The Application of Neural Networks to Communication Channel Equalisation : A Comparison between Localised and Non-localised Basis Functions

Half-dissertation

Prepared by :
Avron B.Olshewsky, *B.Sc.(Eng.), University of Cape Town.*

Prepared for :
The Department of Electrical Engineering at the University of Cape Town.

February 1997

Thesis prepared in partial fulfilment of the requirements
for the degree of M.Sc. in Electrical Engineering.

Declaration

I hereby declare that this dissertation is my own work. It is being submitted for the degree of Master of Science in Engine ing at the University of Cape Town. This work has not been submitted before for any gree or examination in any other University.

Signed by candidate

Candidate

Acknowledgements

I would like to thank my supervisor Assoc. Prof. John Greene for his assistance and guidance during the course of this research. I would like to thank Dr R.M.Braun for the use of a computer and office space.

I would like to thank my parents for their continual support and love.

This work is dedicated to my late grandfather Simon Fortes who passed away during the writing of this work. He will not be forgotten.

Terms of Reference

This half-dissertation document describes the research done by the author under the supervision of Assoc. Prof. J.R.Greene. The area of research is the application of Neural Networks to adaptive channel equalisation.

The goals were as follows :

- To understand the equalisation problem.
- To find Neural Network architectures and algorithms that have/could be applied to this problem.
- To understand the theory and operation of these structures.
- To simulate these structures and analyse their operation and performance.
- To complete this research in a period of 6 months.
- To hand this dissertation in by the 24th of February 1997.

Synopsis

Equalisers are needed in digital communication systems in order to compensate for the intersymbol interference that arises due to transmission bandwidth restrictions. Modern communication systems are operating in increasingly adverse conditions. The available frequency spectrum is becoming over-subscribed, many communication services are being provided via satellite communication links, whose receivers operate near saturation. Many of the modern communication channels are time-varying. It is for this reason that modern adaptive equalisers require increasingly more processing power.

Neural networks have been applied to a number of problems over the past few years. One of the emerging applications of neural networks is adaptive communication channel equalisation. This area of research has become prominent due to the reformulation of the equalisation problem as a classification problem. Viewing equalisation as a classification problem allows researchers to apply the knowledge gained from other fields to equalisation. A wide variety of neural network structures have been suggested to equalise communication channels. Each structure may in turn have a number of different possible algorithms to train the equaliser.

A neural network is essentially a non-linear classifier; in general a neural network is able to classify data by employing a non-linear function. The primary subject of this dissertation is the comparative performance of neural networks employing non-localised basis (non-linear) functions (Multi-layer Perceptron) versus those employing localised basis functions (Radial Basis Function Network).

When considering the performance of an adaptive equaliser there are a number of related issues that must be considered. One important issue is the convergence of the training algorithms to the optimum solution. It is desirable to employ equalisers that perform consistently well and are able to adapt in as short a time period as possible. Another issue is the computational complexity required to implement the training algorithms. At high data rates it may not be possible to construct a equaliser that requires a large number of computations to adapt itself, due to the lack of fast enough processors. The processing speed required by the computational elements in the equaliser is directly related to the cost of the equaliser.

In order to classify a set of data as belonging to one or another set, a classifier will form a decision boundary. Linear classifiers are only able to form linear decision boundaries in the input space. The adaptive linear transversal filter is one such linear classifier, this equaliser structure forms a hyperplane in the input space to classify the input (received) data. The disadvantage with this approach is that in most practical applications this linear decision boundary is sub-optimum. This is particularly true when considering data points that are not linearly separable (as is the case in non-minimum phase channels) or non-linear channels found in satellite communication systems; naturally in these instances a linear classifier performs very poorly. The advantage with this approach is that the equaliser structure is relatively easy to analyse in terms of convergence and is computationally efficient.

The question arises as to what the optimum strategy for the symbol by symbol classification of data is ? The answer to this question lies in the statistics of the data transmission. Consider a binary transmission alphabet (the transmitted data is either a $+1$ or -1 signal), during transmission the signal is corrupted by the reminisce of previous transmitted symbols (ISI) and at the receiver is further corrupted by additive noise. The receiver will receive some signal represented by the vector y the received signal must then be classified as having been caused by a $+1$ or -1 transmitted symbol. A relevant question to pose is: given that the vector y was observed at the receiver, what is the probability that this vector was caused by a $+1$ transmitted symbol ?. If it is more probable that the vector y was caused due to a $+1$ being transmitted than a -1, it must be concluded that the vector y was caused by a $+1$ transmitted symbol. Naturally the converse is also true. The decision boundary created by this strategy is the locus of received signal points for which the probability of y having been caused by a $+1$ is equal to the probability of y having been caused by a -1. This strategy is known as Bayesian Estimation. It can be shown that this is the optimum strategy for the symbol-by-symbol classification of data. Generally the decision boundaries formed by this approach are non-linear.

In a noise-less environment the received signal y will be one of N discrete signal vectors. In a nosy environment the received signal y will be distributed in some way about these discrete 'states'. The Bayesian decision boundary is given by the intersection of the conditional density functions associated with a $+1$ transmitted symbol and the conditional density functions associated with a -1 transmitted symbol. If it is assumed that the noise component is additive white Gaussian noise, the conditional density functions are Gaussian kernels. It can be shown that a Radial Basis Function Network employing Gaussian kernels as localised basis functions is able to directly model the Bayesian Estimation formula, provided the correct number of basis functions are supplied and provided that they are correctly trained. Thus the Radial Basis Function Network equaliser has a direct relationship to Bayesian Estimation; the optimal classification strategy.

The multi-layer perceptron employs non-localised basis functions. This structure is also able to form non-linear decision boundaries. A little known novel technique, which has been shown to be more robust than the Back-propagation algorithm, is used to train the multi-layer perceptron (MLP).

Cover's theorem on the separability of patterns states that a pattern set cast into a high dimensional space via a non-linear mapping is more likely to be linearly separable than in a low dimensional space. The MLP performs this high dimensional non-linear mapping, the output weights effectively pass a hyperplane through his high dimensional space (in a least-squares sense). It is the inverse mapping of this hyperplane to the input space that forms the non-linear decision boundary.

Simulations show that the Radial Basis Function Network Equaliser (RBFNE) outperforms the MLP in all cases, both these structures outperform the Linear transversal Equaliser. The RBFNE has the advantage that the algorithm used to train the equaliser is guaranteed convergence and this algorithm is computationally efficient. The performance of the MLP is dependent on the number of basis functions employed. With too many basis functions the MLP is unable to learn, this is particularly true in low noise environments where the training data only occupies certain concentrated regions of the input space.

The RBFNE has a direct relationship to Bayesian Estimation, the MLP has no such relationship. At best the MLP can only approximate the performance of the RBFNE; assuming the RBFNE has been correctly trained. However, the MLP has more generalised classification powers in that no assumptions regarding the noise distribution are made. It is this lack of pre-determined structure (the non-localised nature of the basis functions) that results in an inferior performance to the RBFNE.

The algorithms needed to train the RBFNE are computationally efficient and robust. This together with the performance gain offered by this structure, over the LTE and MLP, make it a viable alternative. The algorithms needed to train the MLP require additional computation. A one-shot training algorithm is used which allows for the update of parameters at the end of the training sequence, thus parameter updates do not need to be performed in real-time. In general this technique requires relatively few hidden neurons.

Table of Contents

List of Figures and Tables

Figures :

Tables :

List of Symbols

$E[]$	The expectation operator.
ϕ	A non-localised non-linear function.
i	Index value.
j	The number of centres or channel states.
φ	A localised non-linear function.
k	Index value or discrete variable of samples.
m	Filter order or number of network inputs.
M	The number of symbols in an alphabet.
N	The number of neurons in a network.
η	Learning rate for LMS.
$\Re$	Refers to Dimension.
σ^2	Denotes variance.
t	A variable of time.
τ	A time delay.
μ	Learning parameters.
x	Variable used to represent an input.
y	Variable used to represent an output.

AWGN	Additive white Gaussian noise.
LTE	Linear Transversal Equaliser.
LTF	Linear Transversal Filter.
MLP	Multi-layer Perceptron.
RBFN	Radial Basis Function Network.
RBFNE	Radial Basis Function Network Equaliser.
SNR	Signal-to-noise Ratio.

Chapter One

Introduction

1.1 Background

Equalisation has been the subject of much research over the past few decades. As the requirements of communication systems become more stringent there is a need to construct equalisers of increasing complexity and sophistication. Bandwidth restrictions have intensified due to the overcrowding of available bandwidth. Satellite communication receivers operate near saturation, introducing non-linearities. These factors amongst others have necessitated the need to develop equalisers that are able to compensate for the adverse conditions they operate under.

Neural networks have been the focus of extensive research over the past two decades. Neural networks have been applied to a wide variety of problems, which include function approximation; pattern classification and prediction. Problems that have been traditionally difficult or impossible to solve with linear techniques often may be solved using neural networks. Thus neural networks have evolved from a "gimmick" into a widely accepted engineering tool.

Neural networks have been applied successfully to communication channel equalisation. A variety of network architectures have been proposed many of which offer an improvement of performance over standard linear techniques. This performance gain is usually at the expense of computational complexity. Many of the proposed architectures are impractical due to their computational complexity.

1.2 Objectives

The primary objective of this dissertation is to focus on two classes of neural networks for the adaptive equalisation of communication channels. The first class of neural network is one employing localised functions (known as Radial Basis Function Networks), while the second class of network employs non-localised functions (known as the Multi-layer Perceptron). Based on work conducted by Mulgrew *et al* [1] it is hypothesised that under certain conditions structures employing localised basis functions offer an advantage over networks employing non-localised functions, in terms of performance and complexity.

A secondary objective is to introduce in the context of equalisation, a novel new technique for training Multi-layer Perceptrons which has recently been applied to pattern classification and function approximation problems. It will be shown that both the Radial Basis Function Network Equaliser and the Multi-layer Perceptron equaliser are practically viable solutions to the equalisation problem, while the former technique offers a number of significant advantages in this application.

1.3 Procedure

This dissertation arose out of an interest in adaptive systems and in the emerging field of neural networks. The research began by considering the application of infinite impulse response filters in conjunction with neural networks for the adaptive equalisation of communication channels. This particular area of research soon reached a 'dead-end'. During the course of this research it became apparent that there where interesting avenues being explored by researchers with neural networks employing both localised and non-localised basis functions. After analysing the theory of channel equalisation, the next step was to focus on the two types of neural networks and gain an understanding of how they work and where their processing powers lie, particularly with application to channel equalisation. The final step involved simulating these networks and analysing the results.

1.4 Scope and Limitations

The major limitation was time. The research was conducted over a period of six months, as a result the research was highly focused. Due to the computational time required to simulate various aspects of the research it was necessary to limit the number of simulations. This in turn affects the number of conclusions that can be drawn.

1.5 Plan of Development

The dissertation is organised in the following manner: Chapter two briefly introduces the equalisation problem and the reformulating thereof as a classification problem. The adaptive linear transversal equaliser is then introduced as a linear classifier after which the optimal classification strategy is presented. Chapter three deals with the neural network structures considered and their relevance to the equalisation problem. Chapter four deals with the training of these neural network adaptive equaliser structures. This includes an analysis of issues such as convergence and computational complexity. The results and analysis of the simulations conducted are presented in chapter five. Chapter six presents a summary and conclusions of the dissertation, areas of further research are also proposed.

Chapter 2

The Equalisation Problem

2.1 Introduction

In this chapter the equalisation problem will be presented. The chapter begins with a brief description of Intersymbol Interference (ISI) which necessitates the use of equalisers in digital communication systems. Equalisation is presented as an inverse-modelling problem. The Adaptive Linear Transversal Filter (LTF) is then briefly presented. A simple example is then used to shown how equalisation can be reformulated as a pattern classification problem. The chapter concludes with a an introduction to the theory of Bayesian Estimation with application to equalisation.

2.2 Intersymbol Interference

Digital communication systems are subjected to ISI. The interference is usually a result of the restricted bandwidth allocated to the channel and/or the presence of multipath distortion in the medium through which the data is transmitted [2].

In other words the received signal consists of the current symbol and interference caused by the reminisce of previous symbols and/or delays in the various paths that the signal may follow.

2.2.1 Modelling a Communication Channel with ISI

The effects of ISI can generally be modelled by a Finite Impulse Response (FIR) filter. In general the interference of one bit on future bits will only be significant for a finite time duration. If the order of the FIR filter is made sufficiently large the communication channel can be accurately modelled by such a filter.

Communication channels are also subjected to noise. This noise is generally modelled as Additive White Gaussian Noise (AWGN). This type of noise process is usually a very good approximation of the real situation and is well understood mathematically.

The general format of a digital communication channel model is depicted in figure 2.1.

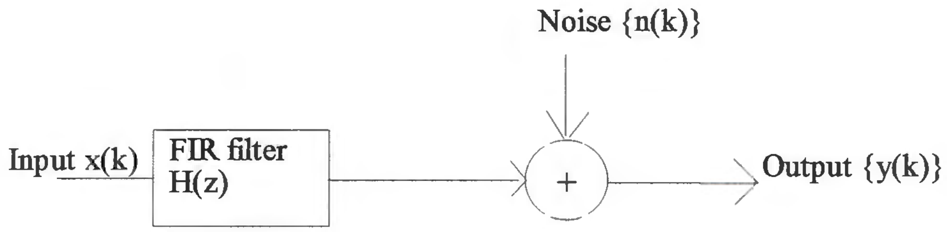


Figure 2.1: A Communication Channel Model

2.3 The Equalisation Problem: Inverse Modelling

It is evident that, in order to receive digital data without error over a communication channel subjected to ISI, a means of negating the effects of ISI is needed. The device used to accomplish this is the equaliser.

The problem of channel equalisation is generally categorised as an inverse modelling problem. That is, given a specific channel $H(z)$, construct a filter that approximates the inverse of the channel $H(z)^{-1}$. In the ideal case the overall response of the channel and the inverse filter (equaliser) will be unity. The data at the system output will be exactly the data that entered the system, delayed possibly by a time τ. There are many reasons, however, why it is not possible (or practical) to construct an inverse filter for this application. The interested reader should refer to Qureshi [3] for a concise coverage of the topic.

A simple explanation of the problem is as follows. Consider a simple channel modelled by the transfer function $H(z)=1+az^{-1}$ (the received symbol is comprised of the current symbol plus a times the previous symbol). The inverse of this channel is given by $H(z)^{-1}$. Consider the pole-zero plot of $H(z)^{-1} = z/(z+a)$ in the z-domain (see figure 2.2). There is a pole located at $-a$ and a zero at the origin. The non-zero pole will result in the impulse response having a term of the form $a^nU(n)$ [4] ($U(n)$ is the unit step function), which will result in the filter having an infinite impulse response. Thus in order to construct the inverse filter of such a channel a filter with an infinite impulse response (IIR) is required.

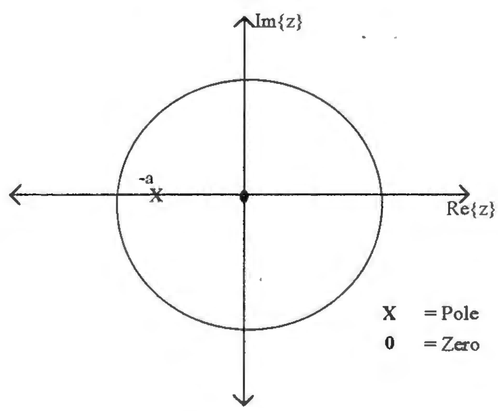


Figure 2.2 : A Pole-zero Plot for $H(z)^{-1}=z/(z+a)$.

The reasons why IIR filters are not implemented, specifically in adaptive systems are as follows [5]:

- IIR filters are recursive in nature, hence the algorithms used to adapt them are more computationally complex than those for FIR adaptation.
- The algorithms used to adapt IIR filters are not guaranteed to find the optimum solutions, due to the fact that the error-surface is non-quadratic and will possibly contain local minima.

* Due to the recursive nature of IIR filters, stability is not guaranteed. This is especially problematic during the adaptation phase, where adaptation of filter parameters could move poles out of the unit circle (z-domain) and cause instability.

It is possible to approximate this inverse with a sufficiently large order FIR filter. One such filter is known as a Zero Forcing (ZF) equaliser. This type of equaliser suffers from noise enhancement at frequencies with high attenuation, which limits its usefulness [3].

Since it is problematic to construct a filter with an inverse response to that of the channel, equalisers are designed to maximise performance by minimising some cost function (criterion). One such cost function is the *mean-square error*.

2.4 The Adaptive Linear Transversal Filter

The Adaptive Linear Transversal Filter (LTF or LTE) is a time-varying non-recursive digital adaptive filter. This filter structure is fundamental in many adaptive signal processing applications. Due to it's non-recursive structure, it is a relatively simple device to understand and analyse. It is for this reason and the existence of straightforward learning algorithms, that this device appears in many adaptive systems [2].

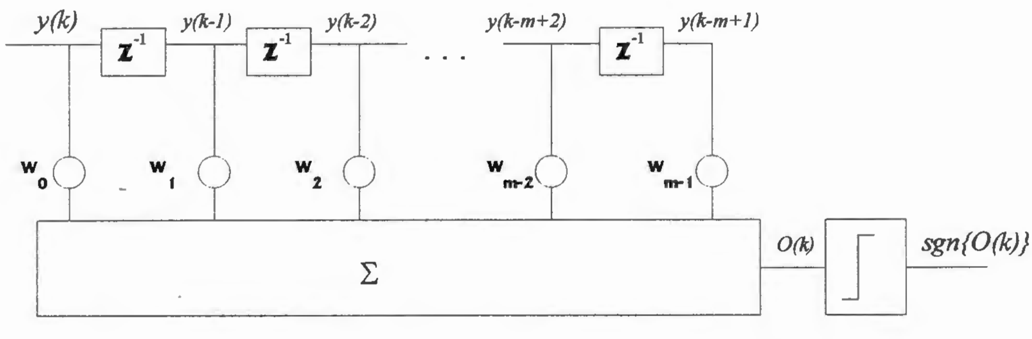


Figure 2.3: The Structure of an Adaptive Linear Transversal Filter/Equaliser

Figure 2.3 shows the structure of the LTF. The output of the filter, before the threshold element is simply a weighted sum of the input vector $[y(k)\ y(k-1)...y(k-m+1)]^T$. The input vector is simply the current channel output and $m-1$ previous channel outputs, where m is the filter order.

The weights of the filter are set or trained to minimise the *mean-square error*. In other words the sum of the squares of all the ISI terms and the noise power at the output of the filter is minimised.

The output before the decision function (usually *sgn* function) is given by the following equation.

$$o(k) = \sum_{i=0}^{m-1} w_i y(k-i) = w_0 y(k) + w_1 y(k-1) + ... + w_{m-1} y(k-m+1) \qquad (2.1)$$

Where the w_i's are the filter weights and the $y(k)$'s are the inputs to the filter. From the expansion of the above equation, it can be seen that the output before the decision function is simply a linear combination of the input vector elements. The output is then passed through a decision function (device) such as $sgn(o(k))$ which assigns a value of ± 1, depending on whether the output $o(k)$ is greater or less than zero (assuming a binary transmission alphabet).

2.5 Equalisation as Pattern Classification

The fact that the final output of the above filter is the output of a decision function (or classifier) suggests that it is possible to view equalisation as a classification problem rather than an inverse-modelling problem. This view has become prominent over the past few years [1][6][7] *et al.*

In this formulation of the equalisation problem the equaliser in conjunction with the decision function (device) is viewed as a pattern classifier. The role of this equaliser is to correctly classify the input data (channel output) as belonging to one of M symbols.

In this dissertation only the simple binary case ($M=2$) will be considered. It is trivial, however to generalise the argument to higher order alphabets.

2.5.1 An Illustrative Example

The formulation of equalisation as a pattern classification problem is best understood by considering an example which is widely quoted in the literature [1][6][7] *et al.*

Consider a channel that is modelled by the transfer function $H(z)=1+0.5z^{-1}$. It will be assumed that the transmitted data is chosen from a finite alphabet of symbols that may take on the value of $\{-1,+1\}$ ($M=2$). Assume further that the symbols in the alphabet are transmitted with equal probability, the sequence of transmitted symbols is *i.i.d* (independently identically distributed). The channel has two coefficients, therefore the output of the channel $y(k)$ is composed of two components one from the symbol transmitted at time k and one from the symbol transmitted at time $k-1$. In other words ISI extends over two symbols ($N=2$). The channel output can be written in vector notation as: $\mathbf{y}(k)=\mathbf{H}\mathbf{x}(k)+\mathbf{n}(k)$ where $\mathbf{y}$ is a vector of channel outputs (consisting of m elements), $\mathbf{x}$ is the channel input and $\mathbf{n}$ is the additive noise. $\mathbf{H}$ is a m by $m+N-1$ Toeplitz matrix of channel coefficients. The above equation can be written in matrix notation as follows:

$$\begin{vmatrix} y(k) \\ y(k-1) \end{vmatrix} = \begin{vmatrix} h_0 & h_1 & 0 \\ 0 & h_0 & h_1 \end{vmatrix} \begin{vmatrix} x(k) \\ x(k-1) \\ x(k-2) \end{vmatrix} + \begin{vmatrix} n(k) \\ n(k-1) \end{vmatrix} \tag{2.2}$$

Where $y(k)$ is the current channel output, $y(k-1)$ is the previous channel output, the $x(k)$'s are the channel inputs at time k, $k-1$ and $k-2$. The h's are the coefficients of the channel transfer function and the $n(k)$'s are the additive noise component.

