

# COMMUNICATION CHANNEL EQUALIZATION USING MINIMAL RADIAL BASIS FUNCTION

## A DISSERTATION

*Submitted in partial fulfillment of the  
requirements for the award of the degree*

*of*

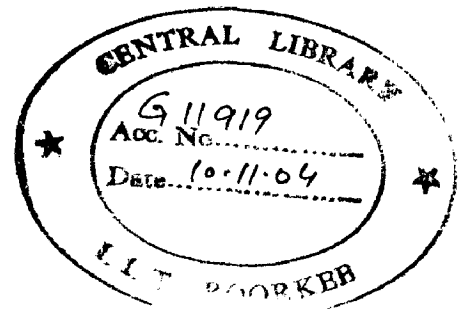
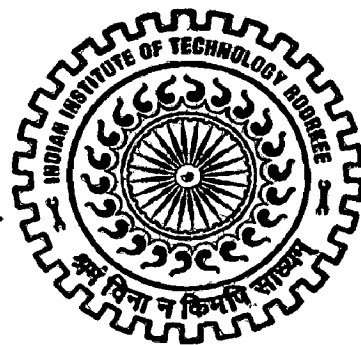
MASTER OF TECHNOLOGY

*in*

INFORMATION TECHNOLOGY

By .

**VISWANATH VEERA**



**IIT Roorkee - CDAC, NOIDA,  
c-56/1, "Anusandhan Bhawan"  
Sector 62, Noida-201 307**

**JUNE, 2004**

## CANDIDATE'S DECLARATION

---

---

I hereby declare that the work presented in this dissertation titled "Communication Channel Equalization Using Minimal Radial Basis Function", in partial fulfillment of the requirements for the award of the degree of **Master of Technology in Information Technology**, submitted in **IIT-Roorkee - CDAC campus, Noida**, is an authentic record of my own work carried out during the period from June 2003 to June 2004 under the guidance of **Dr.Moinuddin**, Professor, Jamia Millia Islamia (Central University), New Delhi.

I have not submitted the matter embodied in this dissertation for the award of any other degree or diploma.

Date:

Place: NOIDA

  
(VISWANATH VEERA)

---

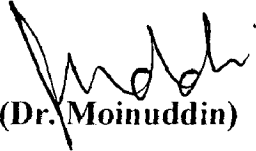
---

## CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

Date: 26-6-2004

Place: NOIDA

  
(Dr. Moinuddin)  
Professor

**Jamia Millia Islamia (Central University)**

**New Delhi.**

## ACKNOWLEDGEMENT

---

I express my sincere thanks and gratitude to my Guide **Dr. MOINUDDIN**, Professor, Jamia Millia Islamia (Central University), New Delhi, for his inspiring guidance and sustained interest throughout the progress of this dissertation.

I hereby take the privilege to express my deep sense of gratitude to **Prof. PREM VRAT**, Director, Indian Institute of Technology, Roorkee, and **Mr. R.K.VERMA**, Executive Director, CDAC, Noida for providing me with the valuable opportunity to carry out this work. I am very grateful to **Prof. A.K.AWASTI**, Programme Director, **Prof. R.P. AGARWAL**, course coordinator, M.Tech(IT), IIT, Roorkee and **Mr. V.N.SHUKLA**, course coordinator, M.Tech(IT), CDAC, NOIDA for providing the best of the facilities for the completion of this work and constant encouragement towards the goal.

I am thankful to **Mr. R.K.SINGH** and **Mr. MUNISH KUMAR**, project engineer, CDAC, Noida, for providing necessary infrastructure to complete the dissertation in time.

I owe special thanks to my friends, all of my classmates and other friends who have helped me formulate my ideas and have been a constant support.

I thank my **Parents** and my brother **Kalyan** for their moral support throughout my career.

(**VISWANATH VEERA**)

Enroll. No. 029029

## ABSTRACT

---

With the growth of Internet technologies, efficient high-speed data transmission techniques over communication channels have become an important topic for research. The channels used to send the data distort signals in both the amplitude and phase, causing what is known as Intersymbol Interference (ISI). Other factors like thermal noise, impulse noise, cross talk and the nature of the channel itself, cause further distortions to the received symbols. Signal processing techniques used at the receiver, to overcome these interferences, so as to restore the transmitted symbols and recover their information, are referred to as “equalization methods”.

The study of non-linear channel equalization in data communications using minimal radial basis function neural network structure, referred to as Minimal Resource Allocation Network (MRAN) are presented here . A parsimonious network is ensured by the MRAN algorithm, which uses online learning and has capability to grow and prune the RBF network’s hidden neurons. Compared to earlier methods, the proposed scheme does not have to reduce the channel order first, and fix the model parameters.

This learning algorithm for the network referred to as MRAN, not only allocates a new hidden neuron based on the novelty of observation but also prunes those hidden neuron units which have insignificant contribution to the outputs of the RBF network.

Here in the thesis, performance of MRAN on a number of applications has been tested. These applications consist of problems, which are both static and dynamic systems. The static problems include channel equalization. The problem for dynamic type consists of nonlinear dynamic system identification.

# CONTENTS

---

CANDIDATE'S DECLARATION

ACKNOWLEDGEMENT

ABSTRACT

CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

|  |             |
|--|-------------|
| <b>1. INTRODUCTION</b>                     | <b>1-3</b>  |
| 1.1 Motivation                             | 1           |
| 1.2 Problem Definition and its Description | 2           |
| 1.3 Organization of Dissertation Report    | 3           |
| <br>                                       |             |
| <b>2. LITERATURE SURVEY</b>                | <b>5-26</b> |
| 2.1 Introduction                           | 5           |
| 2.2 Artificial Neural Networks             | 5           |
| 2.3 Architecture of neural networks        | 7           |
| 2.3.1 Feedforward networks                 | 7           |
| 2.3.2 Feedback networks                    | 7           |
| 2.3.3 Network layers                       | 7           |
| 2.4 Learning process: An overview          | 8           |
| 2.4.1 Supervised Learning                  | 8           |
| 2.4.2 Unsupervised Learning                | 9           |
| 2.4.3 Learning Rates                       | 10          |
| 2.4.4 Learning Laws                        | 11          |
| 2.5 Multilayer Perceptrons                 | 13          |
| 2.6 Radial basis functions                 | 16          |
| 2.6.1 Network Topology                     | 16          |

|  |              |
|--|--------------|
| 2.6.2 Properties of RBF  | 18           |
| 2.6.3 Training Algorithms  | 19           |
| 2.7 Comparison of RBF and MLP networks   | 20           |
| 2.8 Channel Equalization   | 21           |
| 2.8.1 Equalization Problem   | 22           |
| 2.9 Summary  | 25           |
| <b>3. MINIMAL RESOURCE ALLOCATION NETWORKS</b>                                   | <b>27-32</b> |
| 3.1 Introduction   | 27           |
| 3.2 Resource Allocation Network via Extended Kalman<br>Filter Algorithm (RANEKF) | 27           |
| 3.3 Limitations of RANEKF  | 29           |
| 3.4 Pruning Strategy   | 29           |
| 3.5 Sliding Window RMS criterion for adding Hidden Neurons                       | 31           |
| 3.6 Summary  | 32           |
| <b>4. DESIGN AND IMPLEMENTATION</b>  | <b>33-37</b> |
| 4.1 Introduction   | 33           |
| 4.2 Design   | 33           |
| 4.3 Implementation   | 35           |
| 4.4 Summary  | 37           |
| <b>5. SIMULATION &amp; RESULTS</b>   | <b>39-51</b> |
| 5.1 Introduction   | 39           |
| 5.2 Results  | 39           |
| 5.3 Summary  | 51           |
| <b>6. CONCLUSION AND FUTURE WORK</b>   | <b>53</b>    |
| <b>REFERENCES</b>  | <b>55</b>    |

