves for 10 data points Separable Uniform distribution.

Graph (6.9b) Learning and generalisation curves for 20 data points Separable Uniform distribution.
Graph (6.9c) Learning and generalisation curves for 40 data points Separable Uniform distribution.

Graph (6.9d) Learning and generalisation curves for 80 data points Separable Uniform distribution.
Graph (6.9e) Learning and generalisation curves for 160 data points Separable Uniform distribution.

Graph (6.9f) Learning and generalisation curves for 320 data points Separable Uniform distribution.
Graph (6.10) Effect of training set size on learning and generalisation using Separable Uniform distribution.

<table>
<thead>
<tr>
<th>training set size</th>
<th>BP training</th>
<th>BP Generalisation</th>
<th>EML training</th>
<th>EML Generalisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>16</td>
<td>---</td>
<td>16</td>
<td>---</td>
</tr>
<tr>
<td>20</td>
<td>33</td>
<td>---</td>
<td>16</td>
<td>---</td>
</tr>
<tr>
<td>40</td>
<td>95</td>
<td>---</td>
<td>16</td>
<td>---</td>
</tr>
<tr>
<td>80</td>
<td>98</td>
<td>117</td>
<td>33</td>
<td>35</td>
</tr>
<tr>
<td>160</td>
<td>149</td>
<td>149</td>
<td>45</td>
<td>45</td>
</tr>
<tr>
<td>320</td>
<td>240</td>
<td>240</td>
<td>67</td>
<td>67</td>
</tr>
</tbody>
</table>

Table (6.5) Number of iterations required to obtain optimum error for Separable Uniform distribution.
The faster convergence of EML becomes even more prominent at the 40 point data set shown in graph (6.9c), but still both methods fail to produce zero error generalisation.

For the next three data set sizes, 80, 160 and 320 both the learning methods produce near perfect generalisation. The generalisation error rates for the respective data set are 1.79%, 1.13% and 0.62% for BP and 1%, 0.8% and 0.5% for EML. As shown in graphs (6.9d), (6.9e) and (6.9f) in all the cases the EML has converged faster on training data and has produced a better generalisation as compared with BP.

The Table (6.5) shows the convergence to zero error rate for both the learning rules. The generalisation for 10, 20 and 40 training set sizes is not shown in the table since in none of the cases the networks produced near optimum generalisation. The generalisation error rate for these entries can be extracted from the appropriate graphs. EML always converges faster both on training and generalisation in comparison to BP. The table also shows that there is a progressive reduction in the speed of convergence as the number of training patterns increases. This is because increasing the training set size produces a more accurate representation of the underlying data distribution and, since the data is marginally separable, decreases this margin. Hence this becomes a harder learning task for the network and therefore the number of iterations increases.

Graph (6.10) shows the percentage of generalisation error rate against training set size, at 500 iteration (convergence). It can be seen from the graph that increasing the training set size increases the generalisation capability of both the learning rules. The generalisation obtained for EML is consistently better than for BP, but in this experiment the distinction is marginal.

(b) Overlapping Triangular Distribution

In this experiment the speed of convergence and generalisation of BP and EML learning rule for overlapping triangular distribution is tested. As shown in Fig.(6.5b), the two classes extend from -1 to 2 for Class 1 and 0 to 5 for Class 2.

A single layer network with $\eta = 0.5$ for BP, $\eta = 0.13$ for EML and $\alpha = 0.85$ for both cases was trained using six different training set sizes. The training of the networks was terminated at 500 iterations.
Graph (6.11a) Learning and generalisation curves for 10 data points Overlapping Triangular distribution.

Graph (6.11b) Learning and generalisation curves for 20 data points Overlapping Triangular distribution.
Graph (6.11c) Learning and generalisation curves for 40 data points Overlapping Triangular distribution.

Graph (6.11d) Learning and generalisation curves for 80 data points Overlapping Triangular distribution.
Chapter 6 - Empirical Maximum Likelihood Learning

Graph (6.11e) Learning and generalisation curves for 160 data points Overlapping Triangular distribution.

Graph (6.11f) Learning and generalisation curves for 320 data points Overlapping Triangular distribution.
Graph (6.12) Effect of training set size on learning and generalisation using Overlapping Triangular distribution.