The channel input vector $x(k)$ consists of three elements which can take on any of two possible values ($+1$ or -1). Thus the output vector is one of 2^3 possible values (states).

Table 2.1 illustrates these possible states (note the additive noise component is ignored here).

The Channel Input			**The Channel Output**	
x(k)	x(k-1)	x(k-2)	y(k)	y(k-1)
-1	-1	-1	-1.5	-1.5
-1	-1	1	-1.5	-0.5
-1	1	-1	-0.5	0.5
-1	1	1	-0.5	1.5
1	-1	-1	0.5	-1.5
1	-1	1	0.5	-0.5
1	1	-1	1.5	0.5
1	1	1	1.5	1.5

Table 2.1 : Possible states for channel $H(z)=1+0.5z^{-1}$

If the current channel output $y(k)$ is plotted versus the previous channel output $y(k-1)$, a graph is obtained as in figure 2.4. Ignoring the effects of noise, it can be seen that there are eight discrete states. In order to determine whether a $+1$ or -1 symbol was transmitted all that is needed is to determine to which state the current input vector $[y(k)\ y(k-1)]^T$ belongs. Since each state is caused by a particular input pattern it will thus be possible to make the distinction between a $+1$ and a -1 transmitted symbol. For example if $y(k)=-1.5$ and $y(k-1)=-0.5$ table 2.1 shows that the transmitted symbol $x(k)=-1$.

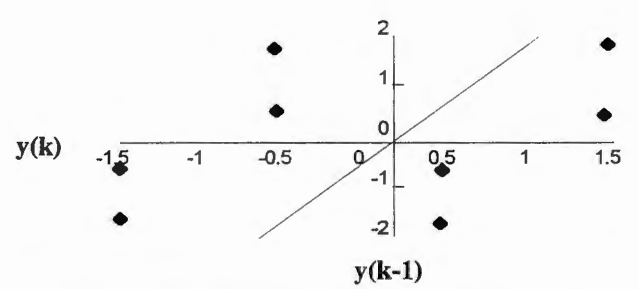


Figure 2.4: Channel Output and Decision Boundary *(m=2,N=2)* for $H(z)=1+0.5z^{-1}$

If an equaliser is constructed with two inputs, one for the current output of the channel and one for the previous channel output, the input to the equaliser is the vector $y=[y(k)\ y(k-1)]^T$. Table 2.1 shows that all output states to the left of the origin in figure 2.4 are caused by a $x(k)=-1$ whereas all the output states to the right are caused by a $x(k)=+1$. If the equaliser is able to classify the input as belonging to a particular subset, the current transmitted symbol $x(k)$ may be determined.

2.5.2 The Adaptive Linear Transversal Filter as a Linear Classifier

Consider the adaptive linear transversal filter (equaliser) discussed in section 2.4. The output of the equaliser before the decision device is given by $w_0y(k)+w_1y(k-1)+...+w_{m-1}y(k-m+1)$. This is applied to a decision device (threshold element). If the resultant output is greater than zero a decision that a $+1$ symbol was transmitted is made. If the output is less than zero a decision that a -1 symbol was transmitted is made. The equaliser thus effectively forms a decision boundary given by equation 2.3. Figure 2.4 illustrates a possible decision boundary for the channel $H(z)=1+0.5z^{-1}$.

$$w_0y(k)+w_1y(k-1)+...+w_{m-1}y(k-m+1)=0 \tag{2.3}$$

The decision boundary formed by the adaptive linear transversal filter is a hyperplane in the input space (see equation 2.3). For the channel $H(z)=1\ +0.5z^{-1}$ the states associated with a $+1$ and those associated with a -1 are linearly separable; a single hyperplane can be passed through the input space to separate the two classes of states. If the states were not linearly separable the linear transversal filter would not be able to correctly classify the input. Linear separability is an important concept in pattern classification and will discussed further in section 3.3.

2.6 Optimal Decision Boundaries by Bayesian estimation

Once again we consider the channel $H(z)=1+0.5z^{-1}$: The linear transversal filter is able to form a decision boundary (see figure 2.4) since the input states are linearly separable. Clearly, however the decision boundary created by the linear equaliser is sub-optimal. In the presence of noise, states closer to the boundary have a higher probability of mis-classification than states further away. The optimal decision boundary is one in which all states have an equal probability of mis-classification. Such a boundary is non-linear and cannot be realised by any linear filter [6].

2.6.1 Bayesian Estimation

An increasingly popular approach, which has been shown to produce optimum decision boundaries, is Bayesian Estimation [6][8].

This approach exploits the statistical nature of the problem: the fact that the transmitted symbols are selected from a finite alphabet with a specific probability. The estimation is based on the observed output vector: given that the output y was observed, what is the probability that this observation was caused by the input symbol $x(k-\tau)$? For the binary case the decision boundary is given by the following equation.

$$P(x(k)=+1 \mid \text{received } y) = P(x(k)=-1 \mid \text{received } y) \qquad (2.4)$$

This is interpreted as follows: given that y was received, the decision boundary is the locus of points for which the likelihood that a $+1$ or -1 had been transmitted is equiprobable.

Using Bayes rule [9] and the fact that it is assumed that the transmitted symbols ($+1$ or -1) are transmitted with equal probability, the equation above can be written in terms of the *aposteriori* probabilities. In other words given that a $+1$ or -1 symbol was transmitted what is the probability of receiving y ?

P(received y| $x(k)$=+1) = P(received y| $x(k)$=-1) ,

i.e. P(received y| $x(k)$=+1) - P(received y| $x(k)$=-1) = 0 (2.5)

Where the symbol | should be read as "given that".

The probabilities in the above equation are conditional probabilities. If it is assumed that additive white Gaussian noise with zero mean and variance σ^2 is added during the transmission of a data symbol. Each observed output vector y will be Gaussian distributed about one of the possible output states. The decision boundary is the **intersection**, of the sum, of all the Gaussian conditional density functions about all the states associated with a $+1$ symbol and the states associated with a -1 symbol. The equation of this decision boundary is given by [1]:

$$\sum_{y \in S+} \frac{1}{(2\pi\sigma^2)^{m/2} m} \exp\left(\frac{-(\| \mathbf{y}(k) - \mathbf{y}_i \|)^2}{2\sigma^2} \right) - \sum_{y \in S-} \frac{1}{(2\pi\sigma^2)^{m/2} m} \exp\left(\frac{-(\| \mathbf{y}(k) - \mathbf{y}_i \|)^2}{2\sigma^2} \right) \tag{2.6}$$

Where $S+$ is the set of states belonging to the $+1$ class and $S-$ is the set of states belonging to the -1 class. The y_i's are the centres of the conditional density functions (the channel output states when no noise is present) and the $y(k)$'s are the received signal points. m is the order of the equaliser. An explanation of equation 2.6 is as follows:

The term on the left represents the conditional distribution of the received vector y (the input to the equaliser) given that a $+1$ symbol was transmitted ($x(k)=+1$). There may be a number of possible states associated with a $+1$ transmitted symbol, hence the summation over all the states. Likewise the term on the right represents the conditional distribution of the received vector y given that a -1 symbol was transmitted.

It is important to note that the Bayesian decision boundary is dependent on :

- The probability of transmission of the symbols in the symbol alphabet.
- The distribution of the additive noise process corrupting the transmitted symbol.
- The variance of the noise process.

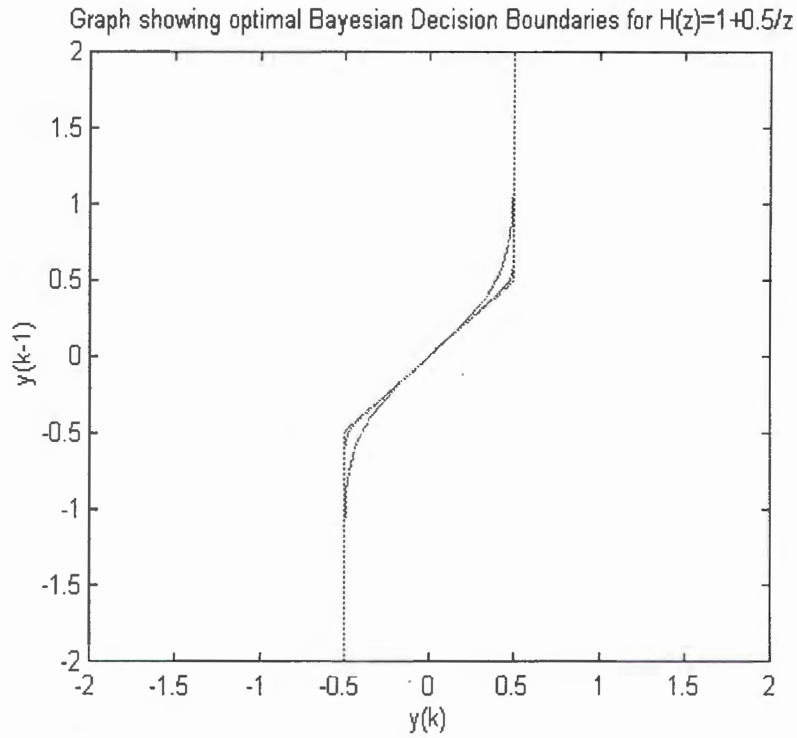


Figure 2.5: Graph Depicting Three Bayesian Decision Boundaries for Noise variance 0.01,0.0625 and 0.25 for the channel $H(z)=1+0.5z^{-1}$.

Figure 2.5 illustrates three Bayesian decision boundaries formed for the channel $H(z)=1+0.5z^{-1}$ for three different noise variances. These boundaries should be compared to the boundary created by the LTE (which is necessarily a hyperplane). In low noise conditions the decision boundary formed by Bayesian Estimation is piecewise linear, becoming more "rounded" as the noise increases. This is due to the fact that as the noise increases the variance of the conditional Gaussian density functions increases and thus their intersection occurs "higher up", which when projected into the input space creates a more "rounded" decision boundary (see equation 2.6).

Chapter 3

Neural Networks For Channel Equalisation

3.1 Introduction

There have been many applications over the past few years to which neural networks have been applied. In the communication field alone the list of successful applications is numerous. These applications include Traffic and connection admission control in ATM (asynchronous transfer mode) switching [10], image compression and decoding [11]. This chapter presents a brief overview of neural networks and their application to channel equalisation.

Many architectures have been proposed for the equalisation of various types of communication channels. Many of the most recent proposed architectures employ some form of neural network as classifiers in the equaliser. For example in [12] a recurrent neural network architecture is proposed for the adaptive equalisation of non-linear channels. In [13] a modified decision feedback equaliser employing a radial basis function network is proposed. And in [14] a multi-layer perceptron decision feedback equaliser is suggested. In all cases the proposed architectures outperform the conventional linear techniques, especially in adverse environments, such as poor signal-to-noise ratio signals or non-linear channels. This performance is at the expense of computational and implementation complexity.

In this chapter neural networks will be analysed with application to channel equalisation. Factors that contribute to the success of neural networks in this application will be discussed as well as certain short-comings. A comparison will be made between networks employing **localised** basis functions and those employing **non-localised** basis functions.

3.2 Neural Networks as Pattern Classifiers

As mentioned, neural networks have been applied to many types of problems. In this dissertation the focus will be on neural networks as pattern classifiers as this relates directly to channel equalisation. Researchers have also derived numerous architectures for the various applications; in this dissertation the focus will be on simple feed-forward networks.

3.2.1 Structure of a General Feedforward Network

The general feed-forward neural network comprises three distinct parts. The first of these is known as the input layer. The input layer is connected via a set of weights to a hidden layer; there may be any number of subsequent hidden layers all interconnected via a set of weights. The final part is the output layer the output of which is usually a weighted sum of the output of the final hidden layer. A threshold element may classify the network output as being a *+1* or *-1*. Figure 3.1 illustrates this architecture.

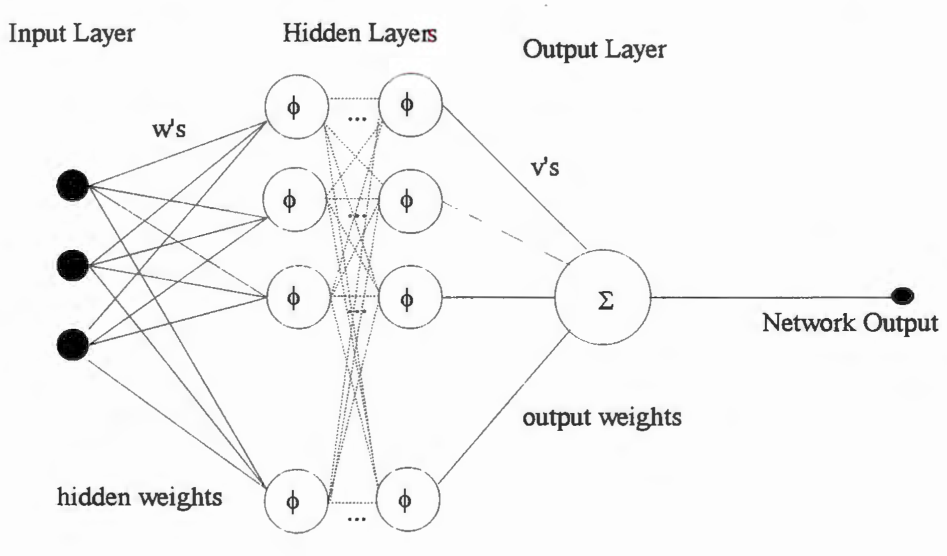


Figure 3.1: Structure of a General Feed-forward Neural Network

The hidden layer is the heart of the neural network. Each hidden layer node implements a non-linear activation function (basis function).

The output of each hidden neuron has the following form:

$$y_i = \phi\left(\sum_{j=0}^{m-1} x_j * w_j\right) \quad \text{for i} = 0,1\ldots,\text{N} - 1 \tag{3.1}$$

Where x_j is the j^{th} input, w_j is the associated (hidden) weight and m is the number of inputs (order). ϕ is the non-linear activation function and N is the number of hidden layer nodes. The output of the neural network is a weighted sum of all the hidden layer outputs (all the y_i*'s*).

$$Z_k = \sum_{i=0}^{N-1} y_i * v_i \tag{3.2}$$

Where Z_k is one of k output nodes, y_i is one of i hidden node outputs, the v_i*'s* are the associated weights and N is the number of hidden layer nodes. The output Z_k may then be fed to a decision device that classifies the output as *+1* or *-1* depending on whether the input to the device is greater or less than zero (assuming binary classification).

3.2.2 The Multi-layer Perceptron

The multi-layer perceptron (MLP) is a general feedforward network. The non-linear activation functions in the hidden layers of the network are non-localised. Common non-linear functions are the *tanh* or *sigmoid* functions. Figure 3.2 is a plot of the *tanh* function.

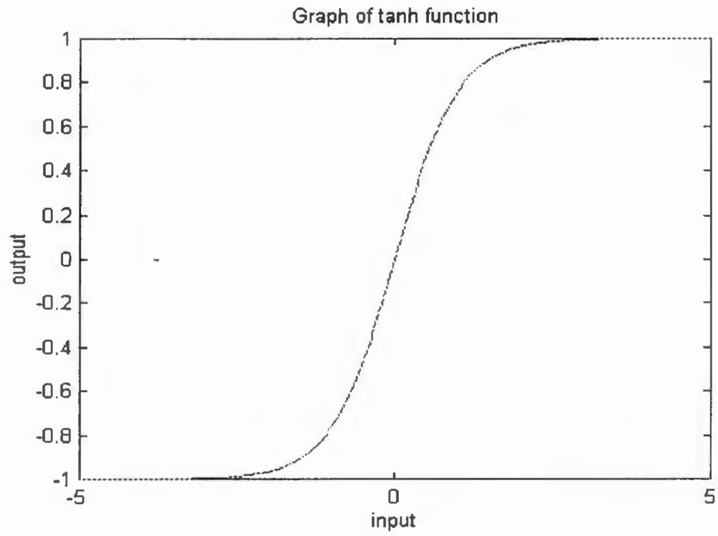


Figure 3.2 : Plot of *tanh* function

The MLP may have any number of hidden layers. Generally all the non-linear activation functions in the network are the same for a given network.

3.2.3 The Radial Basis Function Network

The Radial Basis Function Networks (RBFN) were originally developed for data interpolation [15], the idea being that a weighted sum of basis functions could approximate any *N*-dimensional surface to the required degree of accuracy (by using sufficiently large numbers of basis functions).

A RBFN also has the structure of a general feedforward network. In general a RBFN has only one hidden layer, the non-linear activation functions employed are localised as opposed to the non-localised activation functions of the MLP. There are no input layer weights as for the MLP. Thus the output of a RBFN is given by equation 3.3.

$$F(\mathbf{x}) = \sum_{i=0}^{N-1} w_i \varphi(\|\mathbf{x} - x_i\|) \tag{3.3}$$

The $\varphi(\|\mathbf{x} - x_i\|)$'s are a set of N non-linear functions known as radial basis functions. $\|\ \|$ denotes norm, usually taken to be Euclidean. The points x_i, where $i=0,1,\ldots,N-1$, are the centers of the radial basis functions and x the input vector. The w_i's are the output weights.

The output of a given basis function is determined by the Euclidean distance between the input vector and the centre of the basis function (point about which the basis function is located). The localised functions are usually chosen to be radially symmetric. Possible choices of basis functions include: multi-quadratics, inverse multi-quadratics or Gaussian kernels. The equation for the Gaussian kernel is given by:

$$\varphi(\mathbf{x}) = \exp\left[-\frac{\|\mathbf{x} - x_i\|^2}{2\sigma^2}\right] \tag{3.4}$$

The variance σ^2, determines the radius of influence of each basis function. In other words the radius determines how quickly the basis functions approach zero as x tends to infinity. The x_i's are the centres of the radial basis functions. It will become apparent why the focus of this dissertation is on RBFN's employing Gaussian kernels.

3.3 Cover's Theorem on the Separability of Patterns

The ability of neural networks to perform complex classification problems can be intuitively understood on the basis of Cover's theorem of the separability of patterns. Briefly Cover's theorem states that a complex pattern-classification problem cast in a high-dimensional space is more likely to be linearly separable than in a low dimensional space [15]. If the patterns are linearly separable it is relatively easy to classify the patterns into one or other sets, since a hyperplane can be passed through the space. Patterns in the space located above the hyperplane can be classified as belonging to some set S^+. Patterns in the space located below the hyperplane can be classified as belonging to some other set S^-.

This concept is more clearly understood by considering the following example. Consider the four points in 2-dimensional space illustrated in figure 3.3. Clearly it is not possible to separate (classify) these points with a linear device. Only a non-linear boundary can separate the two classes of patterns.

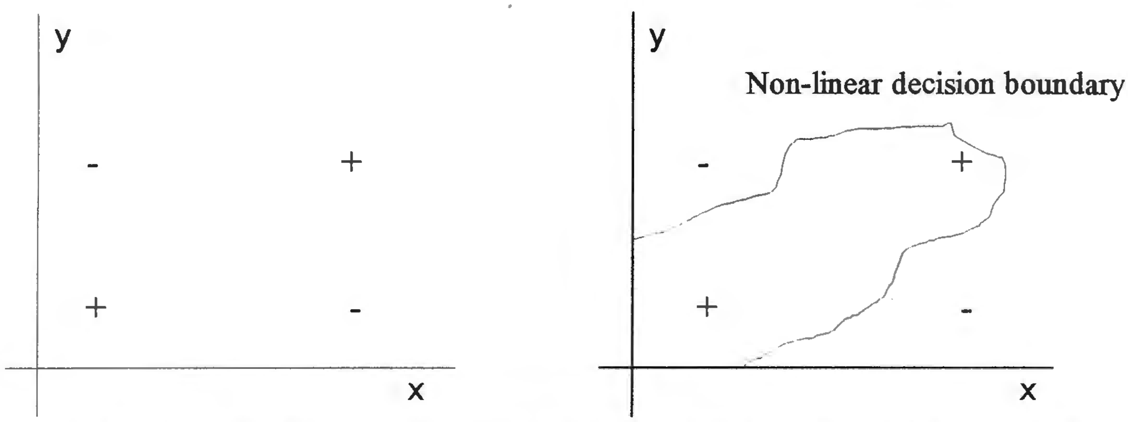


Figure 3.3 : Graph showing non-linearly separable points and a possible non-linear decision boundary

If these four points are mapped into some higher dimensional space by a non-linear mapping, they are likely to be linearly separable in this higher-dimensional space. The higher the dimensionality of the space the greater the likelihood of linear separability. Figure 3.4 illustrates this concept.

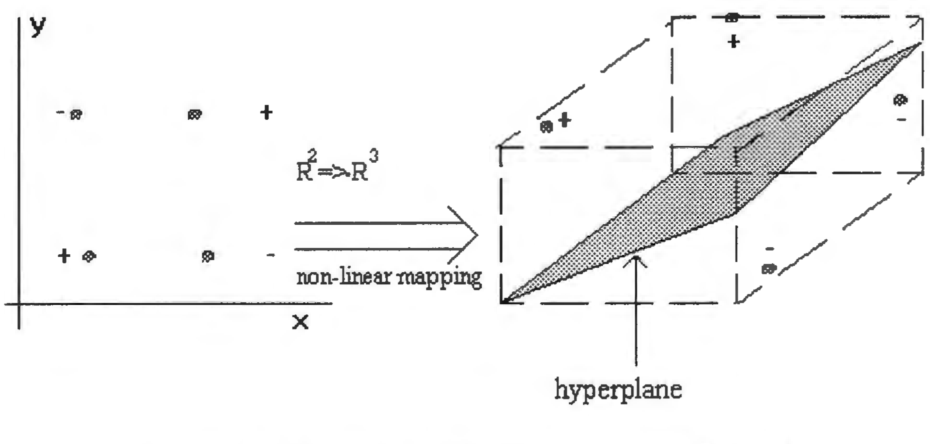


Figure 3.4 : a non-linear mapping from $\Re^2$ to $\Re^3$

It is the task of the neural network to perform this non-linear mapping and to create the decision boundary in N-dimensional space. Equations 3.1 and 3.2 show that the m inputs are mapped via some non-linear activation function into an N-dimensional space.