## LIST OF FIGURES

---

| <b>Figure No</b> | <b>Caption</b>  | <b>Page</b> |
|------------------|---|-------------|
| Figure 2.1       | Signal flow graph of Perceptron                               | 14          |
| Figure 2.2       | Discrete time model of data transmission system               | 22          |
| Figure 4.1       | Flow diagram of MRAN Algorithm                                | 34          |
| Figure 5.1       | Desired and Approximated function values for Model 1          | 40          |
| Figure 5.2       | Growth of Hidden Units as training progresses for Model 1     | 41          |
| Figure 5.3       | Error b/n Desired and Approximated as training progresses     | 41          |
| Figure 5.4       | Growth of Hidden Units as training progresses for Model 2     | 43          |
| Figure 5.5       | Equalizer Input and Output for Model 3                        | 45          |
| Figure 5.6       | Number of centres obtained as training progresses             | 45          |
| Figure 5.7       | Error Curves for RBF Networks for Model 3                     | 46          |
| Figure 5.8       | Equalizer Input and Output for Model 4                        | 47          |
| Figure 5.9       | Number of centres obtained as training progresses for Model 4 | 48          |

|             |   |    |
|-------------|---|----|
| Figure 5.10 | Error Curves for RBF Networks for Model 4                     | 48 |
| Figure 5.11 | Equalizer Input and Output for Model 5                        | 50 |
| Figure 5.12 | Number of centres obtained as training progresses for Model 5 | 50 |



## LIST OF ABBREVIATIONS

---

|        |  |
|--------|--|
| ANN    | Artificial Neural Networks                             |
| ART    | Adaptive Resonance Theory                              |
| BER    | Bit Error Rate   |
| MRAN   | Minimal Resource Allocation Network                    |
| CMRAN  | Complex Minimal Resource Allocation Network            |
| DFE    | Decision Feedback Equalizer                            |
| ISI    | Intersymbol Interference                               |
| LMS    | Least Mean Square                                      |
| MATLAB | MATrix LABoratory                                      |
| MLP    | Multilayer Perceptron                                  |
| RBF    | Radial Basis Function                                  |
| MSE    | Mean Square Error                                      |
| SNR    | Signal to Noise Ratio                                  |
| RANEKF | Resource allocation Network via Extended Kalman Filter |
| RAN    | Resource Allocation Network                            |
| RMSE   | Root Mean Square Error                                 |

## INTRODUCTION

---

### 1.1 Motivation

Radial Basis Function (RBF) neural networks have recently drawn much attention due to their good generalization ability and a simple network structure that avoids unnecessary and lengthy calculation as compared to the Multilayer Feedforward Networks (MFNs). For MFNs, their architecture tends to cause the problem of over generalization, and when the network is used for pattern classification problems, this problem becomes evident. The network may be trained to have high accuracy in classifying patterns from a set of unknown categories, but it will also classify any pattern which is out of these categories as one of them. This can cause severe problems in real-world applications. However, for an RBF classifier, especially with Gaussian function as its radial basis function, the network learns the pattern probability density instead of dividing up the pattern space as MFNs do. Therefore, when an out of category pattern is evaluated, such an RBF network is likely to classify it as an unknown category. In addition, the widely used Back Propagation (BP) training algorithm for MFNs is often too slow, particularly in case of large size problems. Since RBF network can establish its parameters for hidden neurons directly from the input data and train the network parameters by the way of linear optimization, it is generally much faster, compared to MFNs, to complete the training. Finally, due to their simple topological structure, RBF networks have the ability to reveal how learning proceeds in an explicit manner [12].

In the classical approach to RBF neural networks implementation, the number of hidden neurons is often fixed a priori based on some properties of input data. The weights connecting the hidden and output units are estimated by linear least square methods, e.g. Least Mean Square (LMS), Recursive Least Square (RLS) etc. The disadvantage of these approaches is that it usually results in too many hidden units [13]. Hence, these learning

methods for RBF networks are not suitable for sequential learning which may be needed for on-line system identification with the objective of adaptive control. Furthermore, a neural network with minimum size is less likely to learn noise in the training data and may thus generalize better to new data [14, 15].

Motivated by the above, a sequential learning algorithm for a minimal RBF neural network called Minimal Resource Allocation Network (MRAN) is presented here.

## **1.2 Problem Definition and its Description**

High speed communication channels are often impaired by channel intersymbol interference and additive noise. Equalization is nothing but restoring the transmitted symbols and recover their information by using some signal processing techniques at the receiver.

Estimation theory suggests that the best performance for symbol detection is obtained by using a maximum likelihood Sequence Equalizer (MLSE) for the entire symbol sequence, which involves batch processing scheme. But the computational complexity of MLSE is prohibitive and this has led to the popularity of equalizers that make decisions symbol by symbol as an alternative. The optimal solution for these symbol decision equalizers has been approached using the Bayesian decision theory. It can be seen from Bayesian solution that the optimal solution corresponds to non linear classification problem. As RBF networks are well suited for nonlinear classification problems, here I used RBF networks for channel equalization. Here, a sequential learning algorithm for implementing minimal Radial Basis Function neural networks has been developed. This learning algorithm for the network referred to as Minimum Resource Allocation Network (MRAN), not only allocates a new hidden neuron based on novelty of observation but also prunes those hidden neuron units which have insignificant contribution to the outputs of the RBF network. Therefore, when applied to time-varying dynamic systems, the architecture of MRAN will be adjusted automatically on-line to fit closely the dynamics of the observation data.

### **1.3 Organization of the Dissertation Report**

The organization of the Report can be briefly summarized as follows:

Chapter 2 presents a brief description about the Basic Architecture of Artificial Neural Networks (ANN), Multilayer Perceptrons, Radial Basis Function (RBF) and Channel Equalization. Chapter 3 presents a brief description about RANEKF (Resource Allocation Network via Extended Kalman Filter) Algorithm, its limitations and pruning strategy. Chapter 4 presents Design and Implementation part. The Implementation results are presented in Chapter 5. A summary of major conclusions from this thesis is presented in the Chapter 6. References are also included at the end of the report.

## LITERATURE SURVEY

---

### 2.1 Introduction

Artificial Neural Networks are being touted as the wave of the future in computing [16]. They are indeed self-learning mechanisms which don't require the traditional skills of a programmer. But unfortunately, misconceptions have arisen. Writers have hyped that these neuron-inspired processors can do almost anything. These exaggerations have created disappointments for some potential users who have tried, and failed, to solve their problems with neural networks. These application builders have often come to the conclusion that neural nets are complicated and confusing. Unfortunately, that confusion has come from the industry itself. An avalanche of articles has appeared touting a large assortment of different neural networks, all with unique claims and specific examples. Currently, only a few of these neuron-based structures, paradigms actually, are being used commercially. One particular structure, the feed forward, back-propagation network, is by far and away the most popular. Most of the other neural network structures represent models for "thinking" that are still being evolved in the laboratories. Yet, all of these networks are simply tools and as such the only real demand they make is that they require the network architect to learn how to use them.

### 2.2 Artificial Neural Networks

Artificial Neural Networks are relatively crude electronic models based on the neural structure of the brain [16]. The brain basically learns from experience. It is natural proof that some problems that are beyond the scope of current computers are indeed solvable by small energy efficient packages. This brain modeling also promises a less technical way to develop machine solutions. This new approach to computing also provides a more graceful degradation during system overload than its more traditional counterparts.

These biologically inspired methods of computing are thought to be the next major advancement in the computing industry. Even simple animal brains are capable of functions that are currently impossible for computers. Computers do rote things well, like keeping ledgers or performing complex math. But computers have trouble recognizing even simple patterns much less generalizing those patterns of the past into actions of the future.