Table (6.6) Percentage of misclassified samples after 500 iterations through the Overlapping Triangular distribution.
Learning and generalisation curves for these experiments were plotted and are shown in graph (6.11a) to (6.11f). Table (6.6) shows the summary of training and generalisation error rates at convergence (at 500 iterations). Although this table does not show speed with which these error rates have been achieved, these can be observed from the learning and generalisation curves.

From the graph (6.11a) and graph (6.11b) it can be seen that both the learning curves (BP and EML) have converged to zero error rate. This is because the 10 and 20 data points selected from the triangular distribution do not form an overlapped area, however both the networks have not generalised correctly for this case where the training data does not exactly represent the underlying probability distribution.

The next two graphs, graph (6.11c) and graph (6.11d) show the curves for 40 and 80 data points respectively. For the 80 data point case BP fails to produce optimum learning and generalisation. BP network has converged to 10% training and 9.8% generalisation error as compared with 6.25% training and 4.57% generalisation error for EML.

The lower generalisation error rate for EML can also be observed for the 160 and 320 data set sizes (graphs (6.11e) and (6.11f)), where EML has produced error rates of 5.17% and 4.50% respectively. The corresponding error rates for BP are 10.37% and 9.145%, and the minimum theoretical generalisation error rate is 4.44%.

For all the cases of overlapping triangular distribution EML has produced a more accurate generalisation than its counterpart at a much faster speed of convergence.

Table (6.6) and graph (6.12) show the percentage of training and generalisation at 500 iterations. Table (6.6) shows that for the training set sizes greater than 80 data points BP has failed to converge to a good solution and hence resulted in a poor generalisation of the network. Table (6.6) also illustrates that as the training set size has increased, the performance of BP degrades; this is due to the reduced solution region. However increasing the training set size has shown no effect on the convergence speed of EML, and as shown EML consistently converges to an efficient solution both in training and generalisation for all the training set sizes.
(c) Normal Distribution

As in the two previous cases, this data set is also a randomly generated two class case. As shown in Fig.(6.5c), both Class1 and Class2 are normally distributed, with Class1 mean $m_1 = 0$, standard deviation $SD_1 = 1$ and Class2 mean $m_2 = 2.5$, standard deviation $SD_2 = 0.6$.

A single layer network with $\eta = 0.2$ for BP, $\eta = 0.05$ for EML and $\alpha = 0.75$ for both cases, was trained using the six data set sizes. The training was terminated at 500 iterations.

Graphs (6.13a) to graph (6.13g) illustrate the learning and generalisation curves for this experiment. For the first four cases, 10, 20, 40 and 80 data set sizes, both the training methods produce approximately identical speed of convergence and similar error rates both on the training and generalisation. Also in the last two cases, 160 and 320 data points both the methods produce similar error rates, however a clear advantage in the speed of convergence for EML can be observed.

The summary of training and generalisation error rates at convergence (500 iterations) is shown in table (6.7) and graph (6.14). Only a marginal advantage in EML generalisation can be observed. Compared with the 5.67% theoretical generalisation error rate, the best BP and EML generalisation error rates occur for the 320 data points set, giving 6.08% and 5.72% misclassification rates respectively.

6.3.7 Performance on Experimental Data

The experimental data were generated from binary template matching of text character images at low resolution under noisy conditions[21,22]. The data used for these tests were obtained by measuring two features of characters extracted from a single document (appendix A). Six data set sizes; 50, 100, 200, 300, 400 and 500 were used for training and a separate 1700 pairs were used for testing the trained networks.
Graph (6.13a) Learning and generalisation curves for 10 data points Normal distribution.

Graph (6.13b) Learning and generalisation curves for 20 data points Normal distribution.
Graph (6.13c) Learning and generalisation curves for 40 data points Normal distribution.

Graph (6.13d) Learning and generalisation curves for 80 data points Normal distribution.
Chapter 6: Empirical Maximum Likelihood Learning

Graph (6.13c) Learning and generalisation curves for 160 data points Normal distribution.

Graph (6.13f) Learning and generalisation curves for 320 data points Normal distribution.
Graph (6.14) Effect of training set size on learning and generalisation using Normal distribution.