The neural network output is simply a weighted sum of the N-dimensional non-linear mapping, fed through a threshold element, thus the decision boundary is seen to be a hyperplane in N-dimensional space.

$$(v_0 \times y_0) + (v_1 \times y_1) + \ldots + (v_{N-1} \times y_{N-1}) = 0 \tag{3.5}$$

Where the v_i*'s* are the output weights and the y_i*'s* are the outputs of the N hidden neurons. In the input space the decision boundary is a non-linear function; the hyperplane inverse-mapped (φ^{-1}) to the input space.

3.4 Radial Basis Function Networks as Bayesian Estimators

There is a more elegant explanation of why RBFN's have an application to channel equalisation. The explanation based on cover's theorem of separability of why neural networks are able to solve complex classification problems is equally valid for MLP's as it is for RBFN's. This visualisation, however does not highlight the consequence of localised basis functions as opposed to non-localised basis functions in this application.

In section 2.6 Bayesian Estimation was presented as the optimum strategy for the symbol-by-symbol classification of signals in the presence of noise. The decision boundary is given by the intersection of the conditional density functions of all points belonging to the $+1$ set and those belonging to the -1 set (in the binary case). The conditional density functions are dependent on the noise model. If the noise is assumed to be Additive White Gaussian Noise, the conditional density functions are Gaussian kernels (see equation 2.6). If Gaussian kernels are used as the non-linear functions in a RBFN, a RBFN can be used to directly model the Bayesian Estimation Formula (equation 2.6).

Consider the Radial basis Function Network Equaliser (RBFNE) shown in figure 3.5 The equaliser consists of three distinct parts. The tapped delay line provides the input to the Radial Basis Function Network. The Radial Basis Function Network's output is fed to a threshold element.

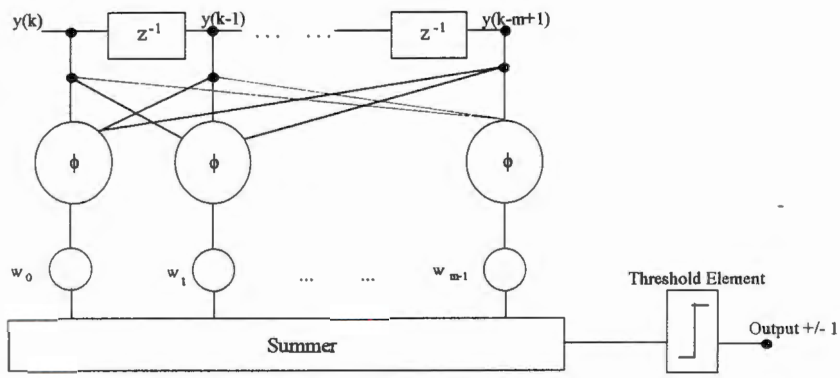


Figure 3.5: Diagram of a Radial Basis Function Network Equaliser.

The decision boundary created by the network, if Gaussian basis functions are employed is given by equation 3.6.

$$\sum_{i=0}^{N-1} w_i \times \exp\left[-\frac{\|y - c_i\|^2}{2\sigma^2}\right] = 0 \tag{3.6}$$

Where y is the input to the RBFN and c_i is the ith centre. N is the number of localised basis functions. Note the scaling factor has been omitted here (see section 4.3.4 for an explanation). Comparing this equation to equation 2.6 it can be seen that should the centres of the basis functions (c_i's) be trained to coincide with the possible channel output states, the variance σ^2 trained to equal the actual noise variance and the weights correctly determined, the RBFNE exactly models the Bayesian Estimation Formula. This is the primary reason that RBFN's have been suggested as classifiers in an equaliser rather than the Adeline or MLP; the other reasons have to do with training and complexity which will be discussed in the next chapter.

Figure 3.6 illustrates eight Gaussian density functions. Each density function is located about a possible channel output state. The intersection of the $S+$ set with the $S-$ set defines the Bayesian Decision boundary. The RBFNE models these Gaussian functions and hence is able to closely approximate the Bayesian decision boundary.

Graph showing Gaussian conditional density functions

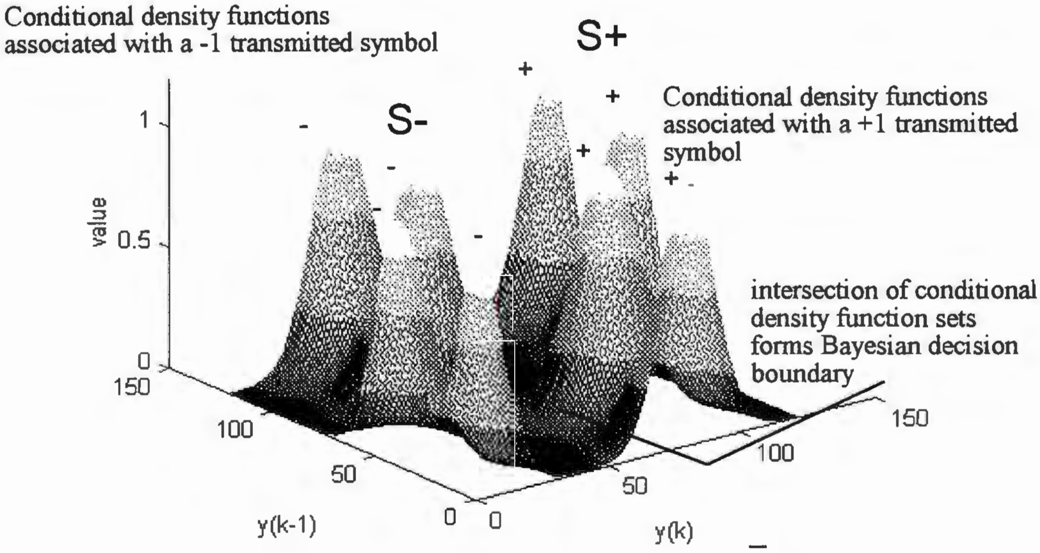


Figure 3.6: A graphical illustration of eight Gaussian Conditional density functions centred about eight channel states.

3.5 Discussion

The RBFNE structure explicitly models the communication channel, which includes the statistics of the channel. By placing Gaussian kernels on the possible received states, the structure of a RBFNE is an attempt to directly model the fact that the receiver will receive a signal that is Gaussianly distributed about one of the possible states of the channel output (recalling that these states are caused by the ISI present in the channel). Doing this directly models the Bayesian Estimation criterion if the noise is in fact Gaussian.

The MLP on the other hand can only implicitly arrive at the Bayesian decision boundary. However the MLP makes no assumptions about the distribution of the noise. The estimate of the Bayesian Decision boundary arises due to the fact that the input is non-linearly mapped into a high dimension space, and a hyperplane is then passed through this space to separate the input into two classes. It is the projection (inverse mapping) of this hyperplane into the input space that forms the estimate of Bayesian decision boundary.

Chapter 4

Training the Equaliser Structures

4.1 Introduction

A number of different algorithms can be employed to train the equaliser structures; the equaliser structures in question are the standard Adaptive Linear Transversal Equaliser (LTE) the Radial Basis Function Network Equaliser (RBFNE) and the Multi-layer Perceptron based Equaliser (MLP). The performance of each structure is dependent on the algorithms used to train the equaliser. When considering training algorithms the critical issues are: the computational complexity of the training algorithms, the convergence of the algorithms to the optimum solution and the *a priori* knowledge required to implement the equalisers. These issues will be discussed in this section.

4.2 Training the Adaptive Linear Transversal Equaliser

The Adaptive Linear Transversal Equaliser was described in section 2.4. The Equaliser is trained with the Least-Mean Squares (LMS) algorithm. The LMS algorithm minimises the instantaneous estimate of the cost function *J(n)* [16]. The cost function is the *mean-squared error* given by equation 4.1. A feature of the LMS algorithm is it's simplicity; it does not require matrix inversion or prior knowledge of the correlation matrix (as for steepest descent methods).

$$J(n) = \frac{1}{2}\mathrm{E}[e^2] \tag{4.1}$$

Where E[] is the expectation operator and the error e is the difference between some desired response d and the actual response y of the system. The desired response is obtained from a replica of the training sequence at the receiver.

$$e = d - y \tag{4.2}$$

Recall that the output of the equaliser before the threshold element is given by equation 4.3.

$$y = \sum_{k=0}^{m-1} w_k \times x_k \tag{4.3}$$

Where m is the filter order, the w_k*'s* are the weights of the equaliser and the x_k*'s* are the input to the equaliser.

The LMS algorithm forms an estimate $\hat{\mathbf{w}}(n)$ of w(n) at each iteration, as the number of iterations approaches infinity the estimate $\hat{\mathbf{w}}(n)$ performs a random walk about the optimum solution [15] .The LMS algorithm is summarised as follows [16]:

<u>Summary of the LMS algorithm:</u>
• Initialise Weights : $\hat{w}_k(n) = 0$ For $k=0,1,2,\ldots,m-1$
• For time $n=1,2,\ldots,$ *Training Length* do the following:
1. Calculate the output : $y(n) = \sum_{k=0}^{m-1} \hat{w}_k(n) \times x_k(n)$
2. Calculate the error : $e(n) = d(n) - y(n)$
3. Update the Weights : $\hat{w}_k(n+1) = \hat{w}_k(n) + \eta \times e(n) \times x_k(n)$

Table 4.1: Summary of the LMS Algorithm

4.2.1 The *a priori* Knowledge Required to Train the Adaptive LTE.

In order to successfully equalise a channel with the Adaptive Linear Transversal Filter trained by the LMS algorithm, there are only two major parameters that need to be established. The first of which is the channel order. In terms of computational complexity versus performance there is an optimum filter order for a specific channel order. Gibson *et al* shows in [6] that as the order of the equaliser increases the total power of the noise on the equaliser input increases this tends to diminish any advantage gained by increasing the order of the filter. This is particularly true in high noise environments where the signal-to-noise ratio is less than about 10db. Note this does not imply that increasing the filter order beyond the optimum diminishes the overall performance, merely that there is no advantage in doing so since the overall performance gain is minimal at the expense of a decrease in efficiency. The second parameter concerns the LMS algorithm itself. The LMS algorithm requires one parameter η which controls the rate at which the weights are updated. This parameter affects the rate of convergence of the algorithm and is problem-dependent. If η is too large the weight estimates will tend to oscillate about the optimum. If η is too small the weight estimates will tend to converge very slowly towards the optimum.

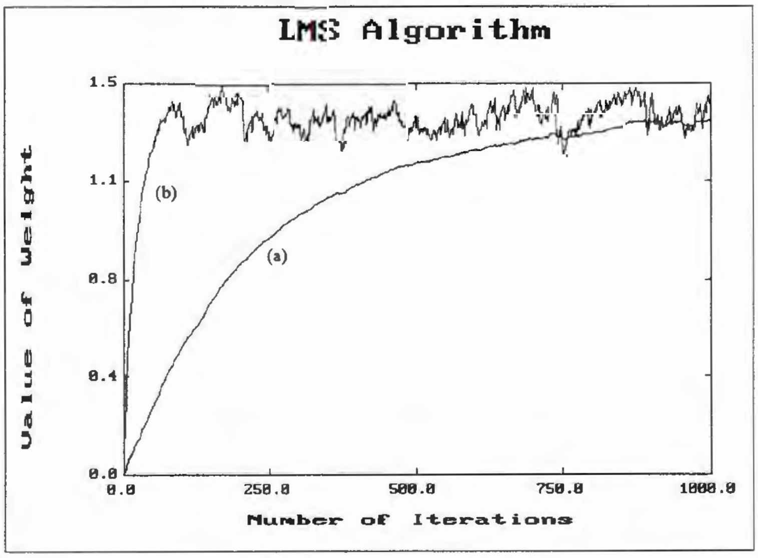


Figure 4.1 : Graph showing how the learning rate effects the convergence of the weights in the LMS algorithm. (a) Learning rate too small (b) Learning rate too large.

4.2.2 Convergence of the LMS Algorithm

The convergence of the LMS algorithm towards the optimum Weiner solution is dependent on the learning parameter η. It is possible to determine bounds on η for which the weight vector mean will converge to the optimum weight vector. The speed of adaptation and the noise in the weight vector solution are determined by the size of η within the bounds. For the transversal filter the bounds on η is given by equation 4.4 [2].

$$0 < \eta < \frac{1}{(L+1)(\text{signal power})} \tag{4.4}$$

Where L is the number of delay elements on the transversal Equaliser. Equation 4.4 is intuitively explained as follows: If the signal power is large and an error is made, it is likely that the error will be large, it is desirable to keep η small to avoid the weight updates from being excessively large. The converse is true if the signal power is small. For a particular choice of η within the bounds given by equation 4.4 the LMS algorithm will converge to the optimum Weiner solution within a certain time with a certain mis-adjustment. The mis-adjustment in an adaptive process is a measure of how closely the adaptive process tracks the optimum Weiner solution. The mis-adjustment is defined as the ratio of the excess *mean-square error* to the minimum *mean-square error*. A complete treatment of the convergence characteristics of the LMS algorithm is given in [2],[15] and [16] amongst others.

4.2.3 Complexity of the LMS algorithm

If the filter order is m, then there will be m weights to train. Disregarding the negligible computation required for the initialisations, each computation of the output requires m multiplication's, calculation of the error e requires one subtraction and the computation of the new weights requires m additions and $2m$ multiplication's. The complexity of the LMS algorithm is thus $O(m)$.

4.3 Training the Radial Basis Function Network

There are a number of approaches that can be used to train the radial basis function network equaliser. There are three sets of parameters that need to be trained as opposed to the single set of parameters (the weights) of the LTE. However, each parameter set may be trained independently. There are two main approaches that can be used to train the RBFNE. The first approach is to use gradient descent algorithms, which minimise the *mean-square error*. The Stochastic Gradient Algorithm is one such algorithm [17]. The second approach is a modelling approach [18]. This approach exploits the fact that radial basis functions model conditional density functions. It trains the network to model the Bayesian Estimation formula and is independent of any cost function (such as the *mean-square error*).

In either case the three parameter sets that need to be trained are:

- The centres of the radial basis functions (Gaussian kernels); these are the points about which the radial basis functions are located.
- The variance of the basis functions, which determines their radius of influence.
- The output weights, which determine the relative contribution of each of the basis functions output.

Gradient descent methods adapt all the parameters at each iteration in order to minimise the cost function. The modelling approach is able to exploit the independent nature of the parameter sets. In the following discussion it will be assumed that a modelling approach is used except where the Stochastic Gradient Algorithm is explicitly referred to. The Stochastic Gradient Algorithm will be presented at the end of the discussion for completeness.

4.3.1 The *a priori* Knowledge Required to Train the RBFNE

As for the LTE the channel order must be known or determined. This knowledge together with knowledge of the order of the transmission alphabet (the number of symbols) will determine the number of inputs to the network and the number of radial basis functions required. In the gradient descent approach there are a number of *learning parameters* that must be determined. These are usually determined heuristically.

4.3.2 Training the Centres

Two approaches may be used to train the centres. One approach involves *supervised learning* while the other approach involves *unsupervised learning*. In both approaches a clustering algorithm such as the *k-means* or *adaptive k-means* [19] is used to determine the appropriate cluster centres of the input data. Yet another method involves setting the centres to the required values. This would be possible if the channel states are known. A gradient descent method or an *unsupervised* or *supervised* clustering method could then be used to adapt the centres if the channel is time-varying.

Recall that the cluster centres are in fact the states about which the Gaussian Conditional Density functions are located. In order for the RBFNE to correctly model the Bayesian Estimation Criterion it is important that there is a unique cluster centre for each state and that the cluster centre coincides to within some margin of error with the actual centre (channel state). Any deviation will naturally cause a degradation in performance, since the intersection of the conditional density functions (decision boundary) will be offset.

The *k-means* clustering algorithm starts with the initialisation of the cluster centres to some value (usually set to the first j received vectors). Each subsequent received vector is compared to the current cluster centres to determine which centre is closest to the received data. The closest centre is then updated, and the value of any particular cluster centre at any given time is determined by taking the mean of the points associated with that particular cluster centre.

(a) Unsupervised Approach

In the *unsupervised* approach a clustering algorithm is used to "sort" a set of input data into j cluster centres. There are a number of algorithms that may be used; the most computationally efficient of these is the simple *k-means* algorithm. The problem with this approach is that convergence of the clustering algorithm to j (where j is the number of channel states) discrete cluster centres is not guaranteed. In fact it was observed by simulation that the convergence is highly dependent on the initial locations of the centres. The approach which seems to perform reasonably well is to initialise the centres to the first j input vectors. There is however, still no guarantee of convergence. The algorithm tends to form duplicate centres, causing one or more of the actual cluster centres not to have a centre allocated to it.

The disadvantage with *unsupervised* clustering is that it is an ill-conditioned problem. Consider a cluster of points; it is possible that due to the initial initialisation of the cluster centres, two or more cluster centres converge to the same region (the same cluster of points). The clustering algorithm will then proceed to incorrectly classify one cluster of points as two or more clusters, leaving other clusters devoid of a cluster centre. The problem is exaggerated if the input space has many clusters. As the number of clusters in the input space increases and the clusters become closer to one another, the greater the chance that the clustering algorithm will mis-classify one or more clusters. The advantage with this method is that the centres may be trained "blindly" - no training sequence is needed.

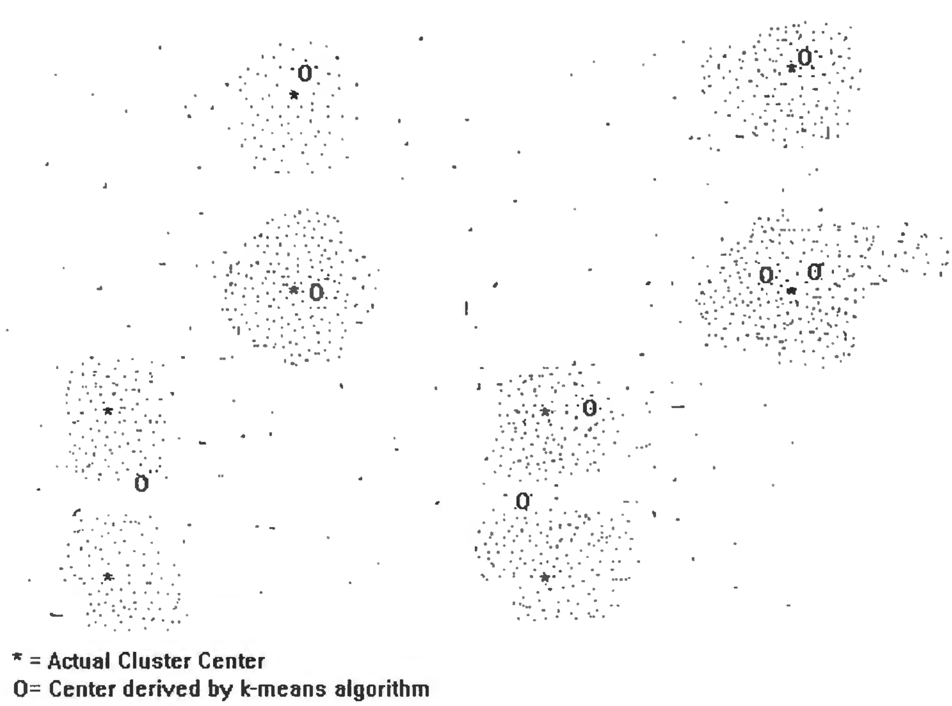


Figure 4.2 : An illustration of the typical convergence characteristics of an *unsupervised* clustering algorithm

(b) Supervised Approach

The second method is a *supervised* clustering method described by Mulgrew *et al* [18]. This method assumes that a training sequence is transmitted and that the receiver has a replica of this training sequence (naturally synchronisation between the transmitter and receiver is assumed). The training sequence is transmitted through the communication channel. The original sequence is corrupted by ISI and AWGN before being received by the receiver. If it is known that there are j channel states (possible received states), the input to the equaliser at any given time will be one of the j channel states corrupted by additive noise. Instead of using the input to the equaliser to directly determine the cluster centres (*unsupervised* method), the replica of the training sequence is used to determine which sequence of bits caused the current received signal. Once this has been determined the input is used to update the **appropriate** centre.

For example consider the channel $H(z) = 1+0.5z^{-1}$ and consider an equaliser with two inputs, one input for the channel output at time k and one for the output at time k-1.