Now, advances in biological research promise an initial understanding of the natural thinking mechanism. This research shows that brains store information as patterns. Some of these patterns are very complicated and allow us the ability to recognize individual faces from many different angles. This process of storing information as patterns, utilizing those patterns, and then solving problems encompasses a new field in computing. This field, as mentioned before, does not utilize traditional programming but involves the creation of massively parallel networks and the training of those networks to solve specific problems. This field also utilizes words very different from traditional computing, words like behave, react, self-organize, learn, generalize, and forget.

An artificial neural network (ANN) is an information-processing system that is based on generalizations of human cognition or neural biology. A neural network (NN) is characterized by its particular:

- Architecture; its pattern of connections between the neurons.
- Learning Algorithm; its method of determining the weights on the connections.
- Activation function; which determines its output.

A NN consists of a large number of processing elements, called neurons. Each neuron has an internal state, called its activation or activity level which is a function of the inputs it has received. Typically, a neuron sends its activation as a signal to several other neurons. A neuron can send only one signal at a time, although that signal may be broadcast to several other neurons.

## **2.3 Architecture of Neural Networks**

### **2.3.1 Feed-forward networks**

Feed-forward networks allow signals to travel one way only; from input to output. There is no feedback (loops) i.e. the output of any layer does not affect that same layer. Feed-forward networks tend to be straightforward networks that associate inputs with outputs. They are extensively used in pattern recognition [14,15]. This type of organization is also referred to as bottom-up or top-down.

### **2.3.2 Feedback networks**

Feedback networks can have signals traveling in both directions by introducing loops in the network [14]. Feedback networks are very powerful and can get extremely complicated. Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found. Feedback architectures are also referred to as interactive or recurrent, although the latter term is often used to denote feedback connections in single-layer organizations.

### **2.3.3 Network layers**

The commonest type of artificial neural network consists of three functional layers, namely Input, Output and Hidden layers. A layer of "input" units is connected to a layer of "hidden" units, which is connected to a layer of "output" units. The activity of the input units represents the raw information that is fed into the network. The activity of each hidden unit is determined by the activities of the input units and the weights on the connections between the input and the hidden units. The behavior of the output units depends on the activity of the hidden units and the weights between the hidden and output units. This simple type of network is interesting because the hidden units are free to construct their own representations of the input. The weights between the input and hidden units determine when each hidden unit is active, and so by modifying these weights, a hidden unit can choose what it represents. We also distinguish single-layer and

multi-layer architectures. The single-layer organization, in which all units are connected to one another, constitutes the most general case.

## **2.4 Learning Process: An Overview**

### **2.4.1 Supervised Learning**

The vast majority of artificial neural network solutions have been trained with supervision. In this mode, the actual output of a neural network is compared to the desired output. Weights, which are usually randomly set to begin with, are then adjusted by the network so that the next iteration, or cycle, will produce a closer match between the desired and the actual output. The learning method tries to minimize the current errors of all processing elements. This global error reduction is created over time by continuously modifying the input weights until acceptable network accuracy is reached.

With supervised learning [15], the artificial neural network must be trained before it becomes useful. Training consists of presenting input and output data to the network. This data is often referred to as the training set. That is, for each input set provided to the system, the corresponding desired output set is provided as well. In most applications, actual data must be used. This training phase can consume a lot of time. In prototype systems, with inadequate processing power, learning can take weeks. This training is considered complete when the neural network reaches an user defined performance level. This level signifies that the network has achieved the desired statistical accuracy as it produces the required outputs for a given sequence of inputs. When no further learning is necessary, the weights are typically frozen for the application. Some network types allow continual training, at a much slower rate, while in operation. This helps a network to adapt to gradually changing conditions.

Training sets need to be fairly large to contain all the needed information if the network is to learn the features and relationships that are important. Not only do the sets have to be large but the training sessions must include a wide variety of data. If the network is trained just one example at a time, all the weights set so meticulously for one fact could be drastically altered in learning the next fact. The previous facts could be forgotten in



learning something new. As a result, the system has to learn everything together, finding the best weight settings for the total set of facts. For example, in teaching a system to recognize pixel patterns for the ten digits, if there were twenty examples of each digit, all the examples of the digit seven should not be presented at the same time.

How the input and output data is represented, or encoded, is a major component to successfully instructing a network. Artificial networks only deal with numeric input data. Therefore, the raw data must often be converted from the external environment. Additionally, it is usually necessary to scale the data, or normalize it to the network's paradigm. This pre-processing of real-world stimuli, be they cameras or sensors, into machine readable format is already common for standard computers. Many conditioning techniques which directly apply to artificial neural network implementations are readily available. It is then up to the network designer to find the best data format and matching network architecture for a given application.

After a supervised network performs well on the training data, then it is important to see what it can do with data it has not seen before. If a system does not give reasonable outputs for this test set, the training period is not over. Indeed, this testing is critical to insure that the network has not simply memorized a given set of data but has learned the general patterns involved within an application.

#### **2.4.2 Unsupervised Learning**

Unsupervised learning is the great promise of the future. It shouts that computers could someday learn on their own in a true robotic sense. Currently, this learning method is limited to networks known as self-organizing maps. These kinds of networks are not in widespread use. They are basically an academic novelty. Yet, they have shown they can provide a solution in a few instances, proving that their promise is not groundless. They have been proven to be more effective than many algorithmic techniques for numerical aerodynamic flow calculations. They are also being used in the lab where they are split into a front-end network that recognizes short, phoneme-like fragments of speech which are then passed on to a back-end network. The second artificial network recognizes these strings of fragments as words.

This promising field of unsupervised learning is sometimes called self-supervised learning [15]. These networks use no external influences to adjust their weights. Instead, they internally monitor their performance. These networks look for regularities or trends in the input signals, and makes adaptations according to the function of the network. Even without being told whether it's right or wrong, the network still must have some information about how to organize itself. This information is built into the network topology and learning rules.

An unsupervised learning algorithm might emphasize cooperation among clusters of processing elements. In such a scheme, the clusters would work together. If some external input activated any node in the cluster, the cluster's activity as a whole could be increased. Likewise, if external input to nodes in the cluster was decreased, that could have an inhibitory effect on the entire cluster.

Competition between processing elements could also form a basis for learning. Training of competitive clusters could amplify the responses of specific groups to specific stimuli. As such, it would associate those groups with each other and with a specific appropriate response. Normally, when competition for learning is in effect, only the weights belonging to the winning processing element will be updated.

At the present state of the art, unsupervised learning is not well understood and is still the subject of research. This research is currently of interest to the government because military situations often do not have a data set available to train a network until a conflict arises.

### **2.4.3 Learning Rates**

The rate at which ANNs learn depends upon several controllable factors [16]. In selecting the approach there are many trade-offs to consider. Obviously, a slower rate means a lot more time is spent in accomplishing the off-line learning to produce an adequately trained system. With the faster learning rates, however, the network may not be able to make the fine discriminations possible with a system that learns more slowly. Researchers are working on producing the best of both worlds.

Generally, several factors besides time have to be considered when discussing the off-line training task, which is often described as "tiresome." Network complexity, size, paradigm selection, architecture, type of learning rule or rules employed, and desired accuracy must all be considered. These factors play a significant role in determining how long it will take to train a network. Changing any one of these factors may either extend the training time to an unreasonable length or even result in an unacceptable accuracy.

Most learning functions have some provision for a learning rate, or learning constant. Usually this term is positive and between zero and one. If the learning rate is greater than one, it is easy for the learning algorithm to overshoot in correcting the weights, and the network will oscillate. Small values of the learning rate will not correct the current error as quickly, but if small steps are taken in correcting errors, there is a good chance of arriving at the best minimum convergence.