<table>
<thead>
<tr>
<th>training set size</th>
<th>% training error</th>
<th>% Generalisation error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BP</td>
<td>EML</td>
</tr>
<tr>
<td>10</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>20</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>40</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>80</td>
<td>3.75</td>
<td>3.75</td>
</tr>
<tr>
<td>160</td>
<td>4.06</td>
<td>3.52</td>
</tr>
<tr>
<td>320</td>
<td>4.38</td>
<td>4.63</td>
</tr>
</tbody>
</table>

Table (6.7) Percentage of misclassified samples after 500 iterations through the Normal distribution.
A two layer network with two hidden nodes was used for training. The learning parameters used were $\eta = 0.048$ for BP and $\eta = 0.012$ for EML, with $\alpha = 0.7$ for both methods.

Graphs (6.15a) to graph(6.15f) show the learning and generalisation curves for both BP and EML, averaged over 10 random starting points, so to remove the dependency on the start-up weights.

These graphs illustrate that in all the cases EML converges faster to a more accurate solution than BP. Comparing the graphs (6.15a) (50 data points) and graph(6.15f) (500 data points), it can be observed increasing the sample set size has improved the generalisation error rates for both the training methods.

A Summary of training and generalisation error rates at convergence (100 iterations) is illustrated in table (6.8) and graph (6.16). It can be seen in all the cases the EML method has yielded a more efficient generalisation than BP. The generalisation error rates produced by the two learning methods for 50 and 500 data points are 7.30% and 1.05% for BP and 5.03% and 0.77% for EML respectively.

### 6.4 Conclusion

An alternative formula for learning in multilayer networks based on EML has been presented. Simulation analysis of BP and EML show that EML has number of advantages over BP and hence significantly improves the performance of a neural network trained with this method. As shown in the formulation of EML (section 6.2), it involves fewer calculations per iteration. In addition to speeding the calculations in conventional computers, this could also reduce the complexity of hardware implementations of neural networks.

In section 6.3.2, using a simple simulation it was demonstrated that EML produces a simpler error surface than BP. This is one of the important contributing factors in enabling a faster speed of convergence than BP.

This issue was investigated further in section 6.3.3 where it was shown that success (convergence to a minimum) of BP was heavily dependent on the initial values of start up weights. These simulations illustrated that performance of EML is not
Graph (6.15a) Learning and generalisation curves for 50 Template Matching data points.

Graph (6.15b) Learning and generalisation curves for 100 Template Matching data points.
Graph (6.15c) Learning and generalisation curves for 200 Template Matching data points.

Graph (6.15d) Learning and generalisation curves for 300 Template Matching data points.
Graph (6.15e) Learning and generalisation curves for 400 Template Matching data points.

Graph (6.15f) Learning and generalisation curves for 500 Template Matching data points.
critically dependent on initial weight assignments.

Further simulations on both experimental and synthetic data confirmed that EML not only speeds up the learning but also it improves the generalisation capability of a network. Moreover analysing the simulation results (sections 6.36 - 6.37) shows that a good generalising solution appears to be achievable in fewer iterations or with fewer training data[14].
References

14. S.Semnani and M.J.J.Holt, 'Improving the Performance of Multilayer...
Chapter 6: Empirical Maximum Likelihood Learning


Chapter 7

Conclusions

7.1 Conclusions

This chapter provides an overview of the research work and identifies the important results and conclusions. This is followed by a discussion and suggestions for further research.

In designing trainable pattern classifiers one is faced with the task of selecting the 'best' classifier for a specific problem. Performance of a classifier is usually assessed by the speed with which the classifier reaches a solution, computational complexity and the error rates on both training and test data. In this thesis two main approaches are considered: classical discriminant functions and neural network pattern classifiers.

The investigation on classifier design started with a short overview of classification methodologies that included syntactic and statistical methods. In the statistical methods, the suitability of supervised versus non-supervised, and of parametric versus non-parametric for trainable classifiers, were discussed.
The following discussion led to simulations and comparison of some well known linear discriminant functions and to the development of an alternative and a more efficient linear discriminant function.

Simulation studies of Chapter 2 showed some of the capabilities and drawbacks of existing LDFs. The Perceptron was found to be an attractive method because of its relative simplicity, but it suffers from slow convergence rate and also its performance degrades for non-separable cases. Other methods such as Widrow-Hoff and Ho-Kashyap were formulated so as to perform well for both separable and non-separable cases, but both classifiers suffer from slow convergence.