Consider a training sequence *-1,1,1,...* after the transmission of the third bit in the sequence the input to the equaliser will be (see table 2.1):

$$\begin{vmatrix} y(k) \\ y(k-1) \end{vmatrix} = \begin{vmatrix} 1\times 1+1\times 0.5 \\ 1\times 1+(-1)\times 0.5 \end{vmatrix} = \begin{vmatrix} 1.5 \\ 0.5 \end{vmatrix}$$

Note the effects of additive noise have been neglected here. A replica of the training sequence at the receiver is used to determine that the current input is associated with the bit sequence *-1,1,1.* This particular sequence of bits is associated with state five (for example) and hence the (possibly noisy) input $(1.5,0.5)^T$ is used to update the centre associated with state five and only state five's centre will be updated. Thus, irrespective of the amount of noise present or the way in which the centres where initialised, the correct centre will always be updated. The algorithm for updating the centres is as follows:

```
if (s(t) = s_i)                                      */ if the input belongs to state i */
{              c_i(t)=counter_i*c_i(t-1)+y(t);       */ Update the ith centre */
               counter_i=counter_i+1;
               c_i(t)=c_i(t)/counter_i;              */ Normalise the updated centre */
}
```

Table 4.2: Pseudo-code Listing of the Supervised Clustering Algorithm

Where $s(t)$ is the current $log_2 j$ training bits (assuming a binary transmission alphabet), s_i is the *ith* possible state, $c_i(t)$ is the *ith* centre at time *t*.

The advantage of this method is that rapid convergence to *j* unique centres is guaranteed since the decision as to which centre the current input belongs to is based on the training sequence, not on a noisy estimate (i.e. the input itself). It is easy to see that the centres will converge:

Consider figure 4.3, the received data will be Gaussianly distributed about one of the eight possible states (states 0 to 7). Since the equaliser has a replica of the transmitted sequence, the appropriate cluster centre will always be updated. As the number of training bit tends to infinity the cluster centre associated with each state will converge to the correct value, since the received points will be Gaussianly distributed about the channel state with zero mean and variance σ^2. As long as enough training points are transmitted, so that a relatively smooth Gaussian distribution about all the channel states occurs, the cluster centres will converge to the eight channel states.

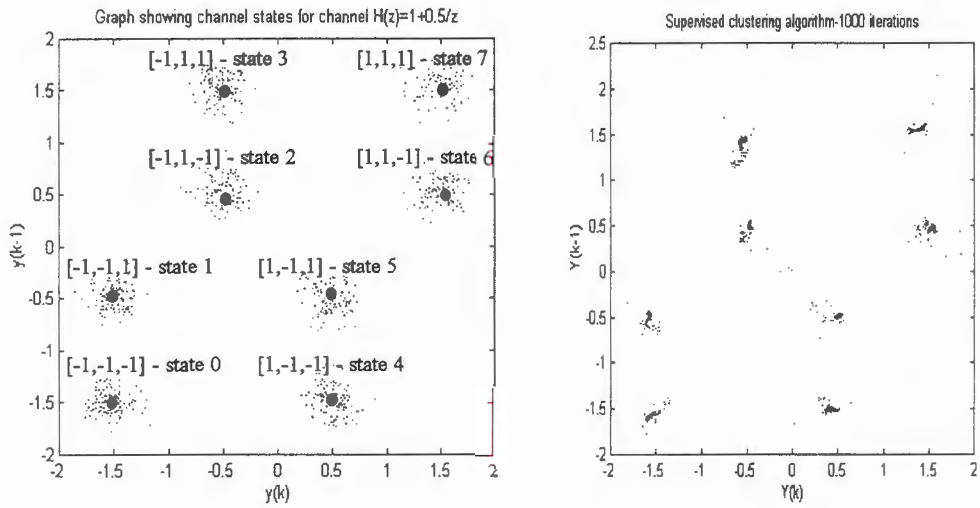


Figure 4.3 : (Left) An illustration of the channel states for the channel $H(z)=1+0.5z^{-1}$ and the bit sequences which caused them. (Right) The locus of centres trained by the supervised clustering algorithm for 1000 iterations.

4.3.3 Estimating the Variance

The variance may be estimated using a Stochastic Gradient method [17]. If however the centres are positioned correctly at the desired channel states, the noise variance is given by the expected deviation of an input associated with the *ith* centre.

$$\sigma_e^2 = \mathrm{E}[\|\mathbf{y}(t) - c_i\|^2 / m] \tag{4.5}$$

Where E[] is the expectation operator, $y(t)$ is the input vector and m is the filter order (the number of inputs). Note this equation follows directly form the definition of variance. The variance can be estimated by the following equation [18]:

$$\hat{\sigma}_e^2 = \left((t-1)\times\hat{\sigma}_e^2(t-1)+\left\|\mathbf{y}(t)-c_i(t)\right\|^2 / m\right)/t \tag{4.6}$$

Which simply computes a running mean of the input variance. Using this method for the simple case where $H(z)=1+0.5z^{-1}$ the following results were obtained after 1000 training samples, the variance was estimated while the centres were being trained. Equation 4.6 produces very accurate estimates of the variance in a relatively short time.

Actual σ^2	0.25	0.16	0.09	0.0625	0.04	0.0225
Estimated σ^2	0.246	0.157	0.089	0.0615	0.039	0.0221

Table 4.3 : A Comparison of the Actual Variance and the Estimated Variance

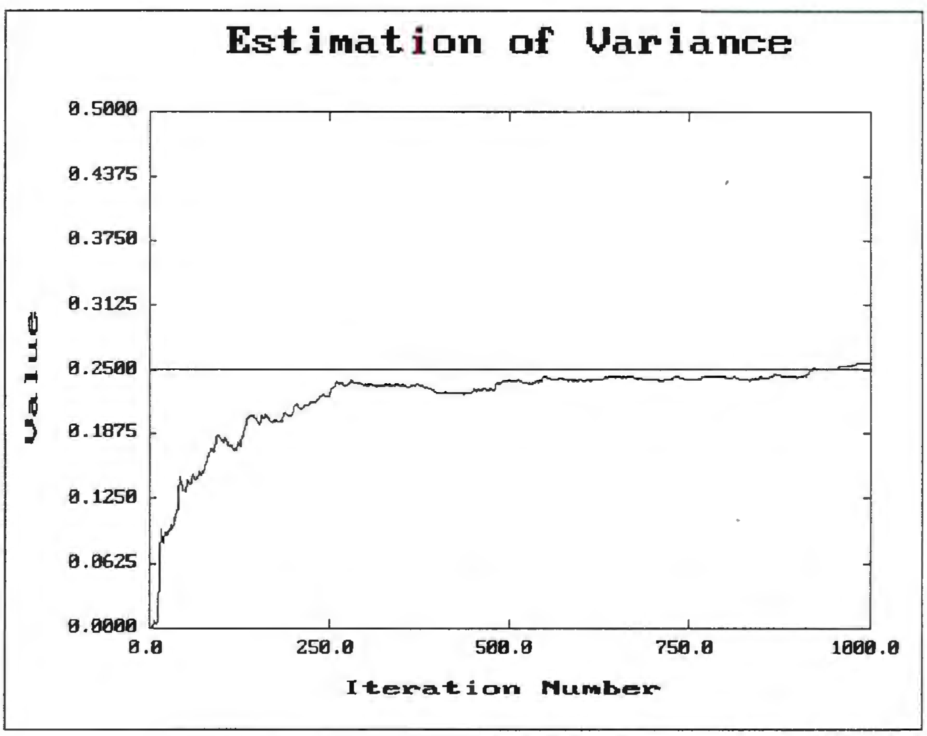


Figure 4.4 : Graph Showing how the Variance Estimate Converges to the Desired Value

Using the above method the variance is estimated and all the radial basis functions in the network use the same estimated variance. In the Stochastic Gradient method, each basis function **may** have its variance updated independently, since in terms of the gradient descent the variance of each basis function is just another parameter to be adapted. The advantage of having one global variance is that any possible error in the estimate of the variance is effectively cancelled out by the other radial basis functions. On the other hand if each variance is updated independently (as in a gradient descent method) there is a possibility that one or more of the variances will not approach the desired value. Figure 4.5 illustrates this phenomena with one of the variances estimated incorrectly. Although the equaliser is still able to correctly classify the input, the decision boundaries formed are sub-optimal.

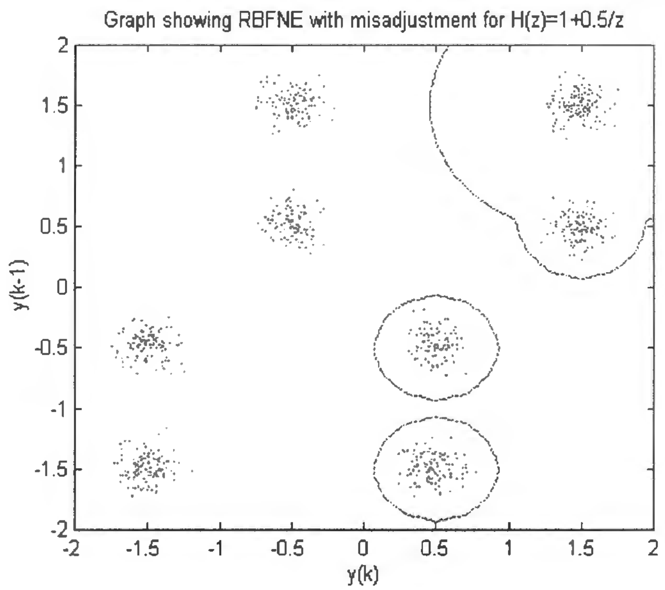


Figure 4.5 : Graph Showing the Decision Boundary formed by a RBFNE with one of the Basis Function Variances set to 0.0625 instead of 0.01.

4.3.4 Training the Weights

The weights may either be "trained" or set. The decision boundary created by Bayesian Estimation depends on the probability of transmission of the symbols in the transmission alphabet. For example if the alphabet consists of two symbols (binary) *{+1,-1}* and the probability of transmitting a *+1* symbol is 80% and that of a *-1* symbol is 20%, the decision boundary will be weighted in favour of the *+1* symbol since it has the higher probability of occurrence. If it is known that the transmitted sequence is *i.i.d* then in the binary case, exactly half the time a *+1* symbol will be transmitted and the other half of the time a *-1* symbol will be transmitted. Thus the conditional density functions associated with the *+1* symbol and those associated with the *-1* symbol will have equal weighting. Thus the scaling factors of $1/[(2\Pi\sigma^2)^{m/2}m]$ in equation 2.6 cancel each other out. If there are *j* channel states, *j/2* of these states will be associated with a *+1* symbol and the other *j/2* with a *-1* symbol. Further, if it is known which channel states belong to which transmitted symbol (set) it is possible to appropriately set the weights to ± 1. Naturally the weights may be set to some initial values and then trained by the LMS algorithm. This is not necessarily the most desirable strategy since minimising the *mean-square* error is not directly related to modelling the Bayesian estimation formula. Bayesian estimation provides the best possible bit-error rate performance but this is not necessarily the best *mean-square* error solution. "Training" the weights also slows the convergence of the equaliser training.

4.3.5 Complexity of the RBFNE Algorithms

The computational complexity of the RBFNE algorithms will be analysed in terms of the number of basis functions required (which is exactly the number of channel states or cluster centres), denoted *j*. Note that this differs from the filter order *m* used to evaluate the complexity of the Linear Transversal Equaliser. The number of basis functions required is dependent on the channel order and the sequence alphabet (i.e. the number of channel states).

Two approaches are discussed here, the first of which is the Stochastic Gradient algorithm described by Cha *et al* [17]. The second is the supervised clustering method described by Mulgrew *et al* [18].

(a) The Stochastic Gradient Algorithm

The stochastic Gradient algorithm can be summarised as follows:

$$
\begin{aligned}
w_{j,n+1} &= w_{j,n} + \mu_w e_n \phi_j(y_n) \\
\sigma_{j,n+1} &= \sigma_{j,n} + \mu_\sigma \phi_j(y_n) w_{j,n} e_n \left\{ \frac{\left\| y_n - c_{j,n} \right\|^2}{\sigma^3_{j,n}} \right\} \\
c_{j,n+1} &= c_{j,n} + \mu_c \phi_j(y_n) \left\{ \frac{w_{j,n} e_n (y_n - c_{j,n})}{\sigma^2_{j,n}} \right\}
\end{aligned}
\tag{4.7}
$$

Where n is the *nth* iteration, j is the *jth* unit and w, σ and c refer to the weights, variance and centres respectively. $\phi_j(y)$ refers to the output of the *jth* basis function in response to the input y, e is the error computed as in equation 4.2. The μ's are the respective learning rates.

Using this method there are three sets of parameters to be trained. If there are j basis functions then there are $3j$ parameters to be trained .

Each iteration in updating the weights requires the quantity $\mu_w x e$ to be computed once. A further j multiplication's and j additions are required. The non-linear function $\phi(y)$ needs to be calculated j times for each iteration of the algorithm. Updating the variance requires j additions, $3j$ multiplication's (since it is assumed that $\mu_\sigma x e$ is computed once) and j evaluations of the Euclidean distance between the input and the jth centre. Updating the centres requires j additions, $3j$ multiplication's and j subtractions. Note determining σ^2 and σ^3 in the above algorithm have been ignored. The complexity of the algorithm is $O(j)$.

(b) Supervised Clustering Algorithm

The second method involves the training of only one set of parameters, namely the centres and a single parameter the variance. In this case the variance is global in that it is assumed that each basis function has the same variance. There are also no learning parameters involved.

If there are j states (centres), in the binary case log_2j bits of the training sequence will determine to which state the current input belongs. On average the algorithm will run through $1/2$ the possible states before the correct one is found, there are thus $1/2jlog_2j$ comparisons that need to be made. The update step requires only 2 additions,1 multiplication and a division. The updating of the variance requires 1 addition, 1 multiplication, 2 divisions and 1 determination of the Euclidean distance which can be derived from the comparison stage above. Determining the Euclidean distance of the input from the current centre is dependent on the filter order and not on the number of basis functions and will be ignored here. The complexity of this algorithm is also $O(j)$. It is easy to see that this algorithm requires far fewer computations than the Stochastic Gradient approach.

4.4 Training the Multi-layer Perceptron

The conventional method for training a MLP is the Back-propagation of Errors Algorithm (BP algorithm). This algorithm has been used successfully in many circumstances. The problem with this algorithm used in conjunction with the MLP is that is known to suffer from overfitting or underfitting of the data, in addition the algorithm is known to get stuck on local minima [20]. Recently it has been shown that pattern classification problems such as the well known 'interleaved spirals' benchmark problem [21] are better solved using a different method [22]. This method has been shown to be robust in its generalisation abilities [23]. This little-known approach will be explored here, it is particularly well suited to disciplining neural networks with a small number of inputs.

4.4.1 The *a priori* Knowledge Required to Train the MLP

As for the LTE and RBFNE the knowledge of the channel order is needed in order to ensure that enough inputs to the equaliser are provided. Should the number of inputs be less than the channel order, the MLP is still able to form non-linear boundaries which may successfully classify the input. The performance of the equaliser will degrade, however.

Too many inputs may require a longer training sequence to be provided so that the MLP is able to cope with the additional data it is required to learn.

The number of hidden neurons is a key factor in the implementation of a MLP equaliser. For each channel there is an optimum number of hidden units in terms of computational complexity versus performance. If performance is to be optimised this factor needs to be determined.

4.4.2 The Back-propagation Algorithm

In its most basic form the BP Algorithm is a stochastic (or estimated) gradient decent algorithm. The algorithm minimises the *mean-square error*. At each iteration the weights are adjusted so as to reduce the error. The amount by which the weights are adjusted is dependent on the step size, often known as the *learning rate*. If the *learning rate* is too small, convergence will be slow. If the *learning rate* is too large, the solution will jump wildly and never converge. There are hundreds of variations of the back-propagation algorithm. The algorithm is susceptible to being caught on local minima. In is often difficult to select an appropriate *learning rate*. Most variations make use of a momentum term; each new search direction is computed as a weighted sum of the current gradient and previous search direction, which avoids the gradient changing wildly. This effectively filters out rapid local fluctuations [24]. The variations of the algorithm are generally empirically derived along with the procedures required to initialise the weights [15].

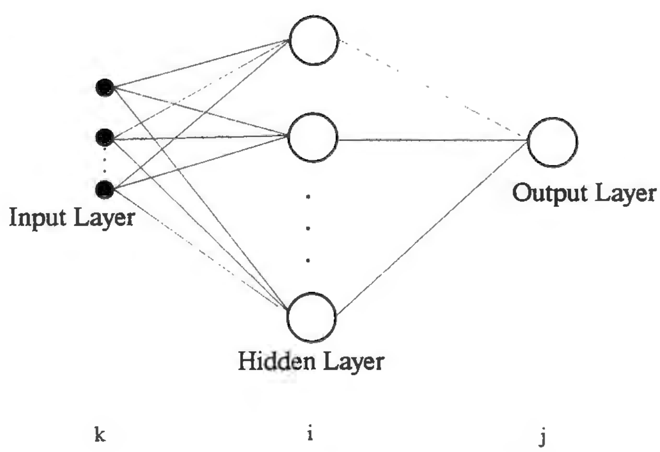


Figure 4.6 : A Three Layer MLP

Consider the neural network shown in figure 4.6. The first step in the BP algorithm is to present the network with an input. The weights between layers k and i and layer i and j are initially set in some way; usually to small random values. The first phase of the algorithm is to compute the output of each layer (forward pass). The second phase is known as the backward pass. This phase involves the following steps [25]:

- (1) The total error at the output is calculated as the difference between the actual and desired response.
- (2) The relative contribution of each of the inputs to the output node is computed to determine how the relative weights (between layers i and j) should be adjusted.
- (3) The hidden layer weights are then determined by considering their relative contribution to the output of the *ith* layer node to which they are connected.

For a detailed description see Rumelhart [25] *et al.*

4.4.3 Linear Regression Method (Random Assignment)

Recently Anderson *et al* has suggested an alternative method for training a three-layer MLP [22]. This method has been shown to be more robust than the BP algorithm; it does not suffer from the overfitting phenomena [26]. Its operation can be understood in terms of Covers theorem (see section 3.3).

This method involves randomly setting the input layer weights so that they uniformly span the input space. These weights are then frozen-they are never changed. The output of the network is simply a linear combination of the hidden layer outputs. A robust linear regression technique can then be used to calculate the output layer weights. One method is to use the LMS algorithm to minimise the *mean-square error*, thus the weights are updated on each iteration. Another method which does not require iteration is to directly solve for the matrix V of output layer weights using a pseudo-inverse technique.

Consider the network depicted in figure 4.6. The output of the network is the scalar z given by:

$$z = \sigma_2\left(\sum_{k=0}^{N-1} v_k \times \sigma_1\left(\sum_{j=0}^{m-1} W_{kj} \times x_j\right)\right) \tag{4.8}$$

Where σ_2 is a threshold element (note the BP algorithm requires that all basis functions including σ_2 be differentiable). σ_1 is the non-linear basis function in the hidden layer; usually the *tanh* or *sigmoid* function is used. The v_k's are the output layer weights which need to be trained. The W_{kj}'s are the weights which are fixed in a random manner, x_j is the input to one of m input nodes and N is the number of hidden layer neurons.

The input layer weights are fixed using pseudo-random or quasirandom numbers. Quasirandom numbers are preferred when the dimension of the input space is low (less than four). It has been shown by Anderson *et al* [27] that in low dimensional space quasirandom numbers span the input space more uniformly than pseudo-random numbers which tend to form clusters in the input space.

Quasirandom numbers, which are related to fibbonicci numbers and the "Golden Mean" [27], are generated from irrationals that are maximally difficult to approximate with rationals. For example in three-space, three irrationals are needed. Let these irrational be denoted as $(\alpha_0,\alpha_1,\alpha_2)$. α_0 is taken to be the positive real solution of $\alpha_0(\alpha_0+1)^3=1$, $\alpha_1=\alpha_0(\alpha_0+1)$ and $\alpha_2=\alpha_0(\alpha_0+1)^2$. Thus $(\alpha_0, \alpha_1, \alpha_2) = (0.380278,\ 0.524889,\ 0{,}724492)$, this is a point in three space which appears to be a suitable generalisation of the Golden Mean $(\sqrt{5}-1)/2=0.618034$.

The weights W_{kj} are then computed by taking the fractional part of the integer multiples of $(\alpha_0,\alpha_1,\alpha_2)$. For the weights to fall in the range [-1,1] they are computed as follows:

$$W_{kj} = 2 \times \mathrm{fract}\{K\alpha_j\} - 1 \qquad \text{For j} = 0,1,2 \text{ and } k = 0,1,\ldots,N-1 \tag{4.9}$$

Where *fract* indicates that only the fractional part is taken.

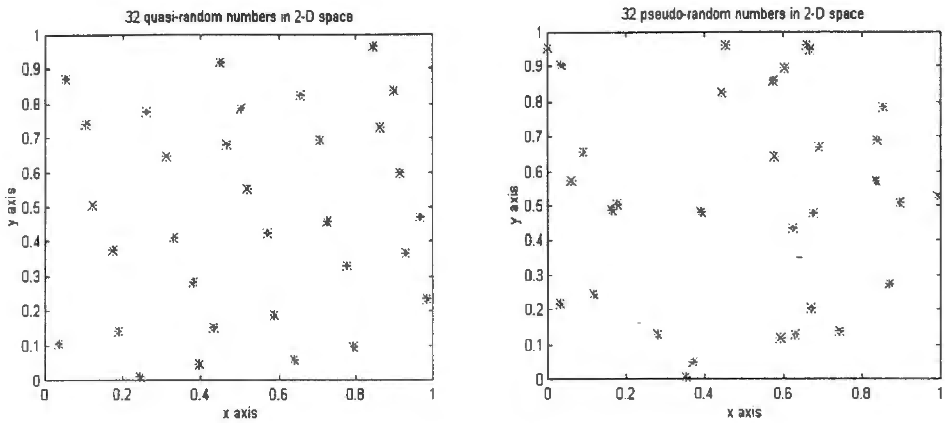


Figure 4.7 : Graphs comparing the distribution of Quasi-random and Pseudo-random numbers in two-space.