#### **2.4.4 Learning Laws**

Many learning laws are in common use. Most of these laws are some sort of variation of the best known and oldest learning law, Hebb's Rule [16]. Research into different learning functions continues as new ideas routinely show up in trade publications. Some researchers have the modeling of biological learning as their main objective. Others are experimenting with adaptations of their perceptions of how nature handles learning. Either way, man's understanding of how neural processing actually works is very limited. Learning is certainly more complex than the simplifications represented by the learning laws currently developed. A few of the major laws are presented as examples.

**Hebb's Rule:** The first, and undoubtedly the best known, learning rule was introduced by Donald Hebb [16]. The description appeared in his book *The Organization of Behavior* in 1949. His basic rule is: If a neuron receives an input from another neuron and if both are highly active (mathematically have the same sign), the weight between the neurons should be strengthened.

**Hopfield Law:** It is similar to Hebb's rule with the exception that it specifies the magnitude of the strengthening or weakening [16]. It states, "if the desired output and the

input are both active or both inactive, increment the connection weight by the learning rate, otherwise decrement the weight by the learning rate."

**The Delta Rule:** This rule is a further variation of Hebb's Rule. It is one of the most commonly used [15]. This rule is based on the simple idea of continuously modifying the strengths of the input connections to reduce the difference (the delta) between the desired output value and the actual output of a processing element. This rule changes the synaptic weights in the way that minimizes the mean squared error of the network. This rule is also referred to as the Widrow-Hoff Learning Rule and the Least Mean Square (LMS) Learning Rule.

The way that the Delta Rule works is that the delta error in the output layer is transformed by the derivative of the transfer function and is then used in the previous neural layer to adjust input connection weights. In other words, this error is back-propagated into previous layers one layer at a time. The process of back-propagating the network errors continues until the first layer is reached. The network type called Feedforward; Back-propagation derives its name from this method of computing the error term.

When using the delta rule, it is important to ensure that the input data set is well randomized. Well ordered or structured presentation of the training set can lead to a network which can not converge to the desired accuracy. If that happens, then the network is incapable of learning the problem.

**The Gradient Descent Rule:** This rule is similar to the Delta Rule in that the derivative of the transfer function is still used to modify the delta error before it is applied to the connection weights [15]. Here, however, an additional proportional constant tied to the learning rate is appended to the final modifying factor acting upon the weight. This rule is commonly used, even though it converges to a point of stability very slowly.

It has been shown that different learning rates for different layers of a network help the learning process converge faster. In these tests, the learning rates for those layers close to

the output were set lower than those layers near the input. This is especially important for applications where the input data is not derived from a strong underlying model.

**Kohonen's Learning Law:** This procedure, developed by Teuvo Kohonen, was inspired by learning in biological systems [16]. In this procedure, the processing elements compete for the opportunity to learn, or update their weights. The processing element with the largest output is declared the winner and has the capability of inhibiting its competitors as well as exciting its neighbors. Only the winner is permitted an output, and only the winner plus its neighbors are allowed to adjust their connection weights.

Further, the size of the neighborhood can vary during the training period. The usual paradigm is to start with a larger definition of the neighborhood, and narrow in as the training process proceeds. Because the winning element is defined as the one that has the closest match to the input pattern, Kohonen networks model the distribution of the inputs. This is good for statistical or topological modeling of the data and is sometimes referred to as self-organizing maps or self-organizing topologies.

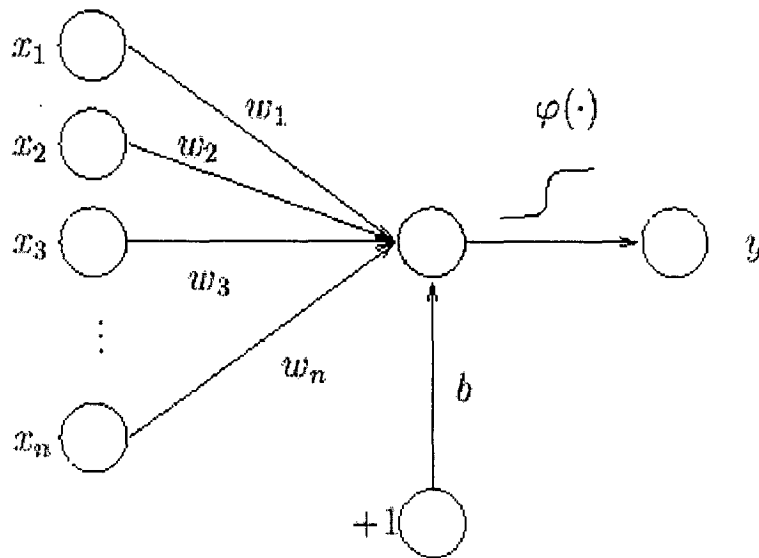
## 2.5 Multilayer Perceptrons

An MLP is a network of simple neurons called Perceptrons [14]. The perceptron computes a single output from multiple real-valued inputs by forming a linear combination according to its input weights and then possibly putting the output through some nonlinear activation function. Mathematically this can be written as

$$y = \Phi \left( \sum_{i=1}^n w_i x_i + b \right) = \Phi(W^T X + b) \quad \text{-----} \quad 2.1$$

where  $W$  denotes the vector of weights,  $X$  is the vector of inputs,  $b$  is the bias and  $\Phi$  is the activation function. A signal-flow graph of this operation is shown in Figure 2.1. The original Rosenblatt's perceptron used a Heaviside step function as activation function  $\Phi$ . Nowadays, and especially in multilayer networks, the activation function is often chosen to be the logistic sigmoid  $1/(1+e^{-x})$  or the hyperbolic tangent  $\tanh(x)$ . They are related by  $(\tanh(x)+1)/2 = 1/(1+e^{-2x})$ . These functions are used because they are mathematically

convenient and are close to linear near origin while saturating rather quickly when getting away from the origin. This allows MLP networks to model well both strongly and mildly nonlinear mappings.



**Fig 2.1: Signal-flow graph of the Perceptron**

A single perceptron is not very useful because of its limited mapping ability. No matter what activation function is used, the perceptron is only able to represent an oriented ridge-like function. The perceptrons can, however, be used as building blocks of a larger, much more practical structure. A typical *multilayer* perceptron (MLP) network consists of a set of source nodes forming the *input layer*, one or more *hidden layers* of computation nodes, and an *output layer* of nodes. The input signal propagates through the network layer-by-layer.

The computations performed by such a feedforward network with a single hidden layer with nonlinear activation functions and a linear output layer can be written mathematically as

$$X = f(s) = B\Phi(As+a)+b \quad \text{-----} \quad 2.2$$

where  $s$  is a vector of inputs and  $X$  is a vector of outputs.  $A$  is the matrix of weights of the first layer,  $a$  is the bias vector of the first layer.  $B$  and  $b$  are, respectively, the weight matrix and the bias vector of the second layer. The function  $\Phi$  denotes an element wise nonlinearity. The generalization of the model to more hidden layers is obvious.

While single-layer networks composed of parallel perceptrons are rather limited in what kind of mappings they can represent, the power of an MLP network with only one hidden layer is surprisingly large. Such networks, like the one in Equation 2.2, are capable of approximating any continuous function  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  to any given accuracy, provided that sufficiently many hidden units are available.

MLP networks are typically used in *supervised learning* problems. This means that there is a training set of input-output pairs and the network must learn to model the dependency between them. The training here means adapting all the weights and biases ( $A, B, a, b$ ) in Equation (2.2) to their optimal values for the given pairs  $[s(t), x(t)]$ . The criterion to be optimized is typically the squared reconstruction error  $\sum_t \|f(s(t)) - s(t)\|^2$

The supervised learning problem of the MLP can be solved with the *back-propagation algorithm* [14]. The algorithm consists of two steps. In the *forward pass*, the predicted outputs corresponding to the given inputs are evaluated as in Equation (2.2). In the *backward pass*, partial derivatives of the cost function with respect to the different parameters are propagated back through the network. The chain rule of differentiation gives very similar computational rules for the backward pass as the ones in the forward pass. The network weights can then be adapted using any gradient-based optimization algorithm. The whole process is iterated until the weights have converged.