A new method of linear classification based on GLM was proposed in Chapter 3. The experimental results demonstrated that the GLM method always converges to a separable solution for separable data sets and consistently converges to a solution giving the same or fewer misclassifications than those produced by Ho-Kashyap and Widrow-Hoff. Moreover the overall convergence rate of GLM was found to be significantly faster than the other simulated methods.

Although GLM offers an improvement in performance, its linearity severely limits the range of classification problems where it can be applied. In chapter 4 some extensions of linear methods overcoming the problem of linearity were considered. These methods included quadratic (polynomial) discriminant functions and multilayer discriminant functions. Simulations of quadratic GLM demonstrate how an LDF can be transformed to produce non-linear decision boundaries. Furthermore higher order discriminant functions can also be produced using the same principles. However a major drawback of PDFs is the computational cost. More terms need to be evaluated in order to produce a more complex decision boundary.

One method of producing PDFs was shown to be through the group method of data handling (GMDH). Although this method requires relatively fewer terms to be evaluated it introduces extra storage, requires relatively larger amounts of training data and is only suitable for very complex problems.

Alternatively one can generate Piecewise linear discriminant functions through multilayer LDFs. This method was proposed by various researches during 60’s, but the fundamental difficulty in this type of system was the efficient adaption (training) of weights in all the layers of a multilayer classifier. It was the recent exploration in the field of neural networks that has led to various methods of training such multilayer classifier.
In Chapter 5 the capability and design issues of a popular neural network, namely the multilayer network trained by backpropagation (BP), was introduced. This was followed by reviews of network design (number of hidden layer and number of nodes on each layer), generalisation, effects of learning constant and momentum term on learning speed, and properties of BP error surface.

Simulations of BP have shown that learning problems that were proved difficult or impossible to solve (eg Xor problem) by the classical methods can easily be solved by a two layer network. Other simulations have illustrated that BP networks are capable of producing a separating solution for separable data and an optimal solution of the non-separable cases.

Major drawbacks of BP learning are the slow speed of convergence and the tendency of getting trapped in local minima. This has led to various strategies to improve these drawbacks.

Jacob's adaptive learning rate[1], learning by diffusion[2], high order correlation[3] and conjugate gradients[4] are some of the variations that are proposed by researchers attempting to improve the learning rate of BP. Although these variations improve the performances of BP, this is usually achieved by inclusion of more complex learning strategies which effectively increase the computational time and in some cases introduce more learning parameters to be optimised[2,4].

An alternative method for multilayer networks was introduced in Chapter 6. This method was formulated by assuming the network outputs as probabilities. The new method EML has proved to have several advantages over BP.

Networks trained using EML had a faster speed of convergence over BP. This was as a result of fewer calculations per iteration and relatively simpler error surfaces. It was also demonstrated that convergence of EML to a minimum was not critically dependent on initial weight assignment. Simulations on both synthetic and experimental data indicate that EML can also provide faster and better generalisation than BP.
7.2 Recommendation for Further Research

The recent surge in the field of neural networks has provided an attractive alternative solution to many classical problems. Various training methods in neural networks have been explored and applied to areas of research and engineering as diverse as market forecasting, horse racing, control, speech and image recognition etc.

In this thesis some of the advantages of EML over the conventional BP were demonstrated. Further exploration of these capabilities can be assessed by the application of EML to a more difficult task like speech recognition[5]. Also one can investigate the effect of varying network architecture (that is varying the number of layers and number of nodes) for a particular problem.

In section (6.3.2) a simple example was used to illustrate the error surfaces generated by BP and EML, and it was shown that EML had a relatively simpler error surface. However it would be interesting to investigate the shape of the error surface for larger networks and see whether EML can reduce the likelihood of getting trapped in a local minima.

In this thesis application of EML was only confined to BP, however one can also apply EML to different neural networks paradigms such as Time Delayed[6], Counterpropagation[7] etc. Whether the advantages of EML extend to such paradigms is a matter of further research.

There are now many different neural network learning procedures that can appropriately be applied to pattern classification problems. To achieve further success in neural networks, one needs more effective learning algorithms, providing faster speed of convergence, good generalisation and an 'optimal' network structure[8].