Once the input layer weights have been fixed, the next stage of the training process is to determine the set of output layer weights which will minimise the *mean-square error*. The network is presented with C training exemplars. In other words C input vectors and the associated desired response to each of the C input vectors is presented to the network. Let the c^{th} input vector be denoted by x_c and the associated desired response by d_c. The output of the hidden layer of the network is given by:

$$\mathbf{y}_c = \sigma_1(\mathbf{W}\mathbf{x}_c) \tag{4.10}$$

The scalar output of the network is given by:

$$z_c = \mathbf{v}\mathbf{y}_c \tag{4.11}$$

It is desired to solve for the set of weights v. This is accomplished in a least squares sense by using the pseudo-inverse of Y. Y is a $N \times C$ matrix whose c^{th} column is y_c. If Z is a row vector of length C whose c^{th} element is z_c, then the *following* equation holds:

$$\mathbf{vY} = \mathbf{Z} \tag{4.12}$$

The matrix Y is not square invertable. In practice a solution is found that can be shown to minimise the *mean-square error* in the predicted values of Y [16] .

$$v = \mathbf{ZY}^T(\mathbf{YY}^T)^{-1} = \mathbf{ZY}^{\#} \tag{4.13}$$

The matrix $\boldsymbol{Y}^{\#}$ is known as the pseudo-inverse of $\boldsymbol{Y}$. The problem still arises of how to calculate $(\boldsymbol{YY}^T)^{-1}$ in the above equation. Direct inversion is computationally inefficient and insufficiently robust to cope with the ill-conditioning that arises, a more economical approach is to use LU decomposition, an even better approach is to use the more robust QR decomposition. The most robust approach is to use a method known as singular value decomposition to calculate the pseudo-inverse $\boldsymbol{Y}^{\#}$ [16].

4.4.4 Complexity of the MLP's Training Algorithms

(a) The Back-propagation Algorithm

The Back-Propagation Algorithm consists of two phases. In the forward pass the weights are fixed and the output of each neuron is computed. For a fully connected MLP with N neurons in the hidden layer an m inputs, this phase requires $m \times N$ multiplication's and N evaluations of the non-linear activation function. In order to calculate the final output a further N multiplication's and additions are required. The backward pass requires N updates of the outer weights and $N \times m$ updates of the inner weights for each iteration. Note the physical umber of operations has been neglected here for simplicity . The MLPs that have been suggested for channel equalisation are generally comprised of more than one hidden layer [6] or have a very large number of neurons in the hidden layer. The computational requirements of a system employing the BP algorithm is likely to be considerably more than that of the RBFNE described in section 4.3, since the number of operations is $O(mN)$. In general a MLP will require more hidden units that a RBFNE for comparable performance **in this application**. In other applications such as function approximation, the number of hidden units required in a RBFN may be considerably more than for a MLP.

(b) Linear Regression Method (Random Assignment)

The complexity of the random assignment technique is difficult to analyse. Generally algorithms used to solve sets of linear equations require $O(n^3)$ operations, where n is the number of unknowns [28] (note this is a rough generalisation). The advantage with this method is that it is not necessary for the computations to be conducted in real-time, which makes the computational complexity less of an issue. Once the training sequence has been transmitted an idle time can be allocated to allow the receiver to complete the required calculations. This is not necessarily the most desirable solution, nevertheless it is plausible. It should be noted that the computational complexity of this approach is dependent on the method used to solve for the pseudo-inverse (i.e. SVD, LU or QR decomposition). For example singular-value decomposition requires approximately $2Cn^2+4n^3$ operations [29]. Where C is the number of training samples and n is the number of unknowns (the number of hidden layer neurons). In order to compare this with the iterated algorithms used to train the other equaliser structures this quantity may be divided by C. As long as the number of unknowns n is very small compared to C the number of operations is $O(n^2)$.

4.5 A Discussion of the Training Algorithms

The LTE has been trained with the LMS algorithm; other algorithms such as the recursive least squares (RLS) algorithm may be used. The LMS algorithm is simple and relatively efficient in this context. It lends itself to direct comparison to the other gradient descent algorithms investigated.

Two basic approaches where investigated to train the RBFNE. The one approach involves gradient descent the other approach is a modelling approach. The Gradient descent approach is computationally more intensive than the modelling approach. The convergence of the algorithm is dependent on three learning parameters that need to be determined heuristically. Heuristic parameters are undesirable as they are problem dependent. It was found that in general the stochastic gradient algorithm exhibited slower convergence than the supervised modelling approach.

The modelling approach may be subdivided into two categories, namely the unsupervised and supervised approaches. The unsupervised approach does not require a training sequence, however convergence is not guaranteed. There are however circumstances where this type of blind equalisation may be desirable. The supervised approach is computationally efficient and converge is guaranteed.

The MLP was trained with the linear regression technique (random assignment), this technique has been found to be more robust than both the back-propagation algorithm or an iterated gradient descent (such as LMS on the output weights). Using the random assignment approach the MLP employs only one hidden layer and has been found in general to require a small number of basis functions (in this application).

Chapter 5

Experimental Results and Analysis

5.1 Introduction

This section deals with simulations conducted on the Adaptive Linear Transversal Equaliser (LTE), the Radial Basis Function Network Equaliser (RBFNE) and the Multi-layer Perceptron Equaliser (MLP). Based on the theory described in chapters two and three, simulations where conducted to verify or disprove the advantages of neural networks applied to channel equalisation. Three different structures are simulated. The first is the standard adaptive linear transversal filter which still enjoys popularity as an equaliser due to its simple structure, despite the fact that this structure can only form linear decision boundaries. The second structure is the Radial Basis Function Network equaliser which employs localised basis functions and has a direct relationship to Bayesian Estimation. The third structure is the Multi-layer Perceptron which employs non-localised basis functions and has an indirect relationship to Bayesian estimation. A number of different channel types are used to highlight certain advantages/disadvantages of the three structures.

The equaliser structures are compared based on their bit error rate performance. An analysis of these results is also included in this chapter.

5.2 A discussion of the simulation

All the simulations where conducted at baseband. Figure 5.1 illustrates the structure of the simulations. The simulations where written in Borland Turbo C++ or Matlab. A pseudo-random number generator was used to generate a NRZ (non-return to zero) baseband signal. For the C simulations the pseudo-random number generator is aperiodic for 10^8 bits

[30]. The same pseudo-random number generator is used in conjunction with the Box-Muller formula to generate the additive Gaussian Noise [30].

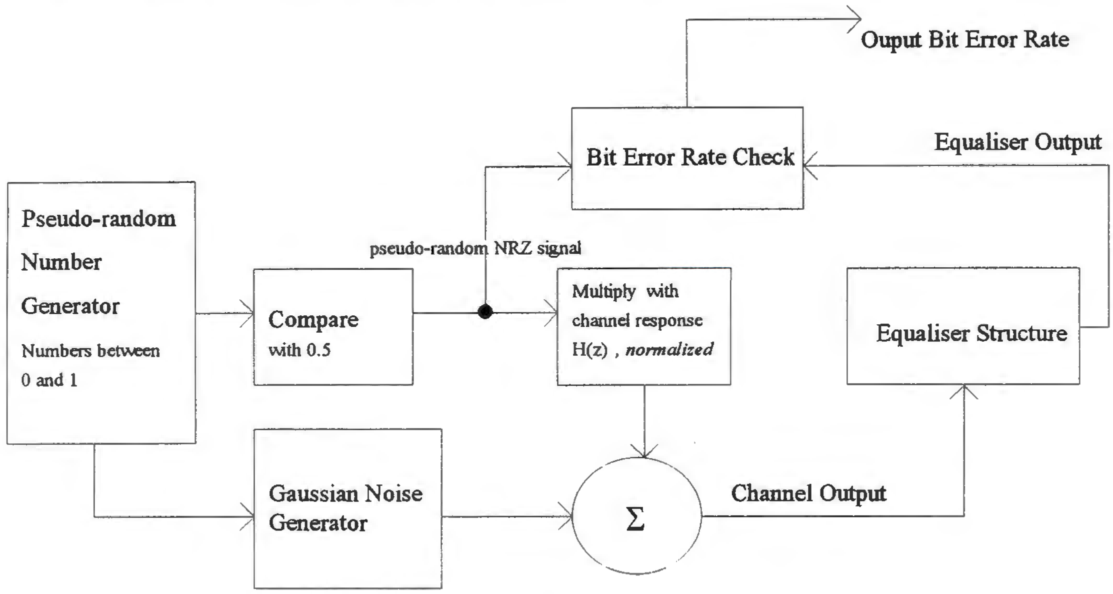


Figure 5.1 : Diagram Showing Structure of Computer Simulation.

The pseudo-random bit stream is multiplied by the channel transfer function *H(z)* to generate a channel output. The resultant channel output is then added to the generated Gaussian noise. The respective signal to noise ratios were calculated as follows [13] :

$$\begin{aligned} &\mathrm{E}[s(k)] = 0 \\ &\mathrm{E}[s(k_1)s(k_2)] = \sigma_s^2 \delta(k_1 - k_2) \end{aligned} \tag{4.1}$$

Where E[] denotes the expectation operator and $s(k)$ is the transmitted signal at time k and $\delta(k)$ is the dirac delta function. The expected **value** of an equiprobable sequence selected from a finite alphabet of $\{+1,-1\}$ is zero. The expected value of the sequence squared is just the variance σ^2_s which is equal to one in this case.

The additive white Gaussian noise $n(k)$ has zero mean and therefore the expected value is zero.

$$\begin{aligned} &\mathrm{E}[n(k)] = 0 \\ &\mathrm{E}[n(k_1)n(k_2)] = \sigma_n^2\delta(k_1 - k_2) \end{aligned} \tag{4.2}$$

Where σ^2_n is the noise variance. Note $s(k)$ and $n(k)$ are assumed to be uncorrelated.

If the transfer function of the channel is normalised (i.e. is divided by the sum of channel coefficients) then the signal-to-noise ratio is defined by the following [13] :

$$SNR = \frac{\sigma_s^2}{\sigma_n^2} \tag{4.3}$$

Where σ^2 denotes variance and the subscripts s and n denote symbol and noise respectively. A complete source code listing for the respective simulations is given in appendices I, II and III.

5.3 Simulations on Standard Linear Channel

The first simulation conducted was on a standard linear channel $H(z)=1+0.5z^{-1}$. The Linear Transversal Equaliser is known to perform well on such a channel as the input states are linearly separable and hence a linear decision boundary is sufficient to separate the two classes of inputs.

5.3.1 The LTE Compared to the RBFNE

A second order LTE was simulated and trained by the LMS Algorithm with $\eta=0.01$ for 1000 samples. This was compared to the Radial-basis function network equaliser trained by the supervised clustering method with the weights set. This network comprised of two inputs with 8 Gaussian basis functions, trained over 1000 samples .

Figure 5.2 shows that the RBFNE outperforms the LTE even for this simple channel type. The performance below about 10db of the two structures is almost identical. This is due to the fact that at these low signal to noise ratio's the effects of noise dominate the effects of the ISI. At higher signal to noise rations (SNR) the RBFNE outperforms the LTE by as much as 1.8db at a SNR of 20db.

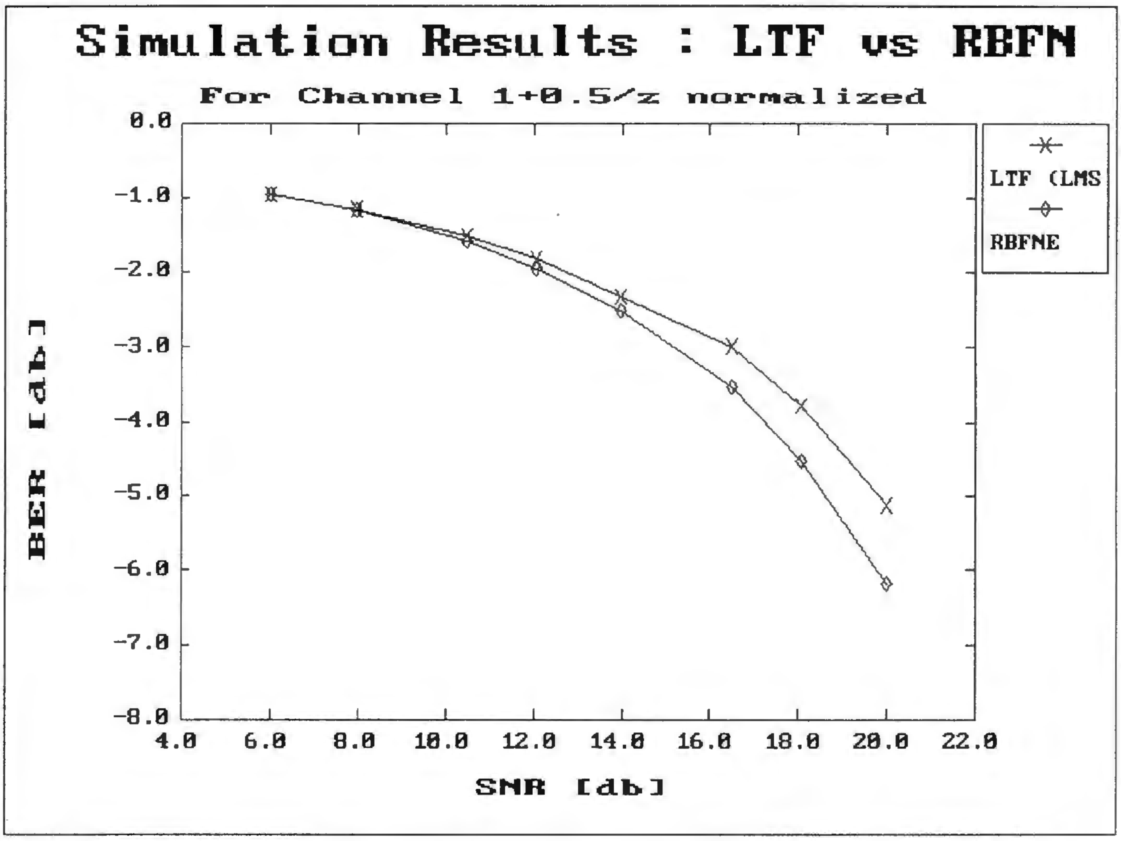


Figure 5.2 Simulation for channel $H(z)=1+0.5z^{-1}$, comparing the LTE to the RBFNE

5.3.2 The RBFNE Compared to the MLP

The next simulation conducted was to compare the performance of the RBFNE to the MLP. Since the RBFNE very closely approximates the Bayesian Decision boundary which is the optimal decision boundary it should be expected that the MLP can only approach the performance of the RBFNE as the number of neurons becomes larger (the mapping is to a high dimensional space). Figure 5.3 shows the comparison of a MLP with 8,16 and 32 units to the RBFNE. Once again the RBFNE was trained with an *supervised clustering* approach with its weights set, the MLP was trained using the linear regression technique (random assignment). The hidden weights where set using quasirandom numbers. A three input MLP was used in all cases with one input permanently set to *+1*, which acts as a bias term. Both structures were trained over 1000 samples. At low SNRs the MLP is able to accurately learn the optimal decision boundary. This is due to the fact that the noisy input spans the input space to a greater extent than if the noise were less, allowing the MLP to more accurately "learn" the optimal decision boundaries. As the noise becomes less, i.e. the SNR increases, the input occupies less of the volume of the input space and the MLP is unable to learn the optimal decision boundary. Figure 5.4 illustrates this phenomenon, figure 5.3 is a graph of the respective simulation results..

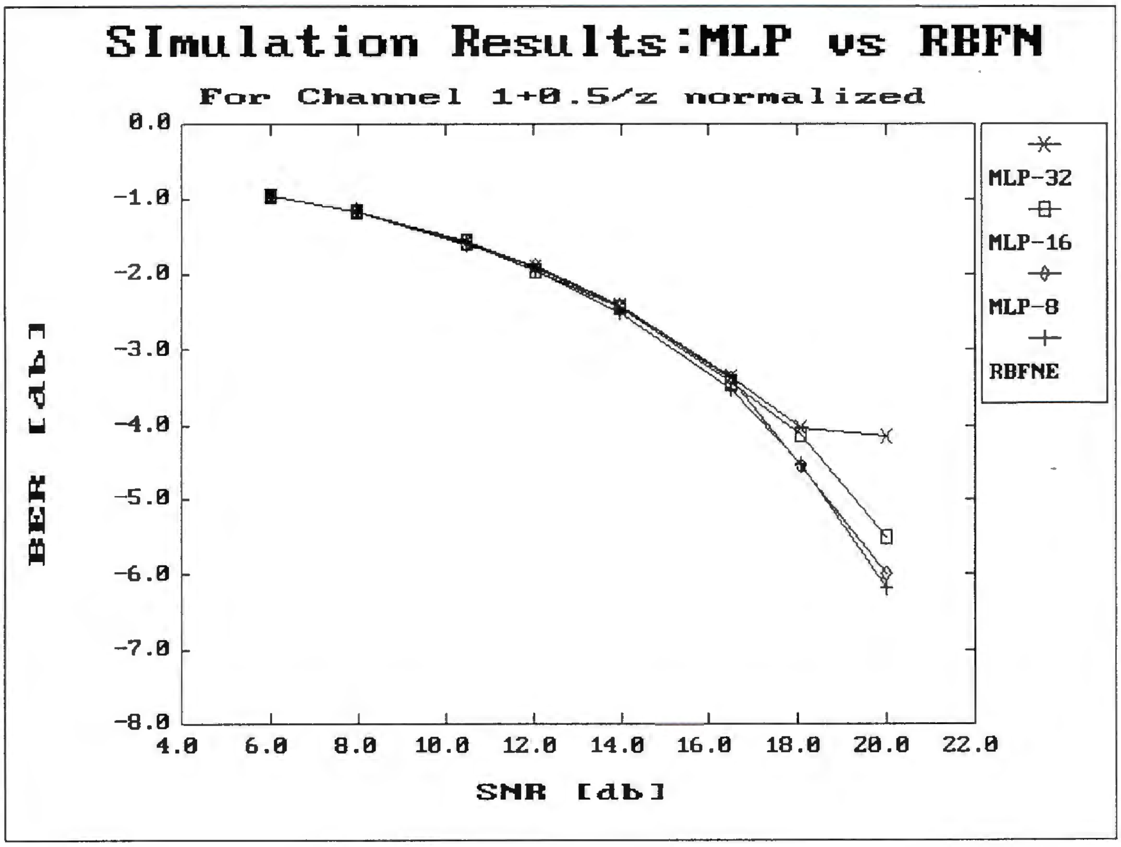


Figure 5.3 : Simulation comparing a MLP with 8,16 and 32 hidden units to a RBFNE for channel $H(z)=1+0.5z^{-1}$

The MLP-8 performs better that the MLP-16 and MLP-32, the performance of the MLP-8 closely approaches that of the RBFNE with exactly the same number of units. Figure 5.3

shows that above a SNR of about 18db the bit error rate of the MLP-32 equaliser begins to level off, indicating the inability of the MLP-32 to learn in such low noise environments.

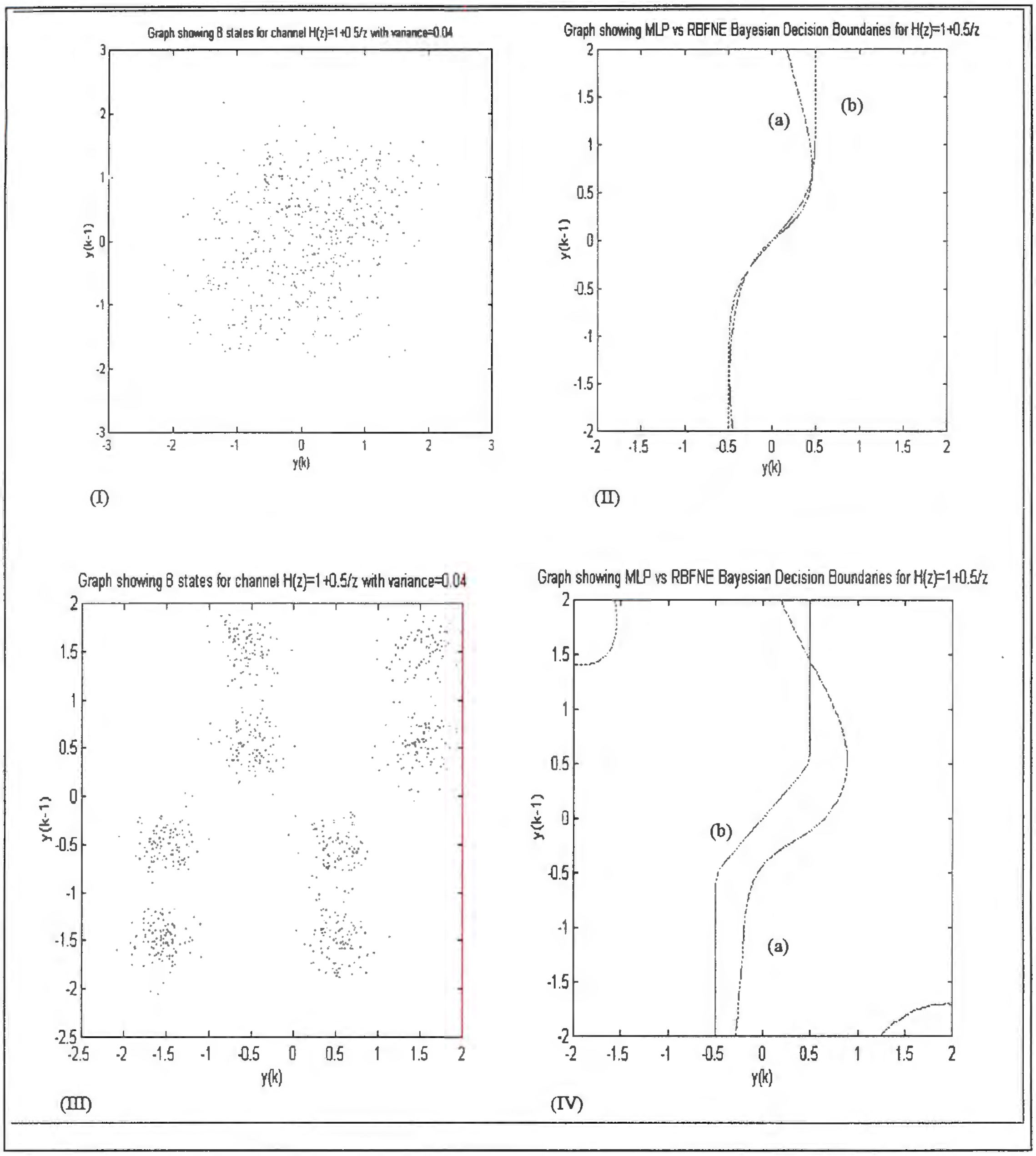


Figure 5.4 : An illustration of the MLP's ability to learn the Bayesian Decision Boundary in the presence of noise. *(I) A training sequence of 1000 samples with variance 0.25 (II) (a) the boundary created by the MLP (b) the optimal boundary. (III) A training sequence of 1000 samples with variance 0.0625. (IV) (a) the boundary created by the MLP (b) the optimal boundary.*

In figure 5.4 (IV) notice the extraneous decision boundaries in the upper left and lower right corners of the graph, created by the MLP due to the lack of training data in that region.