The MLP network can also be used for unsupervised learning by using the so called *auto-associative* structure. This is done by setting the same values for both the inputs and the outputs of the network. The extracted sources emerge from the values of the hidden neurons. This approach is computationally rather intensive. The MLP network has to have at least three hidden layers for any reasonable representation and training such a network is a time consuming process.

## 2.6 Radial Basis Functions

Radial Basis Functions are emerged as a variant of artificial neural networks in late 80's. However, their roots are entrenched in much older pattern recognition techniques as for example potential functions, function approximation, spline interpolation and mixture models [18]. RBF's are embedded in a two-layer network where each hidden unit implements a radial activated function. The output units implement a weighted sum of hidden unit outputs. The input into an RBF network is nonlinear while the output is linear. Their excellent approximation capabilities have been studied in [19, 20]. Due to their nonlinear approximation properties, RBF networks are able to model complex mappings, which perceptron neural networks can only model by means of multiple intermediary layers.

In order to use radial basis function we need to specify the hidden unit activation function, the number of processing units, a criterion for modeling a given task and a training algorithm for finding the parameters of the network. Finding the RBF weights is called network training. If we have a set of input output pairs, called training set, we optimize the network parameters in order to fit the network outputs to given inputs. The fit is evaluated by means of cost function, usually assumed to be the mean square error. After training, the RBF network can be used with data whose underlying statistics is similar to that of the training set. On-line training algorithms adapt the network parameters to the changing data statistics [22]. RBF networks have been successfully applied to a large diversity of applications including interpolation [24], chaotic time-series modeling [21], channel equalization [24,25], system identification, control engineering [23], speech recognition [24], image restoration, shape-from-shading, 3-D object modeling, motion estimation and moving object segmentation, data fusion, etc.

### 2.6.1 Network Topology

Radial basis functions are embedded into two layer neural network; such a network is characterized by a set of inputs and a set of outputs. In between the inputs and outputs there is a layer of processing units called hidden units. Each of them implements a radial



basis function .The way in which the network is used for data modeling is different when approximating time series and in pattern classification .In the first case, the network inputs represent data samples at certain past time laps, while the network has only one output representing a signal value. In a pattern classification application the inputs represent feature entries, while each output corresponds to a class. The hidden units correspond to subclasses in this case [24].

Various functions have been tested as activation functions for RBF networks [25].In time-series modeling the most used activation function is the thin plate spline [25]. In pattern classification applications the Gaussian function is preferred [19,20,23,24]. Mixtures of gaussians have been considered in various scientific fields. The Gaussian activation function for RBF networks is given by:

$$\Phi_j (X) = \exp[ -(X- \mu_j)^T \Sigma_j^{-1} (X- \mu_j) ] \quad \text{-----} \quad 2.3$$

For  $j= 1 \dots L$ , where  $X$  is the input feature vector,  $L$  is the number of hidden units,  $\mu_j$  and  $\Sigma_j$  are the mean and covariance matrix of the  $j$ th gaussian function. In certain approaches a polynomial term is added to the above expression as in [22], while in others the gaussian function is normalized to the sum of all gaussian components as in the Gaussian mixtures estimation. Geometrically, a radial basis function represents a bump in the multi dimensional space, whose dimension is given by the number of entries. The mean vector  $\mu_j$  represent the location, while  $\Sigma_j$  models the shape of the activation function. Statistically, activation function models a probability density function where  $\mu_j$  and  $\Sigma_j$  represent the first and second order statistics.

The output layer implements a weighted sum of hidden unit outputs:

$$\Psi_k (X) = \sum_{j=1}^L \lambda_{jk} \Phi_j (X) \quad \text{-----} \quad 2.4$$

For  $k=1 \dots M$  where  $\lambda_{jk}$  are the output weights, each corresponding to the connection between a hidden unit and an output unit and  $M$  represent the number of output units. The weights  $\lambda_{jk}$  show the contribution of a hidden unit to the respective output unit. In a

classification problem if  $\lambda_{jk} > 0$  the activation field of the hidden unit  $j$  is contained in the activation field of the output unit  $k$ .

$$y_k(X) = 1 / (1 + \exp[-\Psi_k(X)]) \text{ for } k=1, \dots, M \quad \text{-----} \quad 2.5$$

### 2.6.2 Properties of RBF

The RBF's are characterized by their localization (center) by an activation hyper surface. In this case of Gaussian functions these are modeled by the two parameters  $\mu_j$  and  $\Sigma_j$ . The hyper surface is a hyper surface in the case when the covariance matrix is diagonal and has the diagonal elements equal, and is a hyper ellipsoid in the general case. In the general case of a hyper ellipsoid, the activation function influence decreases according to the mahalanobis distance from the center. This means that data samples decreases according to the mahalanobis distance from the center. This means that data samples located at a large mahalanobis distance from the RBF center will fail to activate that basis function. The maximum activation is achieved when the data samples coincides with the mean vector. Gaussian basis functions are quasi orthogonal. The product of two basis functions, whose centers are far away from the each other with respect to their spreads, is almost zero.

The RBF's can be thought of as potential functions. Hidden units whose output weights  $\lambda_{jk}$  have the same sign for a certain output unit, have their activation fields joined together in the same way as electric charges of the same sign form electric fields. For hidden units with output weights of different sign, their activation fields will correspond to the electrical field of opposite sign charges. Other similarities are with windowing functions [10] and with clustering. RBF's properties made them attractive for interpolation and functional modeling. As a direct consequence, RBF's have been employed to model probability density functions.

### 2.6.3 Training Algorithms

By means of training, the neural network models the underlying function of a certain mapping. In order to model such a mapping we have to find the network weights and topology. There are two categories of training algorithms: supervised and unsupervised. RBF networks are used mainly in supervised applications. In a supervised application, we are provided with a set of data samples called training set for which the corresponding network outputs are known. In this case the network parameters are found such that they minimize a cost function:

$$\text{Min} \sum_{i=1}^Q (Y_k(X_i) - F_k(X_i))^T (Y_k(X_i) - F_k(X_i)) \quad \text{-----} \quad 2.6$$

Where  $Q$  is the total number of vectors from the training set  $Y_k(X_i)$  denotes the RBF output vector and  $F_k(X_i)$  represents the output vector associated with the a data sample  $X_i$  from the training set. In unsupervised training the output assignment is not available for the given set. A large variety of training algorithms has been tested for training RBF networks. In the initial approaches, to each data sample was assigned a basis function. This solution proved to be expensive in terms of memory requirement and the number of parameters. On the other hand, exact fit to the training data may cause bad generalization. Other approaches chose randomly or assumed known the hidden unit weights and calculate the output weights  $\lambda_k$  by solving a system of equations whose solution is given in the training set [22]. The matrix inversion required in this approach is computationally expensive and could cause numerical problems in certain situations (when the matrix is singular). In some cases [23], the radial basis function centers are uniformly distributed in the data space. The function to be modeled is obtained by interpolation. In some cases less basis functions than given data samples are used. A least squares solution that minimizes the interpolation error is proposed.

An adaptive training algorithm for minimizing a given cost function is a gradient descent algorithm. Back propagation algorithm adapts iteratively the network weights considering the derivatives of the cost function [19] with respect to those weights [20,24]. Back

propagation algorithm may require several iterations and can get stuck into a local minimum of the cost function [19].