For a given task three major problems facing a network designer are: selection of network structure (number of layers and number of nodes in each layer), number of weights and the network learning constants. These factors are very much task dependent and they determine success or failure of a network. In most of the research works so far, these parameters are selected experimentally, and for a complex task or a task with large amount of training data this can prove to be very time consuming approach.
This argument leads into learning algorithms which can change the network parameters and the network topology dynamically as the learning progress. In this type of learning the network designer is relieved of the burden of guessing in advance precisely what network topology and parameters will best fit a given problem. Changing the network topology will have a direct effect on the speed of convergence and more importantly on generalisation.

Dynamic architectures are exciting because they choose their own topology and because they offer significant advantages in designing of neural networks. Clearly this is an area with a big space of possible designs, only partially explored and it deserves more research.
References

Appendix A

A.1 Extraction of Template Matching Data

One of the applications of template matching arises in the facsimile coding using symbol-matching techniques[1]. Symbol matching is a powerful approach for compression and transmission of digitised documents. The block diagram for symbol matching is shown in Fig (A.1).

Essentially the symbol matching starts by scanning a digitised document to determine if a black pixel exists. If the entire scan line contains no black pixel, this information is coded by end of line code. If a black pixel is encountered a blocking process is conducted to segment the symbol. Each symbol is compared with previously segmented symbols in the library of prototype symbols. This process involves the extraction of a set of easily measurable features, a screening operation to reject unsuitable candidates, and finally template matching. Initially the symbol library is empty and the first segmented symbol and its feature vector we always put into the prototype library, and each subsequent symbol is compared with each entry of the library that passes the screening test.

If a matching library symbol is found only the identification of code of the library symbol and the position of the current symbol are transmitted. If no matching
Begin

Uncompressed Image

Symbol Extraction

Screening

Matching

Symbol Library

Symbol Coding

More Symbol

Yes

No

Residual Image (all symbols removed)

Residue Coding

Compressed Image

End

Fig (A.1) Block diagram of symbol matching process
Fig. (A.2) Features used in the screening process: Height = 15, Width = 15, Perimeter = 112 and Black pixel count = 101.

Fig. (A.3) Calculation of the Weighted And Not count.

WAN Count = 77
library symbol is found the current symbol is added to the library. Once all symbols in the digital document have been coded the remaining "..." objects in the document, known as the 'non-symbols' or residue[2] are coded, usually by the modified read code[3]. Hence the process of facsimile coding using symbol matching can be divided into number of independent activities: character segmentation, symbol matching, library management and coding which are described in detail in appropriate references[4].

However in order to describe the nature of template matching data it is necessary to explain symbol matching activity.

A.2 Symbol Matching

The process of symbol matching is divided into two stages of screening and template matching. Because the bit-by-bit comparison involved in template matching is time consuming, a preliminary screening process is carried out, so only good candidates are passed to the template matching stage. This is accomplished by measuring several features such as height, width, number of black pixels and perimeter of a candidate character (shown in Fig (A.2)) and calculating its feature distance from each library entry to determine which library symbols should be considered for template matching.

Template matching is the final process in the symbol matching. It involves the bit-by-bit comparison of the isolated symbol with the library candidate symbols. This is generally performed by superimposing the two patterns and finding the difference between the two patterns by doing a logical And-Not [5] or Exclusive Or[1] operation. Each resulting error pixel from this logical operation is assigned a weight equal to the number of error pixels in a 3 by 3 neighbourhood of the current pixel. The result of this operation is known as weighted And-Not map and is shown in Fig(A.3).

The matching decision is made on the basis of this error map. This is achieved by summing the elements of WAN map (the WAN count) and comparing it with a threshold. If the WAN count is below a threshold, it indicates a match otherwise the match is rejected. Although the final decision is a 'match' or 'no match' type of decision (two class case) it requires a non-linear function to obtain the threshold.
In Classification terms, the WAN count is a feature measurement used to detect a match between two characters in the same typeface. This feature measurement alone is insufficient, and a good distinction is only achievable when other features, such as black pixel count, width, height of the characters are also taken into account. These features are used to train a classifier such as the one described in Chapters 2, 3 and 6 to produce a decision boundary (threshold). The decision boundary (threshold) generated by the classifiers is used to categorise the measured features into 'match' or 'no match' classes.
References