The following simulations address the issue of how the number of hidden layer nodes effect the performance of the MLP. Increasing the number of hidden units does not necessarily improve the performance of the MLP equaliser, this is due to the fact that the higher dimension MLP has more weights to train and hence more freedom in the generation of a decision boundary. The additional power is not necessarily beneficial since the MLP has more difficulty in "learning" due to its higher dimensionality.

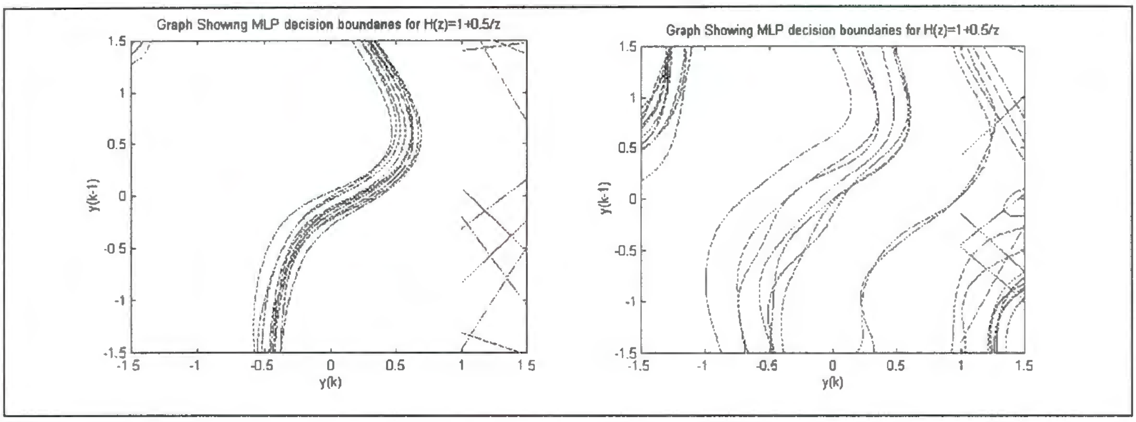


Figure 5.5 : Graph showing the variation of decision boundaries for 10 simulations for a MLP-16 (left) and a MLP-32.(right).

This is best understood as follows: Consider a set of inputs that is not linearly separable in the input space. Cover's theorem of separability suggests that should these inputs be non-linearly mapped to a high enough dimension they will be linearly separable. If the inputs are mapped to the lowest possible dimension for which they are linearly separable a hyperplane may be passed through this space which will separate the classes of inputs. There may be a family of hyperplanes which will separate the classes of inputs (i.e. a number of possible solutions). If the input is mapped to an even higher dimension there is a greater probability that there will be an even larger family of solutions. The solution found by the MLP in this high dimensional space will not necessarily be a better solution than that found in the lower dimensional space, since the hyperplane is likely to have more degrees of freedom in terms of where it is located in this high dimensional space.

This phenomena is clearly seen by considering figure 5.5 which shows the difference in the spread of solutions between a MLP-16 and an MLP-32 for ten simulations. Clearly the family of solutions generated by the MLP-16 are more constrained in their location than the MLP-32. In order to limit the family of solutions in this high dimensional space additional training information would be required. The problem is compounded at high SNR's (low noise) where the training information only occupies certain highly concentrated areas of the input space.

5.4 Simulations on Non-minimum phase channel

Non-minimum phase channels are channels in which in the *z-domain* the zero's of the transfer function $H(z)$ are located outside of the unit circle [4]. LTE's are unable to equalise these channels as the input to the equaliser is not linearly separable. Figure 5.6 illustrates the received states and their associated set membership. Notice how it is not possible to pass a line though the graph that will separate the two classes. It is also worthwhile noting that this example is merely of academic interest as by introducing a delay τ, the zeros of the transfer function will shift into the unit circle and the problem becomes linearly separable. This example does however serve to illustrate the ability of neural networks to create convoluted non-linear decision boundaries.

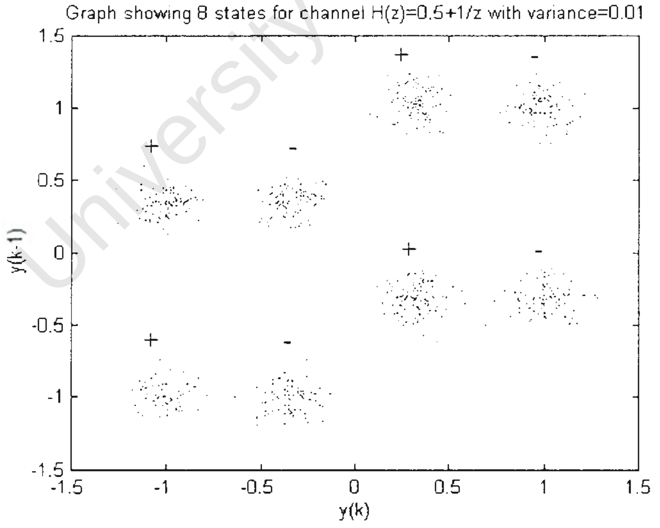


Figure 5.6 : Signal States for Channel $H(z)=0.5+z^{-1}$

Figure 5.6 shows the signal states associated with the non-minimum phase channel $H(z)=0.5+z^{-1}$. The '+' symbol indicates that the state in question is associated with a $+1$ transmitted symbol and a '-' indicates that the state is associated with a -1 transmitted symbol. Figure 5.7 shows the optimal decision boundaries for noise variances of 0.25 and 0.0625.

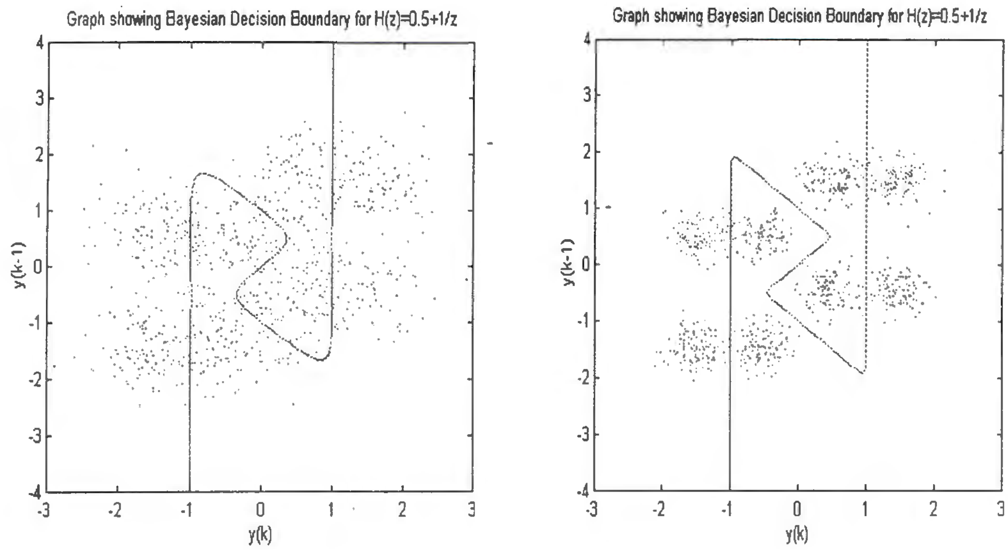


Figure 5.7 : Graphs showing the optimal decision boundaries for channel $H(z)=0.5+z^{-1}$ for two different noise variances.

Due to the convoluted nature of the decision boundaries it is reasonable to expect that compared to the linear channel discussed in section 5.3, the MLP requires additional processing power. Since the channel states are not linearly separable in the input space, it is expected that a higher dimensional mapping than that of the linear channel would be required. The RBFNE on the other hand is exactly the same structure as that used for the linear channel in section 5.3. The inferior performance of the RBFNE for the non-minimum phase channel as compared to the linear channel is only as a result of the reduced distance between differing channel states (see figure 5.6), which increases the probability of error. In other words the centres of the RBFNE converge correctly as for the linear channel, the estimate of the noise variance given by equation 4.6 also closely approximates the expected result as for the linear channel.

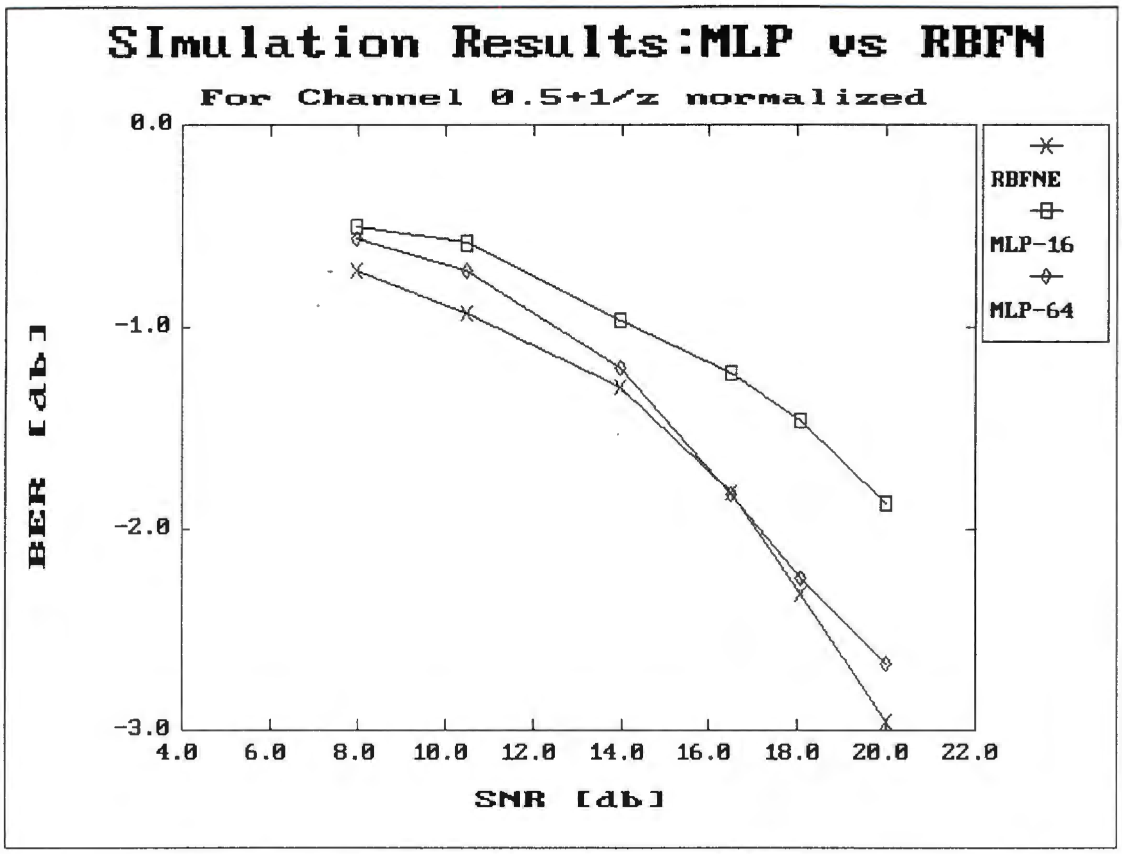


Figure 5.8 : Simulation results for non-minimum phase channel $H(z)=0.5+z^{-1}$.

Figure 5.8 illustrates the fact that the MLP-16 (a multi-layer Perceptron with 16 hidden neurons) performs over 3db worse (in terms of SNR) than the RBFNE. The performance of a MLP with 64 hidden neurons is considerably better than one with 16 hidden neurons. This suggests that the MLP-16 does not have enough processing power to effectively equalise this channel. The additional processing power provided by the MLP-64 improves the performance. The RBFNE offers a substantial performance gain over the MLP is this situation particularly at low SNR's.

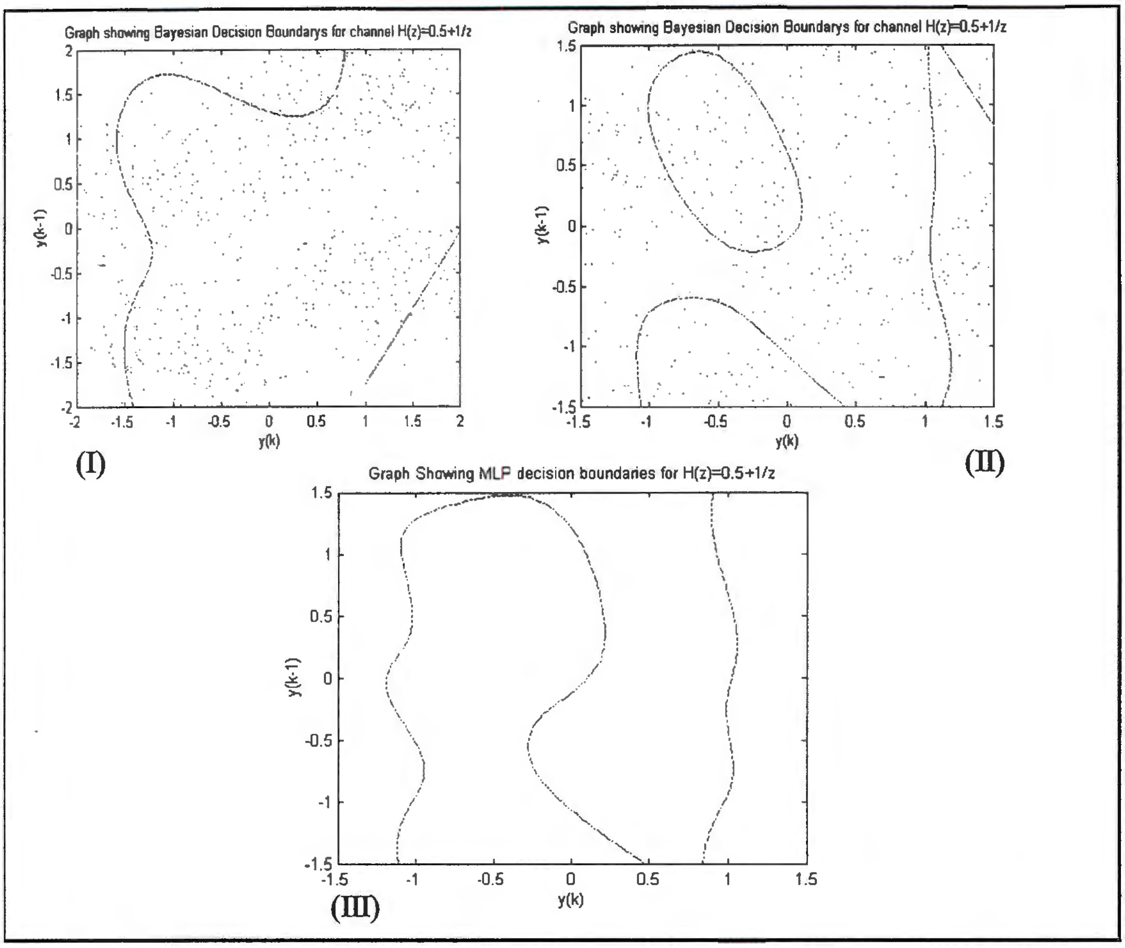


Figure 5.9: Graph showing the ability of a MLP-16 (I) to learn the optimal decision boundary as compared to a MLP-32 (II) and a MLP-64 (III). The noise variance is set at 0.25.

Figure 5.9 illustrates the fact that the MLP-16 has difficulty in creating a decision boundary, whereas the MLP-32 is able to create a better boundary (which is naturally sub-optimal). The MLP-64 creates the closest to optimal boundary of the three.

5.5 Simulations for non-linear Channel

The following simulations where conducted using the non-linear channel model depicted in figure 5.10 [12].

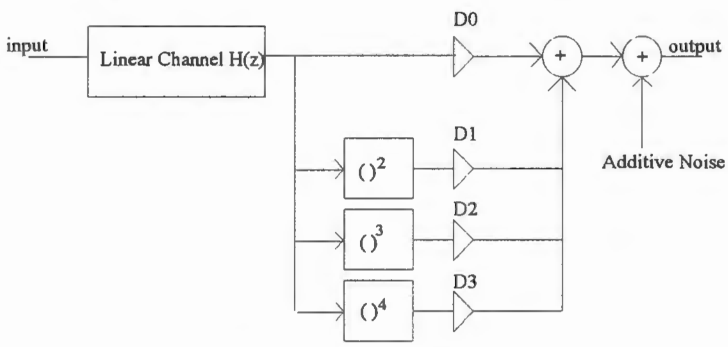
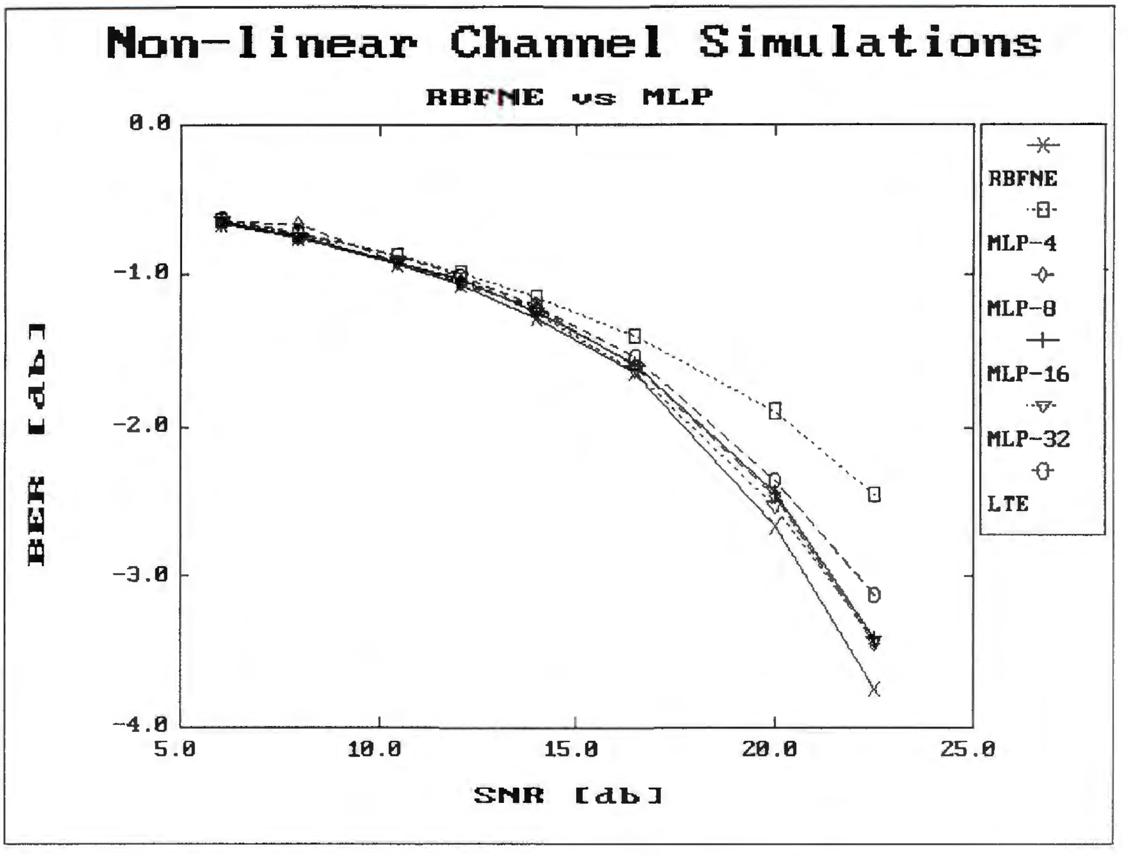


Figure 5.10 : Diagram Showing Non-linear channel model.