## 2.7 Comparison of RBF networks and MLP Networks

Radial basis function networks and multi layer perceptrons play very similar roles in that they both provide techniques for approximating arbitrary non linear functional mappings between multidimensional spaces. In both cases the mappings are expressed in terms of parameterized compositions of functions of single variables. The particular structures of the two networks are very different, however, and so it is interesting to compare them in more detail. Some of the important differences between the multi-layer perceptron and radial basis function networks are as follows:

1. The hidden unit representations of the multilayer perceptron depend on weighted linear summations of the inputs, transformed by monotonic activation functions. Thus the activation of a hidden unit in a multi-layer perceptron is constant on surfaces which consist of parallel  $(d-1)$  dimensional hyperplanes in  $d$ -dimensional input space. By contrast, the hidden units in a radial basis function network use distance to a prototype vector followed by transformation with a localized function. The activation of a basis function is therefore constant on concentric  $(d-1)$  dimensional hypersphere.
2. A multi-layer perceptron can be said to form a distributed representation in the space of activation values for the hidden units since, for a given input vector, many hidden units will typically contribute to the determination of the output value. During training, the functions represented by the hidden units must be such that, when linearly combined by the final layer of weights, they generate the correct outputs for a range of possible input values. The interference and cross coupling between the hidden units which this requires results in the network training process being highly nonlinear with problems of local minima, or nearly flat regions in the error function arising from near cancellations in the effects of different weights. This can lead to very slow convergence of the training

procedure even with advanced optimization strategies. By contrast, a radial basis function network with localized basis functions form a representation in the space of hidden units, which is local with respect to the input space because, for a given input vector, typically only a few hidden units will have significant activations.

3. A multi-layer perceptron often has many layers of weights, and a complex pattern of connectivity, so that not all possible weights in any given layer are present. Also, a variety of different activation functions may be used within the same network. A radial basis function network, however, generally has a simple architecture consisting of two layers of weights, in which the first layer contains the parameters of the basis functions, and the second layer forms linear combinations of the activations of the basis functions to generate the outputs.
4. All of the parameters in a multilayer perceptron are usually determined at the same time as part of a single global training strategy involving supervised training. A radial basis function network, however, is typically trained in two stages, with the basis functions being determined first by unsupervised techniques using the input data alone, and the second –layer weights subsequently being found by fast linear supervised methods.

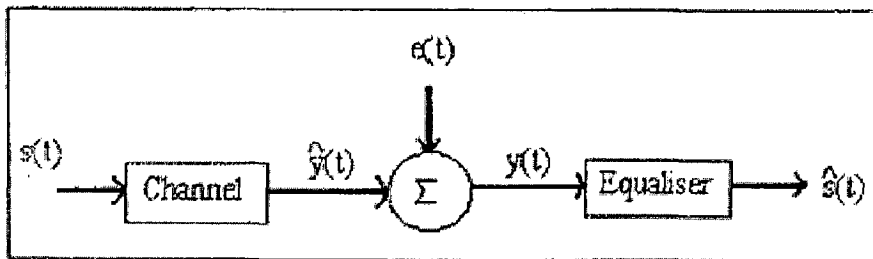
## **2.8 Channel Equalization**

High speed efficient data transmission over physical communication channels has been of great concern in the past decade. Communication channels are usually modeled as linear filters having a low pass frequency response. The filters used to model the channels may not behave like ideal filters. Thus, in the case of these non-ideal filters, their amplitude and the envelope delay response are not constant within the bandwidth concern. The channel modeled using these filters distort the transmitted signal in both amplitude and phase, causing what is known as intersymbol interference (ISI). In addition to linear distortion, the transmitted symbols are subject to other impairments such as thermal noise, impulse noise, cross talk interference, causing further distortions to the received symbols. Signal processing techniques used at the receiver, to overcome these

interferences, so as to restore the transmitted symbols and recover their information, are referred to as “equalization methods”.

Estimation theory suggests that the best performance for symbol detection is obtained by using a maximum likelihood Sequence Equalizer (MLSE) for the entire symbol sequence, which involves batch processing scheme. But the computational complexity of MLSE is prohibitive and this has led to the popularity of equalizers that make decisions symbol by symbol as an alternative. The optimal solution for these symbol decision equalizers has been approached using the Bayesian decision theory. It can be seen from Bayesian solution that the optimal solution corresponds to non linear classification problem.

### 2.8.1 Equalization Problem



**Fig 2.2: Discrete time model of data transmission system**

A discrete time model of a digital communication system is shown in Fig 2.2 where the input digital sequence  $s(t)$  is transmitted through a dispersive channel. Often, this is non-linear channel, like the one shown in equation (2.7)

$$y(t) = x(t) + k_1 x^2(t) + k_2 x^3(t) + e(t) \quad \text{-----} \quad 2.7$$

$$H(Z) = X(Z)/Y(Z) = \sum_{i=0}^{nh} h_i z^{-i} \quad \text{-----} \quad 2.8$$

Where  $k_1, k_2$  and  $k_3$  are constants. The linear component  $H(Z)$ , of the channel can be modeled as a FIR filter, where  $n_h+1$  is the length of the channel impulse response. This channel includes the effects of the transmitter filter, the transmission medium, the receiver matched filter, and other components. The transmitted symbol  $s(t)$  is assumed to be an equiprobable and independent binary sequence taking values of either +1 or -1. The noise free output of the channel,  $\hat{y}(t)$  is added with a zero mean Gaussian white noise,  $e(t)$  to obtain a noisy channel output  $y(t)$ .

The equalizer performs the task of recovering an estimate of  $\hat{s}(t)$  of the transmitted symbols,  $s(t)$ , based on the noisy channel observation  $y(t)$ . Using an estimate theory [6], it is known that the Maximum Likelihood Sequence Estimator (MLSE) for the entire transmitted sequence would yield the best performance for symbol detection. This method however, is highly complex with high computational requirements, and the delay in decision output is often unacceptable in many practical communication systems. Thus, most practical equalization systems employ some form of symbol-by-symbol, decision-making architecture. In such architecture, the past  $m$  channel observations are used to make an estimate,  $\hat{s}(t-\tau)$  of the input symbol,  $s(t)$ , with delay  $\tau$ . For the channel given in eqn (2.8), there will be  $n_s$ , combinations of the input sequence where  $n_s = 2^{nh+m}$ , and  $n_h$  is the order of the linear component of the channel. The  $n_s$  input sequence

$$s(t)=[s(t) \dots s(t-m+1-n_h)]^T \quad \text{-----} \quad 2.9$$

would result in  $n_s$  points of noise free channel output vector.

$$y(t)=[y(t) \dots y(t-m+1)]^T \quad \text{-----} \quad 2.10$$

These output vectors are also referred to as desired channel states, and are partitioned into different classes,  $Y^+_{m,\tau}$  &  $Y^-_{m,\tau}$ , for  $s(t-\tau) = 1$  &  $s(t-\tau) = -1$  respectively. Due to the Additive White Gaussian Noise (AWGN), the channel outputs will form clusters around each of these desired channel states. The noisy observation vector

$$y(t) = [y(t) \dots y(t-m+1)]^T \quad \text{-----} \quad 2.11$$

is used to determine the transmitted symbol  $s(t-\tau)$ , according to the Bayes decision theory [6,8]. For equiprobable symbols, the Bayesian decision function is defined by

$$f_B(\mathbf{y}(t)) = \sum_{i=1}^{N_+} \exp(-\|\mathbf{y}(t) - \mathbf{y}_i\|^2 / 2\sigma_c^2) - \sum_{j=1}^{N_-} \exp(-\|\mathbf{y}(t) - \mathbf{y}_j\|^2 / 2\sigma_c^2) \quad \text{-----} \quad 2.12$$

which is a hyper surface in the observation space. As RBF networks are well suited for realizing this Bayesian function [8] as well as performing such non-linear mapping we investigate an algorithm that could build such a equalizer network.