5.5.1 Mild Non-linearity

The linear channel component was chosen to be $H(z)=1+0.5z^{-1}$. The coefficients where set to D0=0.5, D1=0.2, D2=0.2 and D3=0.1. The Non-linear components where made small compared to the linear component.

The simulation results are displayed in figure 5.11. Again the RBFNE was trained with the supervised clustering algorithm, eight Gaussian kernels where employed and the training sequence was 1000 symbols. The MLP was trained with the linear regression (random assignment) technique over 1000 samples. Four different MLP's where simulated. The difference being the number of units in the hidden layer. A 4,8,16 and 32 neuron MLP where simulated (denoted MLP-4,MLP-8 etc.). A second order LTE was trained over 1000 samples with the LMS algorithm (η=0.01).



5.11 : Simulation Results of Mild Non-linearity.

As expected the RBFNE exhibited the best performance, with a performance gain of approximately 2.4db at an SNR of 20db (in terms of SNR) over the LTE. The LTE was able to equalise this channel type, due to the fact that the linear component dominates the non-linear component. The MLP-4 was unable to learn the decision boundary effectively due to the insufficient number of neurons (processing power) in the hidden layer . The MLP-8, MLP-16 and MLP-32 produced significantly better results than the MLP-4. The RBFNE exhibits an approximate gain of between 1-2db for a SNR of between 20-22.5db. There was no significant difference in performance between the MLP-8, MLP-16 and MLP-32. Thus with as few as 8 neurons in the hidden layer the MLP is able to closely approach the performance of the RBFNE.

5.5.2 Strong Non-linearity

The linear channel component remained the same for the next simulation. The contribution of the non-linear components was increased. The following coefficients where used (see figure 5.10) : *D0=0.2*, *D1=0.5*, *D2=0.1* and *D3=0.2*. The simulation results are presented in figure 5.12. All the equaliser structures where trained as in section 5.5.1.

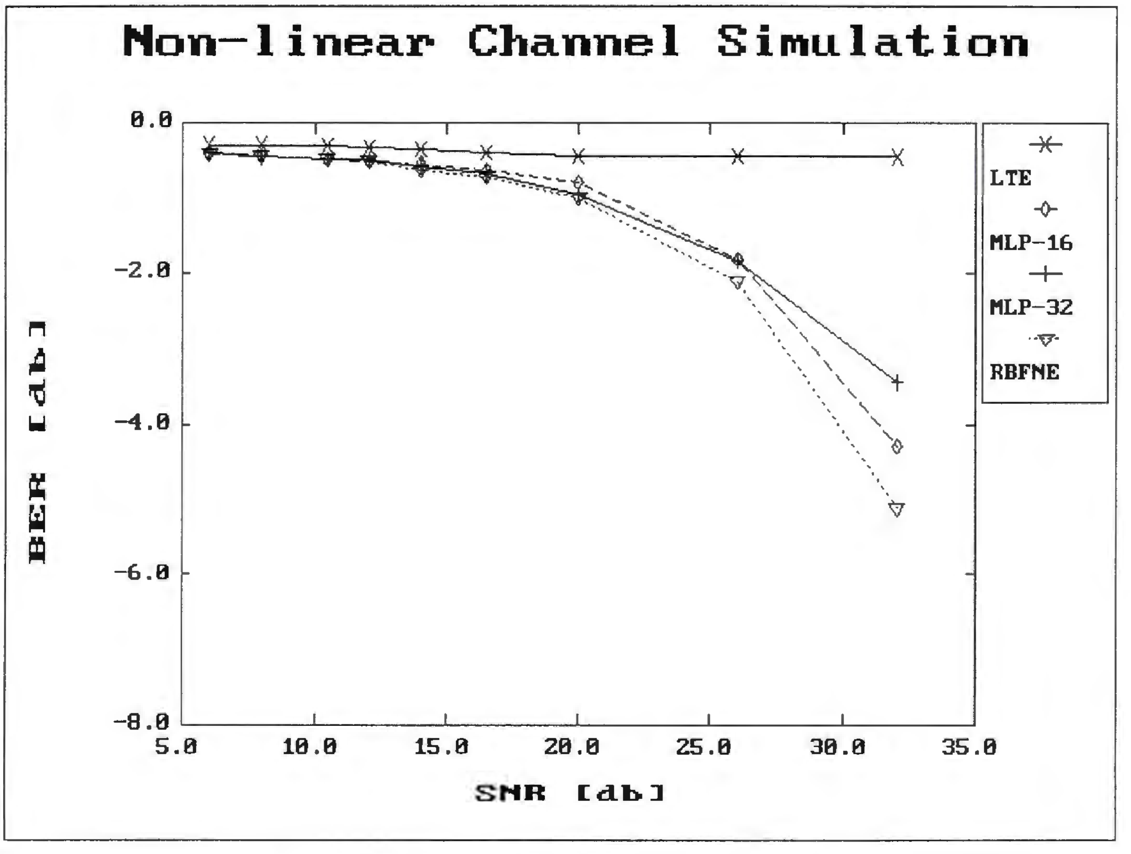


Figure 5.12 : Simulations of Strong Non-linearity

With the non-linear components dominating the linear components, the LTE is unable to successfully equalise the channel. Both the MLP-16 and MLP-32 offer a significant performance gain over the LTE in this situation. The MLP-16 has the advantage that in low noise environments it performs better than the MLP-32. This is due to the fact that the MLP-16 has less weights to train and hence does not require as much "training data". The RBFNE once again outperforms the MLP in this instance, by approximately 1.5db at a SNR of 25db.

Chapter Six

Conclusions and Summary

6.1 Summary

Communication cannel equalisation is one of the many problems that neural networks have been applied to. The application of neural networks to channel equalisation has arisen from extensive work conducted using neural networks as pattern classifiers. Since equalisation can be reformulated as a pattern classification problem, the classification powers of a neural network can be used for channel equalisation.

Bayesian Estimation has been shown to produce optimal decision boundaries for symbol-by-symbol channel equalisation. The Radial Basis Function Network has the structure to model the Bayesian Estimation Formula. For an equiprobable data stream all that is needed is an algorithm that can train the RBFN's centres to coincide with the channel states and is able to estimate the noise variance. If the distribution of the noise is Gaussian with zero mean and Gaussian kernels are used in the RBFN, the RBFNE is able to accurately model the Bayesian Estimation Formula. Doing so enables the RBFNE to create near optimal decision boundaries. An algorithm exists that has been shown by simulation to produce good results. The RBFN trained with this supervised clustering algorithm has been shown to outperform the standard LTE and the MLP. The algorithm is both computationally efficient and robust in that it exhibits good convergence properties.

Whereas the RBFNE gains its power from the fact that it is constrained to model Gaussian conditional density functions, the MLP has no such restrictions. The MLP simply performs a non-linear mapping to a high dimensional space. A decision boundary is created by passing a hyperplane through this space (the projection of which into the input space forms a non-linear boundary). The MLP thus has more processing power than the RBFNE. It is due to this additional processing power that the MLP can only indirectly arrive at the Bayesian solution, by learning. It is for this reason that the performance of the MLP is worse than that of the RBFNE for the cases simulated. In low noise environment the MLP is often unable to learn the decision boundaries as there is not enough data present near the boundaries to aid the learning process.

A new method for training the MLP was investigated, namely the (random assignment) linear regression technique. This method does not suffer from being caught on local minima as the Back-propagation algorithm does. There is still however the problem of how many neurons are required in the hidden layer. With too few neurons the MLP lacks the required processing power, too many neurons effect the ability of the MLP to learn; due to the fact that there are too many parameters to train. The optimal number of neurons required is problem dependent.

6.2 Conclusions

Since it is possible to train the RBFNE so that it models the Bayesian Estimation Formula very accurately, it is reasonable to conclude that if the additive noise is Gaussian noise, no other feed-forward structure will perform better than the Radial Basis Function Network Equaliser for symbol-by-symbol channel equalisation (neglecting structures that employ decision feedback and assuming that the correct number of basis functions are provided). Irrespective of the number of hidden units and the length of the training sequence a MLP can only approximate the Bayesian decision boundary and hence it's performance is always inferior to the RBFNE.

The (random assignment) linear regression technique for training the MLP produces good results. As long as enough training information is provided (dependent on the number of hidden neurons), the MLP is able to equalise a communication channel. In all cases the MLP provides an improvement over the LTE. The number of hidden units required is relatively small.

6.3 Further Research

There are a large number of clustering algorithms in existence. There is a possibility that one or more of these algorithms may be used in conjunction with an algorithm to estimate the channel order (as in [18]) to blindly train a RBFNE. The possibility exists that the blindly trained RBFNE could approach the performance of the RBFNE trained with a training sequence, particularly if the clustering algorithm is able to accurately cluster the input.

Bibliography

[1] B.Mulgrew , *Applying Radial Basis Functions* , IEEE Signal Processing Magazine , March 1996.

[2] B.Widrow and D.Stearns , *Adaptive Signal Processing* ,Prentice-Hall , 1985.

[3] S.U.H.Qureshi, *Adaptive Equalization*, Proc. IEEE, Vol.73, No.9, pp 1349-1387, 1985.

[4] A.V.Openheim and R.W.Schafer , *Discrete-Time Signal Processing* , Prentice-Hall , 1989.

[5] J.J.Shynk, *Adaptive IIR Filtering*, IEEE ASSP Magazine, April 1989, Vol.6, No.2.

[6] G.J.Gibson and C.F.N.Cowan , *The Application of Non-linear Structures to the Reconstruction of Binary Signals* , IEEE Trans. on Signal Processing , Vol. 39 No 8 , August 1991 , pp. 1877-1884.

[7] S.Theodondis, C.F.N.Cowan, C.P. Callender and C.M.S.See , *Schemes for the Equalisation of Channels with Non-linear Impairments* , IEE Proc. on Communication , Vol. 143 No3 , June 1995.

[8] K.Abend, B.D.Fritchman, *Statistical detection for communication channels with intersymbol interference*, Proc. IEEE, Vol.58, No.5, pp 779-785, 1970.

[9] R.A.Johnson , *Miller and Freud's Probability and Statistics for Engineers* 5th edition , Prentice-Hall , 1994.

[10] Y.Park, G.Lee, *Applications of Neural Networks in High-speed Communication Networks*, IEEE Communications Magazine, October 1995.

[11] S.Haykin, *Neural Networks Expand SP's Horizons*, IEEE Signal Processing Magazine, May 1996.

[12] G.Kechriotis, E.Zervas, E.S.Manolakos, *Using Recurrent Neural Networks for Adaptive Communication Channel Equalization*, IEEE Trans. on Neural Networks, Vol.5, No. 2, March 1994.

[13] S.Chen, B.Mulgrew, S.Mclaughlin, *Adaptive Bayesian Equalizer with Decision Feedback,* IEEE Trans. on Signal Processing, Vol. 41, No. 9, September 1993, pp 2918-2927.

[14] M.S.Rafie and W.Newman , *A New Complex Neural-Based Decision-Feedback Equaliser: Application to Bandlimited Non-linear Satellite Links* , Int. Conf. on Signal Processing Applications and Technology , October 1994 , pp. 1251-1256.

[15] S.Haykin, *Neural Networks a comprehensive foundation*, Macmillan, 1994.

[16] S.Haykin , *Adaptive Filter Theory* 2nd Edition , Prentice-Hall , 1991.

[17] I.Cha and S.A.Kassam , *Channel Equalisation using Adaptive Complex Radial Basis Function Networks* , IEEE Journal on Selec. Areas in Communication , Vol. 13 No 1 , January 1995.

[18] S.Chen, B.Mulgrew, P.M.Grant, *A Clustering Technique for Digital Communications Channel Equalization Using Radial Basis Function Networks*, IEEE Trans. on Neural Networks, Vol.4, No.4, July 1993, pp 570-579.

[19] C.Chinrungrueng and C.H.Sequin, *Optimal Adaptive K-Means Algorithm with Dynamic Adjustment of Learning Rate*, IEEE Trans. on Neural Networks, Vol. 6, No.1, January 1995, pp 157-169.

[20] J.Hertz, A.Krough, R.G.Palmer, *Introduction to the theory of Neural Computation*, Addison-Wesley,1991.

[21] K.J.Cios, N.Liu, *A Machine Learning Method for Generation of a Neural Network Architecture*, IEEE Trans. on Neural Networks, Vol.3, No.2, March 1992, pp 280-291.

[22] P.G.Anderson, R.S.Gaborski, M.Ge, S.Raghavendra, M.Lung, *Using Quasirandom Numbers in Neural Networks*, Available FTP : ftp.cs.rit.edu Directory: \pub\pga , File quasi_NN.ps.z.

[23] C.L.P.Chen ,*A Rapid Supervised Learning Neural Network for Function Interpolation and Approximation*, IEEE Trans. on Neural Networks, Vol. 7, No. 5, September 1996, pp 1220-1229.

[24] T.Masters, *Practical Neural Network Recipes in C++*, Academic Press, 1993.

[25] D.E.Rumelhart, G.E.Hinton, R.J.Williams, *Learning representation by back-propagating errors*, Letters to Nature, Vol. 323, October 1986.

[26] Prof. J.R.Greene, *Personal Communications*, December 1996.

[27] P.G.Anderson, *Multidimensional Golden Means*, Available FTP : ftp.cs.rit.edu Directory: \pub\pga , File mgm.ps.z , March 1993.

[28] L.I.Kronsjo, *Algorithms: Their Complexity and Efficiency*, John Wiley & Sons, 1979.

[29] G.H.Goloub, C.F. van Loan, *Matrix Computations*, North Oxford Academic Press, 1983.

[30] W.H.Press, S.A.Teukolsky, W.T.Vetterling, B.P.Flannery, *Numerical Recipes in C : The Art of Scientific Computing*, Second Edition, Cambridge University Press, 1992.

Appendix I

Source Code of LTE Simulation

```
/************************************************************************/
/**** Equalizer Simulation                                          ****/
/**** By A.B Olshewsky                                              ****/
/**** Written for the Digital Communication Research group  in Bolrland c ****/
/**** September 1996                                                ****/
/************************************************************************/
/************************************************************************/
/** These simulations will be conducted on the following channels      */
/**  (a) A standerd minimum phase channel                              */
/**  (b) A Non-minimum phase channel                                   */
/**  (c) A Non-linear channel                                          */
/**                                                                    */
/** All simulations will be conducted for varing SNR's                 */
/**                                                                    */
/************************************************************************/

/*** ================---------- INCLUDE FILES ----------================ ***/
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <time.h>
#include <graphics.h>
#include <conio.h>
#include <math.h>
#include <dos.h>

/*** ================---------- GLOBAL DEFINITIONS ----------================ ***/
#define FILTORD 2                       /* Defines Filter Order m */
#define INPUTS 2
#define SIMSIZE 1000000                 /*Defines the length of the simulation*/

#define CENTERS 8                       /* The number of basis functions*/
#define MEU 0.01                        /* Learning rate for LMS */
#define TRUE 1
#define FALSE 0
#define TRAIN_LEN 1000                  /*length of the training sequence*/

/******** Declarations for Random number generator ************/
#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define NTAB 32
#define NDIV (1+(IM-1)/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)
```

```
/*** ===========------- GLOBAL VARIABLE DECLARATIONS --------============= ***/
long int COUNT;
long int E_COUNT;
float IN_CHANNEL[2];
float CHAN_COEFF[2];
float OUT_CHANNEL;
float FILTER[FILTORD];
float WEIGHTS[FILTORD];
long SEED;

/****************************************************************************/
/****                                                                    ****/
/****           Random Number Generator                                  ****/
/****                                                                    ****/
/****************************************************************************/
float ran1(long *idum)
{
        int j;
        long k;
        static long iy=0;
        static long iv[NTAB];
        float temp;

        if (*idum <= 0 || !iy) {
                if (-(*idum) < 1) *idum=1;
                else *idum = -(*idum);
                for (j=NTAB+7;j>=0;j--) {
                        k=(*idum)/IQ;
                        *idum=IA*(*idum-k*IQ)-IR*k;
                        if (*idum < 0) *idum += IM;
                        if (j < NTAB) iv[j] = *idum;
                }
                iy=iv[0];
        }
        k=(*idum)/IQ;
        *idum=IA*(*idum-k*IQ)-IR*k;
        if (*idum < 0) *idum += IM;
        j=iy/NDIV;
        iy=iv[j];
        iv[j] = *idum;
        if ((temp=AM*iy) > RNMX) return RNMX;
        else return temp;
}
/* (C) Copr. 1986-92 Numerical Recipes Software                  */
/****************************************************************************/
/****                                                                    ****/
/****           Gaussian Noise Generator                                 ****/
/****         uses random number generator defined above                 ****/
/****************************************************************************/
float gasdev(long *idum)
{
        float ran1(long *idum);
        static int iset=0;
        static float gset;
        float fac,rsq,v1,v2;
```

```
        if (iset == 0) {
                do {
                        v1=2.0*ran1(idum)-1.0;
                        v2=2.0*ran1(idum)-1.0;
                        rsq=v1*v1+v2*v2;
                } while (rsq >= 1.0 || rsq == 0.0);
                fac=sqrt(-2.0*log(rsq)/rsq);
                gset=v1*fac;
                iset=1;
                return v2*fac;
        } else {
                iset=0;
                return gset;
        }
}
/* (C) Copr. 1986-92 Numerical Recipes Software                    */

/**************************************************************/
/***                                                        ***/
/***                Bit Error Rate Checker                  ***/
/***                                                        ***/
/**************************************************************/
void BitERR()
{
long int k;
float thresh,SUM,TMP;

for(k=0;k<FILTORD;k++)
        {
          TMP=(float)(FILTER[k]*WEIGHTS[k]);       /* Compute output */
          SUM=(float)(SUM+TMP);
        }

/* Put output through a threshold ellememt */

if(SUM>=0)
thresh=1.0;
else
thresh=-1.0;

if(thresh != IN_CHANNEL[0])         /* Note IN_CHANNEL[0] is the current transmitted bit */
E_COUNT++;                          /* Counts the number of errors made */
clrscr();

k=COUNT-TRAIN_LEN;
printf("\n\n\n The ERROR count is << %ld >> in [ %ld ] bits",E_COUNT,COUNT);
}

/*************************************************************************/
/****                                                               ****/
/****               LEAST MEAN SQUARES ALGORITHM                    ****/
/****                                                               ****/
/*************************************************************************/
void LMS()
{
int i;
float TMP,OUTPUT,SUM,epsilon;
float NEW[FILTORD],threshold;
```

```
SUM=0;
/* Calculate the OUTPUT Yk */
for(i=0;i<FILTORD;i++)
        {
          TMP=(float)(FILTER[i]*WEIGHTS[i]);
          SUM=(float)(SUM+TMP);
        }
            /* Error = desired response-output*/
            epsilon=(float)(IN_CHANNEL[0]-SUM);

/* Pass through threshold ellement */
if(SUM>=0)
threshold=1.0;
else
threshold=-1.0;

if(threshold != IN_CHANNEL[0])
{
sound(440);
delay(100);
nosound();
}

/* Calculate New weights */
for(i=0;i<FILTORD;i++)
{       NEW[i]=(float)(WEIGHTS[i]+(MEU*epsilon*FILTER[i]));
           }

for(i=0;i<FILTORD;i++)
{     WEIGHTS[i]=NEW[i];
                  }
printf("\nWEIGHTS : [ %1.3f : %1.3f ] : ERROR [ %1.3f ]",WEIGHTS[0],WEIGHTS[1],epsilon);
}


/************************************************************/
/***                                                     ***/
/***      Add Gaussian Noise to channel output           ***/
/***                                                     ***/
/************************************************************/
void ADD_NOISE(long *idum)
{
float std_dev;

std_dev=0.025;                  /* the standard deviation is just the noise variance squared */
                                /* this parameter is altered for differing noise variance */
OUT_CHANNEL=OUT_CHANNEL+std_dev*(gasdev(idum));
}
```

```
/**********************************************************/
/***                                                   ***/
/*** Move Coefficients into Filter                     ***/
/***                                                   ***/
/**********************************************************/
void LTF_FILTER()
{
int i;
float temp;

        for (i=FILTORD;i>1;i--)
        {
        FILTER[i-1]=FILTER[i-2];

        }
        FILTER[0]=OUT_CHANNEL;
}
/*******************************************************************/
/****                                                         ****/
/****       Convolve The Channel Reponse with the input       ****/
/****                                                         ****/
/*******************************************************************/
void CONV()
{
OUT_CHANNEL =CHAN_COEFF[0]*IN_CHANNEL[0]+CHAN_COEFF[1]*IN_CHANNEL[1];

/* NOTE FOR NON-LINEAR CHANNEL - COEFFICIENTS ARE INSERTED HERE */
}
/**********************************************************/
/***                                                   ***/
/*** Generates a bipolar-NRZ signal                    ***/
/***                                                   ***/
/**********************************************************/
void NRZ_GEN(long *idum)
{
int i,nrz;
float num;


num=ran1(idum);                        /* Generate a random number from 0-1 */
if (num<=0.5)
        nrz=-1;
else
        nrz=1;


IN_CHANNEL[1]=IN_CHANNEL[0];
IN_CHANNEL[0]=nrz;

}
```

```
/*****************************************************************/
/***                                                          ***/
/*** Initializes Channel to zero and set response             ***/
/***                                                          ***/
/*****************************************************************/
void INIT_CHANNEL()
{
IN_CHANNEL[0]=0;              /*Initially channel set to zero*/
IN_CHANNEL[1]=0;
OUT_CHANNEL=0;
CHAN_COEFF[0]=0.6666;          /* Mimic a channel 1+0.5z^-1 */
CHAN_COEFF[1]=0.3333;          /* Note the normalised channel coefficients */
                               /* Alter these coefficients to change channel */
}

/*****************************************************************/
/***                                                          ***/
/*** Initializes FILTER to zero                               ***/
/***                                                          ***/
/*****************************************************************/
void INIT_FILTER()
{
int i;
        for(i=0;i<FILTORD;i++)
        {
                FILTER[i]=0;
                                }
}
/*****************************************************************/
/***                                                          ***/
/*** Initializes FILTER Weights                               ***/
/***                                                          ***/
/*****************************************************************/
void INIT_WEIGHTS()
{
int i;
        for(i=0;i<FILTORD;i++)
        {
                WEIGHTS[i]=0;          /* Initialize weigths to 1 or 0 */
                                }
}


/*****************************************************************/
/***                                                          ***/
/***                MAIN                                      ***/
/***                                                          ***/
/*****************************************************************/
void main()
{
long int i;
SEED=-1.0;                      /* seed for random number generator , must be -ve integer */

nosound();
clrscr();
```

```
                                        /* Do initializations */
                                        INIT_CHANNEL();
                                        INIT_FILTER();
                                        INIT_WEIGHTS();
E_COUNT=0;
COUNT=0;
for (i=0;i<=SIMSIZE;i++)
        {
                NRZ_GEN(&SEED);             /* Generate Bipolar NRZ signal*/
                CONV();                     /* Convolve with response of filter */
                ADD_NOISE(&SEED);           /* Add Gaussian Noise */
                LTF_FILTER();               /* Pass channel output into  filter */
if(COUNT<=TRAIN_LEN)
                LMS();                      /* Train LTE using LMS */
else
                BitERR();

                COUNT++;
                                            }
}

/********* Undefine the defined constants *********/
#undef FILTORD
#undef INPUTS
#undef SIMSIZE
#undef CENTERS
#undef MEU
#undef IA
#undef IM
#undef AM
#undef IQ
#undef IR
#undef NTAB
#undef NDIV
#undef EPS
#undef RNMX
#undef TRUE
#undef FALSE
#undef TRAIN_LEN
```

Appendix II

Source Code of RBFNE Simulation

```
/**********************************************************************/
/****  Equalizer Simulation                                        ****/
/****  By A.B Olshewsky                                            ****/
/****  Written for the Digital Communication Research group        ****/
/****  November 1996                                               ****/
/**********************************************************************/

/**************************************************************************/
/****** A SIMULATION OF A RADIAL BASIS FUNCTION NETWORK EQUALISER ******/
/****** TRAINED WITH THE SUPERVISED CLUSTERING METHOD              ******/
/**************************************************************************/
/*** ================---------- INCLUDE FILES ----------================ ***/
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <time.h>
#include <graphics.h>
#include <conio.h>
#include <math.h>
#include <dos.h>


/*** ================---------- GLOBAL DEFINITIONS ----------================ ***/
#define FILTORD 2
#define INPUTS 2
#define SIMSIZE 1000000

#define CENTERS 8
#define MEU 0.01
#define TRUE 1
#define FALSE 0
#define TRAIN_LEN 1000

/******** Declarations for Random number generator ************/
#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define NTAB 32
#define NDIV (1+(IM-1)/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)
```

```
/*** ================------- GLOBAL VARIABLE DECLARATIONS --------================ ***/
long int COUNT;
long int E_COUNT;
float IN_CHANNEL[2];
float CHAN_COEFF[2];
float OUT_CHANNEL;
float FILTER[FILTORD];
float WEIGHTS[FILTORD];
float PREV;
float vvar;
long SEED;

struct RAD{
	float inputs[INPUTS];	/* The number of inputs to basis function */
	float output;		/* The output */
	float weight;		/* The weight at the output */
	float center[FILTORD];	/* Centers are located at a location defined by this array*/
	float var;		/* The variance of each basis function */
	int clust_elmnts;	/* The number of ellements associated with a cluster
	}RBF;				/* This Location has the same dimension */
					/* As the Filter Order */
struct RAD RBFS[CENTERS];
/**************************************************************************/
/****                                                                ****/
/****            Random Number Generator                             ****/
/****                                                                ****/
/**************************************************************************/
float ran1(long *idum)
{
	int j;
	long k;
	static long iy=0;
	static long iv[NTAB];
	float temp;

	if (*idum <= 0 || !iy) {
		if (-(*idum) < 1) *idum=1;
		else *idum = -(*idum);
		for (j=NTAB+7;j>=0;j--) {
			k=(*idum)/IQ;
			*idum=IA*(*idum-k*IQ)-IR*k;
			if (*idum < 0) *idum += IM;
			if (j < NTAB) iv[j] = *idum;
		}
		iy=iv[0];
	}
	k=(*idum)/IQ;
	*idum=IA*(*idum-k*IQ)-IR*k;

	if (*idum < 0) *idum += IM;
	j=iy/NDIV;
	iy=iv[j];
	iv[j] = *idum;
	if ((temp=AM*iy) > RNMX) return RNMX;
	else return temp;
}
/* (C) Copr. 1986-92 Numerical Recipes Software 5.){2p491&#!0N17.. */
```

```
/****************************************************************************/
/****                                                                   ****/
/****              Gaussian Noise Generator                             ****/
/****        uses random number generator defined above                 ****/
/****************************************************************************/
float gasdev(long *idum)
{
        float ran1(long *idum);
        static int iset=0;
        static float gset;
        float fac,rsq,v1,v2;

        if (iset == 0) {
                do {
                        v1=2.0*ran1(idum)-1.0;
                        v2=2.0*ran1(idum)-1.0;
                        rsq=v1*v1+v2*v2;
                } while (rsq >= 1.0 || rsq == 0.0);
                fac=sqrt(-2.0*log(rsq)/rsq);
                gset=v1*fac;
                iset=1;
                return v2*fac;
        } else {
                iset=0;
                return gset;
        }
}
/* (C) Copr. 1986-92 Numerical Recipes Software 5.){2p491&#!0N17.. */

/*************************************************************/
/***                                                       ***/
/***              Bit Error Rate Checker                   ***/
/***                                                       ***/
/*************************************************************/
void BitERR()
{
int k,j;
float thresh,SUM,TMP,avr,ssdist;


SUM=0;
for(k=0;k<CENTERS;k++)                          /* Compute output of RBFNE */
        { ssdist=0;
          for(j=0;j<FILTORD;j++)
          {avr=FILTER[j]-RBFS[k].center[j];     /* Take the euclidean distance */
           ssdist=ssdist+(float)pow(avr,2);
          }
          RBFS[k].output=exp(-ssdist/(2*vvar));   /* calculate output of gaussian kernel */
          TMP=(float)(RBFS[k].output*RBFS[k].weight);
          SUM=(float)(SUM+TMP);
        }

/* Put output through a threshold ellememt */
if(SUM>=0)
thresh=1.0;
else
thresh=-1.0;
```

```
if(thresh != IN_CHANNEL[0])
E_COUNT++;                              /* Increase error count */
clrscr();

k=COUNT-TRAIN_LEN;
printf("\n\n\n The ERROR count is << %ld >> in [ %ld ] bits",E_COUNT,COUNT);
}
/*************************************************************************/
/****                                                                 ****/
/****              SUPERVISED LEARNING ALGORITHM                      ****/
/****                                                                 ****/
/*************************************************************************/

void sup_learn()
{
int mark,i;
float TMP,S_DIS,SUM;

mark=100;                       /* Set to some arbitary lare value */

/* decide as to what state the training sequence/input belongs to ? */
if(IN_CHANNEL[0]==-1 && IN_CHANNEL[1]==-1 && PREV==-1)
{
mark=0;
}
if(IN_CHANNEL[0]==-1 && IN_CHANNEL[1]==-1 && PREV==1)
{
mark=1;
}
if(IN_CHANNEL[0]==-1 && IN_CHANNEL[1]==1 && PREV==-1)
{
mark=2;
}
if(IN_CHANNEL[0]==-1 && IN_CHANNEL[1]==1 && PREV==1)
{
mark=3;
}
if(IN_CHANNEL[0]==1 && IN_CHANNEL[1]==-1 && PREV==-1)
{
mark=4;
}
if(IN_CHANNEL[0]==1 && IN_CHANNEL[1]==-1 && PREV==1)
{
mark=5;
}
if(IN_CHANNEL[0]==1 && IN_CHANNEL[1]==1 && PREV==-1)
{
mark=6;
}
if(IN_CHANNEL[0]==1 && IN_CHANNEL[1]==1 && PREV==1)
{
mark=7;
}
if(mark !=100)   /* if the training sequence belongs to a state, which it allways will */
{      SUM=0;
         for(i=0;i<FILTORD;i++)
         {
```

```
        /* Supervised learning of centers */
        RBFS[mark].center[i]=RBFS[mark].clust_elmnts*RBFS[mark].center[i]+FILTER[i];
        RBFS[mark].clust_elmnts++;
        RBFS[mark].center[i]=RBFS[mark].center[i]/RBFS[mark].clust_elmnts;
        RBFS[mark].clust_elmnts--;

        /* Variance estimation */
        TMP=FILTER[i]-RBFS[mark].center[i];
        S_DIS=(float)pow(TMP,2);
        SUM=SUM+S_DIS;
        }
        RBFS[mark].clust_elmnts++;
        vvar=((COUNT-1)*vvar+SUM/2)/COUNT;

}
}

/*********************************************************/
/***                                                  ***/
/***     Add Gaussian Noise to channel output         ***/
/***                                                  ***/
/*********************************************************/
void ADD_NOISE(long *idum)
{
float std_dev;

std_dev=0.025;
OUT_CHANNEL=OUT_CHANNEL+std_dev*(gasdev(idum));
}

/*********************************************************/
/***                                                  ***/
/*** Move Coefficients into Filter                    ***/
/***                                                  ***/
/*********************************************************/
void LTF_FILTER()
{
int i;
float temp;

        for (i=FILTORD;i>1;i--)
        {
        FILTER[i-1]=FILTER[i-2];

        }
        FILTER[0]=OUT_CHANNEL;
}
/*******************************************************************/
/****                                                         ****/
/****      Convolve The Channel Reponse with the input        ****/
/****                                                         ****/
/*******************************************************************/
void CONV()
{
OUT_CHANNEL=CHAN_COEFF[0]*IN_CHANNEL[0]+CHAN_COEFF[1]*IN_CHANNEL[1];
}
```

```
/**************************************************************/
/***                                                       ***/
/*** Generates a antipodal baseband  signal                ***/
/***                                                       ***/
/**************************************************************/
void APBB_GEN(long *idum)
{
int i,apbb;
float num;

num=ran1(idum);
if (num<=0.5)
	apbb=-1;
else
	apbb=1;

PREV=IN_CHANNEL[1];
IN_CHANNEL[1]=IN_CHANNEL[0];
IN_CHANNEL[0]=apbb;
}

/**************************************************************/
/***                                                       ***/
/*** Initializes Channel to zero and set response          ***/
/***                                                       ***/
/**************************************************************/
void INIT_CHANNEL()
{
IN_CHANNEL[0]=0;		/*Initially channel set to zero*/
IN_CHANNEL[1]=0;
OUT_CHANNEL=0;
CHAN_COEFF[0]=0.6666;		/* Mimic a channel 1+0.5z^-1 */
CHAN_COEFF[1]=0.3333;		/* Note these have been normalised here */
				/* Change these coefficeints to change channel type */

}

/**************************************************************/
/***                                                       ***/
/*** Initializes FILTER to zero                            ***/
/***                                                       ***/
/**************************************************************/
void INIT_FILTER()
{
int i;
	for(i=0;i<FILTORD;i++)
	{
		FILTER[i]=0;
				}
}
```

```
/*************************************************************/
/***                                                      ***/
/*** Initializes FILTER Weights                           ***/
/***                                                      ***/
/*************************************************************/
void INIT_WEIGHTS()
{

/*  SET THE WEIGHTS - since it is known that the data stream is equiprobable*/
RBFS[0].weight=-1;
RBFS[1].weight=-1;
RBFS[2].weight=-1;
RBFS[3].weight=-1;
RBFS[4].weight=1;
RBFS[5].weight=1;
RBFS[6].weight=1;
RBFS[7].weight=1;
}
/*************************************************************/
/***                                                      ***/
/***                   MAIN                               ***/
/***                                                      ***/
/*************************************************************/
void main()
{
long int i;
SEED=-3.0;

nosound();
clrscr();
                              /* Do initializations */
                              INIT_CHANNEL();
                              INIT_FILTER();
                              INIT_WEIGHTS();
E_COUNT=0;
COUNT=0;
for (i=0;i<=SIMSIZE;i++)
        {
                BPSK_GEN(&SEED);          /* Generate Bipolar BPSK signal*/
                CONV();                   /* Convolve with response of filter */
                ADD_NOISE(&SEED);         /* Add noise */
                LTF_FILTER();             /* Pass it into filter */
if(COUNT<=TRAIN_LEN)
                sup_learn();              /* Supervised learning algorithm */
else
                BitERR();

                COUNT++;
                                          }
}
/********* Undefine the defined constants *********/
#undef FILTORD
#undef INPUTS
#undef SIMSIZE
#undef CENTERS
#undef MEU
#undef IA
#undef IM
```

```
#undef AM
#undef IQ
#undef IR
#undef NTAB
#undef NDIV
#undef EPS
#undef RNMX
#undef TRUE
#undef FALSE
#undef TRAIN_LEN
```

Appendix III

Source Code of MLP Simulation

```
% Wriiten in Matlab ----
% NN equaliser using quasirandom assignment of weights and block regression
% Written by J.R.Greene and adapted for channel equalisation by A.B Olshewsky
% December 1996.

format compact
clg
hold off
clear X Y
std=0.025;                    % Specifies the noise power as a standard deviation
simsize=100000;               % Length of the simulation /1000
Train_len=1000;               % The Training length

%-------------------------------------------------------------------------
% Create a random polar NRZ signal, 1000 symbols long


R = rand(1,Train_len+1);          % 1001 uniformly dist random numbers in range 0-1
S = R > 0.5;                      % convert to nonpolar binary (0 and 1)
clear R;
S = 2*S -ones(1,Train_len+1);     % convert to polar sequence (-1 and 1)

% S is now a block of 1000 random polar NRZ symbols
%plot(S)
%-------------------------------------------------------------------------

% Model the channel as      s(t) = s(t) + 0.5s(t-1)

C =0.6666*S(2:Train_len+1) + 0.3333*S(1:Train_len);      % Note the normalised coefficients

% C=(0.2.*L)+(0.5.*L.*L)+(0.1.*L.*L.*L)+0.2.*(L.*L.*L.*L); only used for non-linear simulation

C = C + std*randn(1,Train_len);                           % add noise


% we now have 1000 symbols representing the channel output
plot(C)
%-------------------------------------------------------------------------


% Input to the NN (X) is the above signal and the above signal lagged by 1
% We will augment this matrix with a column of 1s to allow for a bias term

X(:,1) = ones(Train_len-1,1);
X(:,2) = C(2:Train_len)';                                 % note transpose ' to convert to column
X(:,3) = C(1:Train_len-1)';
```

```
% The three columns of X represent a block of 999 inputs to the NN
% We can plot the second column against the third to see a "constellation
% diagram" of the desired states

%subplot(1,3,1)
%axis([-5,5,-5,5])

%plot(X(:,2),X(:,3),'.');



%---------------------------------------------------------------------------------
% The output to the NN is the original data stream

Y =  S(3:Train_len+1)';                    % note transpose operator <'>

% Y is a column vector representing a block of 999 desired outputs

%---------------------------------------------------------------------------------
% We arbitrarily choose the number of hidden neurons (Nhidden)

Nhidden  =  64;

% Weights of the neurons are represented in a matrix Whidden (Hhidden x 10)
% These weights will be assigned quasirandomly:

RR=[.380278,.524889,.724492]'*(1:Nhidden);             % This creates a (3xNhidden)
Whidden =  RR - fix(RR);                               % matrix of quasirandom numbers

% Note:  we could have just used ordinary random numbers created by the
% Matlab  rand() function.  However, in a low-dimensional space random
% numbers are not particularly uniformly distributed, and it is possible
% to do much better using "quasirandom" numbers, based on a generalisation
% to higher dimensions of a process of randomised interval-division using
% fibbonacci series and the "golden mean" (suggested by Peter Anderson of
% Rochester Inst.of technology, whose papers are available via the Internet).

% We now have a  (3 x Nhidden) matrix of weights for the hidden neurons.

%---------------------------------------------------------------------------------

% Compute the outputs of the hidden neurons for the block of input data
% We will train the network on a subset of the data (the first Ntrain points)

% Ntrain = 320;

Xtrain  =  X(1:Train_len-1,:);
Ytrain  =  Y(1:Train_len-1);

Ohidden = tanh(Xtrain*Whidden);

% Estimate the output weights Wlinear required to construct an output vector
% which (in a least-squares sense) optimally matches the desired output Y
% For this, we do a robust linear regression with the Moore-Penrose matrix
% Pseudo-inversion (for which Matlab uses Singular Value Decomposition)
```

```
Wlinear = pinv(Ohidden) * Ytrain;

%--------------------------------------------------------------------------

% To see how well our NN performs, we will compare the thresholded output
% of the NN over the full data set, with the original data and determine the
% number of errors

% the Windows Matlab student edition is limited to a matrix size of 8190
% elements. To allow up to 16 hidden neurons we will temporarily
% limit the data size to < 8192/16 = 512 elements. The following 2 lines
% should be commented out when using the full version of Matlab.

% to ensure independence of the test data from the training data we will
% select the former from the values following the training sequence

% clear Xtrain;
% clear Ytrain;

% X = X(Ntrain+1:simsize-1,:);          % shortened test sequence for
% Y = Y(Ntrain+1:simsize-1);            % Student Windows Matlab

% NNoutput = sign(tanh(X*Whidden)*Wlinear);

% errors = sum(NNoutput ~= Y);

% errorrate = errors/(simsize-Ntrain)

% The following block of code creates a table of input values uniformly
% spannng the domain, & calculates a matrix of corresponding output values

terr=0;
for times=1:simsize

R = rand(1,Train_len+1);          % 1001 uniformly dist random numbrs in range 0-1
S = R > 0.5;          % convert to nonpolar binary (0 and 1)
clear R;
S = 2*S -ones(1,Train_len+1);     % convert to polar sequence (-1 and 1)

% S is now a block of 1000 random polar NRS symbols
%plot(S)
%--------------------------------------------------------------------------

% Model the channel as        s(t) = s(t) + 0.5s(t-1)

C =0.6666*S(2:Train_len+1) + 0.3333*S(1:Train_len);

% C=(0.2.*L)+(0.5.*L.*L)+(0.1.*L.*L.*L)+0.2.*(L.*L.*L.*L); Only for non-linear simulation
C = C + std*randn(1,Train_len);                              % add noise


% we now have 1000 symbols representing the channel output
%--------------------------------------------------------------------------
% note the simulation is coducted by iterating using matrices 1000 elements long

% Input to the NN (X) is the above signal and the above signal lagged by 1
% We will augment this matrix with a column of 1s to allow for a bias term
```

```
X(:,1) =  ones(Train_len-1,1);
X(:,2) =  C(2:Train_len)';           % note transpose ' to convert to column
X(:,3) =  C(1:Train_len-1)';

Y =  S(3:Train_len+1)';               % note transpose operator <'>

NNoutput =  sign(tanh(X*Whidden)*Wlinear);

errors = sum(NNoutput ~= Y);
terr=terr+errors;
A=times*1000;
disp([A,terr]);
end
```

Appendix IV

Generation of RBFNE Decision Boundary Plots

```
% Written in Matlab
% This program generates a plot of the decision boundaries created by the RBFNE
% The parameters wo-w7 may be set to the values derived by training the weights
% The parameters V0-V7 are the variance parameters
% Note the simulation may be set up to produce a text file with these parameters which may then be
% directly pasted into this source file

steps = -1.5 : .025 : 1.5 ;
[x,y] = meshgrid(steps,steps);
W0 = -1;
V0 = .25;
W1 = -1;
V1 = .25;
W2 = -1;
V2 = .25;
W3 = -1;
V3 = .25;
W4 = 1;
V4 = .25;
W5 = 1;
V5 = .25;
W6 = 1;
V6 = .25;
W7 = 1;
V7 = .25;
pos=W7*exp(-((x-1.5).^2)/(2*V7^2)).*exp(-((y-1.5).^2)/(2*V7^2))+W6*exp(-((x-
1.5).^2)/(2*V6^2)).*exp(-((y-0.5).^2)/(2*V6^2))+W5*exp(-((x-0.5).^2)/(2*V5^2)).*exp(-
((y+0.5).^2)/(2*V5^2))+W4*exp(-((x-0.5).^2)/(2*V4^2)).*exp(-((y+1.5).^2)/(2*V4^2));
neg=W0*exp(-((x+1.5).^2)/(2*V0^2)).*exp(-((y+1.5).^2)/(2*V0^2))+W1*exp(-
((x+1.5).^2)/(2*V1^2)).*exp(-((y+0.5).^2)/(2*V1^2))+W2*exp(-((x+0.5).^2)/(2*V2^2)).*exp(-((y-
0.5).^2)/(2*V2^2))+W3*exp(-((x+0.5).^2)/(2*V3^2)).*exp(-((y-1.5).^2)/(2*V3^2));
z=pos+neg;
contour(steps,steps,z,1);                                   % plot as a contour
title('Graph showing RBFNE decision boundary for H(z)=1+0.5/z');
xlabel('y(k)');
ylabel('y(k-1)');
pause
hold on;
```