It is well known that neural networks are well suited for solving nonlinear classification problems. Because of this, multiplayer feed forward neural networks, radial basis function (RBF) networks and recurrent neural networks have gained popularity in their use for equalization problems. Initial work had demonstrated that multiplayer perceptron (MLP) equalizers are superior to conventional transversal and decision feedback equalizers in terms of the equalizer performance metric, symbol error rate. The problems which severely restrict the practical implementation of MLP's are their long training times and lack of a methodology for the network architecture selection.

Here in RBF [26], we take a completely different approach by viewing the design of a neural network as a curve-fitting (approximation) problem in a high dimensional space. According to the viewpoint, learning is equivalent to finding a surface in a multi dimensional space that provides a best fit to the training data, with the criterion for the "best fit" being measured in some statistical sense. In the context of neural network, the hidden units provide a set of "functions" that constitute an arbitrary "basis" for the input pattern when they are expanded into hidden space; these functions are called radial basis functions.

The sequential learning MRAN algorithm employs a scheme for adding and pruning RBF's hidden neurons, so as to achieve a parsimonious network structure. In MRAN



algorithm, the network begins with no hidden neurons. As each training data pair (input and output) is received the network builds itself up based on three growth criteria. The algorithm adds new hidden neurons or adjusts the existing network parameters according to the training data received. The algorithm also incorporates a pruning strategy that is used to remove the hidden neurons that do not contribute significantly to the output.

The construction of radial basis function (RBF) network, in its most basic form, involves three layers with entirely different roles. The input layer is made up of source nodes (sensory units) that connect the network to its environment. The second layer, the only hidden layer in the network, applies a nonlinear transformation from the input space to the hidden layer; in most cases the hidden layer is of high dimensionality. The output layer is linear; supplying the response of the network to the activation pattern (signal) applies to the input layer.

In the following chapters, we present the use of MRAN for linear and non linear channel equalization problems. The superiority of this method over existing methods is that, separate channel order estimation is not necessary. The algorithm uses an extended kalman filter to determine the weight and width of each of the nodes. This is different from previous studies, where the width values have to be set to an estimate of the noise variance of the received data. The weights are also not fixed as 1 or -1 as suggested by Chen [8]. This means the binary nature of the data is not exploited. However the advantage in MRAN is that the RBF nodes could adjust their weight and width values, so as to accommodate more data around their location. This would mean that the resultant network may have even fewer nodes than that required, if a node is to be placed in each of the desired channel states.

## **2.9 Summary**

In summary, artificial neural networks are one of the promises for the future in computing. They offer an ability to perform tasks outside the scope of traditional processors. They can recognize patterns within vast data sets and then generalize those

patterns into recommended courses of action. Neural networks learn, they are not programmed.

Yet, even though they are not traditionally programmed, the designing of neural networks does require a skill. It requires an "art." This art involves the understanding of the various network topologies, current hardware, current software tools, the application to be solved, and a strategy to acquire the necessary data to train the network. This art further involves the selection of learning rules, transfer functions, summation functions, and how to connect the neurons within the network. Then, the art of neural networking requires a lot of hard work as data is fed into the system, performances are monitored, processes tweaked, connections added, rules modified, and on and on until the network achieves the desired results.

These desired results are statistical in nature. The network is not always right. It is for that reason that neural networks are finding themselves in applications where humans are also unable to always be right. Neural networks can now pick stocks, cull marketing prospects, approve loans, deny credit cards, tweak control systems, grade coins, and inspect work.

Yet, the future holds even more promises. Neural networks need faster hardware. They need to become part of hybrid systems which also utilize fuzzy logic and expert systems. It is then that these systems will be able to hear speech, read handwriting, and formulate actions. They will be able to become the intelligence behind robots who never tire nor become distracted. It is then that they will become the leading edge in an age of "intelligent" machines.

## MINIMAL RESOURCE ALLOCATION NETWORKS

---

### 3.1 Introduction

In this chapter, a new learning algorithm for a Minimal Resource Allocation Network (MRAN) is proposed. MRAN adopts the basic idea of Resource Allocation Network via Extended Kalman Filter (RANKEF) algorithm and augments it with a pruning strategy to obtain a minimal RBF network. For better understanding a brief review of RANKEF algorithm is presented. Here, a pruning strategy is introduced to remove the hidden neurons which have insignificant contribution to the total output of the network. However, the oscillations in the transition of the number of hidden neurons may result from the introduction of this pruning strategy in the learning algorithm. To overcome the oscillations and make the transition smooth, an additional growth criterion based on Root Mean Square (RMS) error of the network output over a sliding window is presented .

### 3.2 Resource Allocation Network via Extended Kalman Filter (RANEKF) Algorithm

Before getting to the details of the learning algorithms for the RBF neural networks, we first introduce some notations which will be used in the algorithms. The response of one hidden unit to the network input at the  $i^{th}$  instant  $x_i$  can be expressed as follows,

$$\Phi_k(x_i) = \exp[-(\|x_i - \mu_k^i\|^2 / (\sigma_k^i)^2)] \quad , \quad (k=1, \dots, h) \quad \text{-----} \quad 3.1$$

where  $\mu_k^i$  is the center vector for the  $k^{th}$  hidden unit at  $i^{th}$  instant and  $\sigma_k^i$  is the width for the Gaussian function at that time.  $\| \cdot \|$  denotes the Euclidean norm and  $h$  indicates the

total number of hidden neurons in the network. For networks with multiple outputs  $y_i$  of  $p$  dimension, the overall network response is a mapping  $f: \mathbb{R}^q \rightarrow \mathbb{R}^p$ , which is

$$y_i = f(x_i) = \lambda_0^i + \sum_{k=1}^h \lambda_k^i \Phi_k(x_i), \quad \text{-----} \quad 3.2$$

where  $x_i \in \mathbb{R}^q$ . The coefficient vector  $\lambda_k^i$  is the connecting weight vector of the  $k^{\text{th}}$  hidden unit to output layer.

The learning process for both RANKEF and RAN (Resource Allocation Network) involves allocation of new hidden units as well as adjusting the network parameters. The network begins with no hidden units. As observations are received, the network grows by using some of them as the new hidden units. The satisfaction of the two following criteria decides whether a new hidden unit should be added to the network for an observation  $(x_i, y_i)$ :

$$\|x_i - \mu_{nr}^i\| > \epsilon_i \quad \text{-----} \quad 3.3$$

$$e_i = \|y_i - f(x_i)\| > e_{min} \quad \text{-----} \quad 3.4$$

where  $\mu_{nr}^i$  is the center (of hidden unit) which is closest to  $x_i$ .  $\epsilon_i$  and  $e_{min}$  are thresholds to be selected appropriately. When a new hidden unit is added to the network, the parameters associated with the unit are as follows,

$$\lambda_{h+1}^i = e_n \quad \text{-----} \quad 3.5$$

$$\mu_{h+1}^i = x_n \quad \text{-----} \quad 3.6$$

$$\sigma_{h+1}^i = K \|x_n - \mu_{nr}^i\| \quad \text{-----} \quad 3.7$$

where  $K$  is an overlap factor which determines the overlap of the responses of hidden units in the input space.

The algorithms used to adapt network parameters in RANKEF and RAN is different. Least Mean Square (LMS) algorithm is employed in RAN to adjust the existing network parameters. An alternative approach to LMS is Extended Kalman Filter (EKF) algorithm,

which is used in RANKEF. According to [27], using EKF algorithm to adapt network parameters tends to achieve faster convergence than LMS does but at the expense of added memory requirement and computational complexity.

For RANEKF, when the observation  $(x_i, y_i)$  does not meet the criteria for adding a new hidden unit, the network parameters  $w_i$  are adjusted using EKF .

### **3.3 Limitations of RANEKF**

RANEKF and RAN have made great progress by realizing more compact networks with sequential learning as compared with the earlier algorithms which produced networks with more number of apriori fixed hidden neuron centers. However, the algorithms for both RAN and RANEKF only allow new neurons to be added. Once a hidden unit is created, it can never be removed. Because of this, both RAN and RANEKF may produce networks in which some hidden units, although active initially, may subsequently end up contributing little to the network output. If such inactive hidden units can be detected and removed as learning progresses, then a more parsimonious network topology can be realized.

### **3.4 Pruning Strategy**

This section presents a scheme to prune the superfluous hidden unit in the network. Cheng etc. [9] has proposed a method of pruning hidden neurons during the network batch training. In their scheme, for every iteration, the weight for each hidden unit is checked. Those units whose weights are less than a threshold will be deleted. The pruning strategy of MRAN was inspired by this idea. However, for MRAN, pruning is performed in a sequential way instead of being conducted in each iteration during network batch training. Furthermore, Cheng's pruning method only considers the weight value for each hidden unit, while the pruning strategy of MRAN takes whole output value of each

hidden unit into consideration. Following expression is the output  $o_{lk}^i$  of the hidden unit  $k$  to the  $l^{th}$  output node ( $l=1, \dots, p$ ) at the  $i^{th}$  instant,

$$o_{lk}^i = \lambda_{lk} \exp[-(\|x_i - \mu_k^i\|^2 / (\sigma_k^i)^2)] \quad \text{-----} \quad 3.8$$

By observing this expression, we may note that the amount of contribution for each hidden unit is affected by three key values. Weight  $\lambda_{lk}$  is only one of the three values. Therefore, considering only the weight value of the hidden units is not enough to judge the contribution of that hidden unit. The other two elements also have to be considered. They are  $\sigma_k^i$  and  $\|x_i - \mu_k^i\|$ . If the width value  $\sigma_k^i$  for the radial basis function in the  $k^{th}$  unit is small,  $o_{lk}^i$  might become small. Alternately if  $\|x_i - \mu_k^i\|$  is large, which means the center of this hidden unit is far away from the input pattern, the hidden units output at this instant will also become small. For the network with multiple outputs (i.e  $p > 1$ ),  $o_{lk}^i$  ( $l=1, \dots, p$ ) is one of the elements of output vector  $o_k^i$  for the  $k^{th}$  hidden unit at the  $i^{th}$  input,  $o_k^i = [o_{1k}^i, \dots, o_{lk}^i, \dots, o_{pk}^i]^T$  and  $o_k^i$  can be calculated by:

$$o_k^i = \lambda_k^i \Phi_k(x_i) \quad \text{-----} \quad 3.9$$

In order to decide whether or not a hidden neuron should be removed, all the elements in the output vector ( $o_k^i$ ) for all hidden units ( $k=1, \dots, h$ ) should be examined continuously. The hidden unit  $k$  is removed if and only if the value of every element in the output vector  $o_k^i$  is insignificant for the network output in a time period whose length is predefined (i.e  $i=t, (t-1), \dots, (t-n_w+1)$  where  $n_w$  is the length of time period.  $t$  is the current time and  $(t-n_w+1)$  is the past instant when the hidden unit  $k$  started to contribute insignificantly to the network output). Since inconsistency for comparison may be caused by using the absolute values of hidden neuron output, the outputs of hidden units are normalized and the normalized output vector  $r_k^i$  is in the form of  $r_k^i = [r_{1k}^i, \dots, r_{lk}^i, \dots, r_{pk}^i]^T$ , where

$$r_{lk}^i = \|o_{lk}^i\| / \|o_{l,max}^i\| \quad \text{-----} \quad 3.10$$

$\| o_{l,max}^i \|$  is the largest absolute hidden unit output value among all the hidden unit output values to the  $l^{th}$  output node at  $i$  instant. If the values for all the elements in  $r_k^i$  are less than a threshold for consecutive  $n_w$  observations, then it indicates that the  $k^{th}$  hidden neuron makes insignificant contribution to the whole output and should be removed from the network.

Even though the pruning helps to realize a minimal RBF network, the transitions in the number of hidden neurons are not always smooth. This motivated us to include an extra criterion for growth which would smooth the transitions and achieve the minimal size of the network in a short time.

### 3.5 Sliding Window RMS Criterion for Adding Hidden Neurons

If only the two criteria (equation.(3.3) and equation.(3.4)) proposed in section 3.2 are employed in network growth, the problem of over fitting may come up. For each observation, these two growth criteria judge the novelty of the input data purely based on network output error and the distance between the input data and the nearest existing center. When there is noise presented in the network input data, noise may also cause new hidden units to be added in the network. This is because the distance between the input data corrupted by the noise and the nearest existing hidden neuron center might be larger than the threshold value in equation (3.3). Also due to appearance of noise, the output error for the network may exceed  $e_{min}$  in eqn (3.4). In this case, the network with such growth criteria may result in growing for fitting the noise in the input data. This is the problem of overfitting for neural networks. In order to reduce the effect of noise in the network growth an also to make the transition of the number of hidden neurons smooth, we include a third criterion which uses the Root Mean Square (RMS) value of the output error over a sliding data window before adding a hidden unit. The RMS value of network output error at  $i^{th}$  observation is given by,

$$e_{rmsn}^i = [(\sum_{j=i-(M-1)}^i (e_j^T e_j) / M)^{1/2}] \quad \text{-----} \quad 3.11$$

where  $M=n_w$ ,  $e_j = y_j - f(x_j)$ , In the above equation, for each observation, sliding data window is employed to include a certain number of output errors based on  $y_i, y_{i-1}, \dots, y_{i-M}$ .

1). When a new observation arrives, the data in this window are updated by including the latest data and eliminating the oldest one. For example, at the  $i^{\text{th}}$  observation, the data window includes  $e_i, e_{i-1}, \dots, e_{i-(M-1)}$ , while, when the  $(i+1)^{\text{th}}$  observation arrives, the data window will contain  $e_{i+1}, e_i, \dots, e_{i-(M-2)}$ . In this sense, it seems that this data window slides along with the data obtained online, thus we call it as data sliding window. The size of the window is determined by the value of  $n_w$  i.e. the number of samples used to calculate the rot mean square error. Thus, the extra growth criterion to be satisfied is

$$e_{rms}^i > e_{min}^i \quad \text{-----} \quad 3.12$$

where  $e_{min}^i$  is a threshold value to be selected .

### 3.6 Summary

Here in this chapter Resource Allocation Network via Extended Kalman Filter (RANEKF) Algorithm is presented. RANEKF have made great progress by realizing more compact networks with sequential learning as compared with the earlier algorithms which produced networks with more number of apriori fixed hidden neuron centers. However, the algorithms for both RAN and RANEKF only allow new neurons to be added. Once a hidden unit is created, it can never be removed. Because of this, both RAN and RANEKF may produce networks in which some hidden units, although active initially, may subsequently end up contributing little to the network output. If such inactive hidden units can be detected and removed as learning progresses, then a more parsimonious network topology can be realized.

So by adding pruning strategy to that, a sequential learning algorithm for implementing minimal Radial Basis Function (RBF) neural networks has been developed. This learning algorithm for the network referred to as MRAN , not only allocates a new hidden neuron based on the novelty of observation but also prunes those hidden neuron units which have insignificant contribution to the outputs of the RBF network. Therefore, when applied to time-varying dynamic systems, the architecture of MRAN will be adjusted automatically on-line to fit closely the dynamics of the observation data.



## DESIGN AND IMPLEMENTATION

---

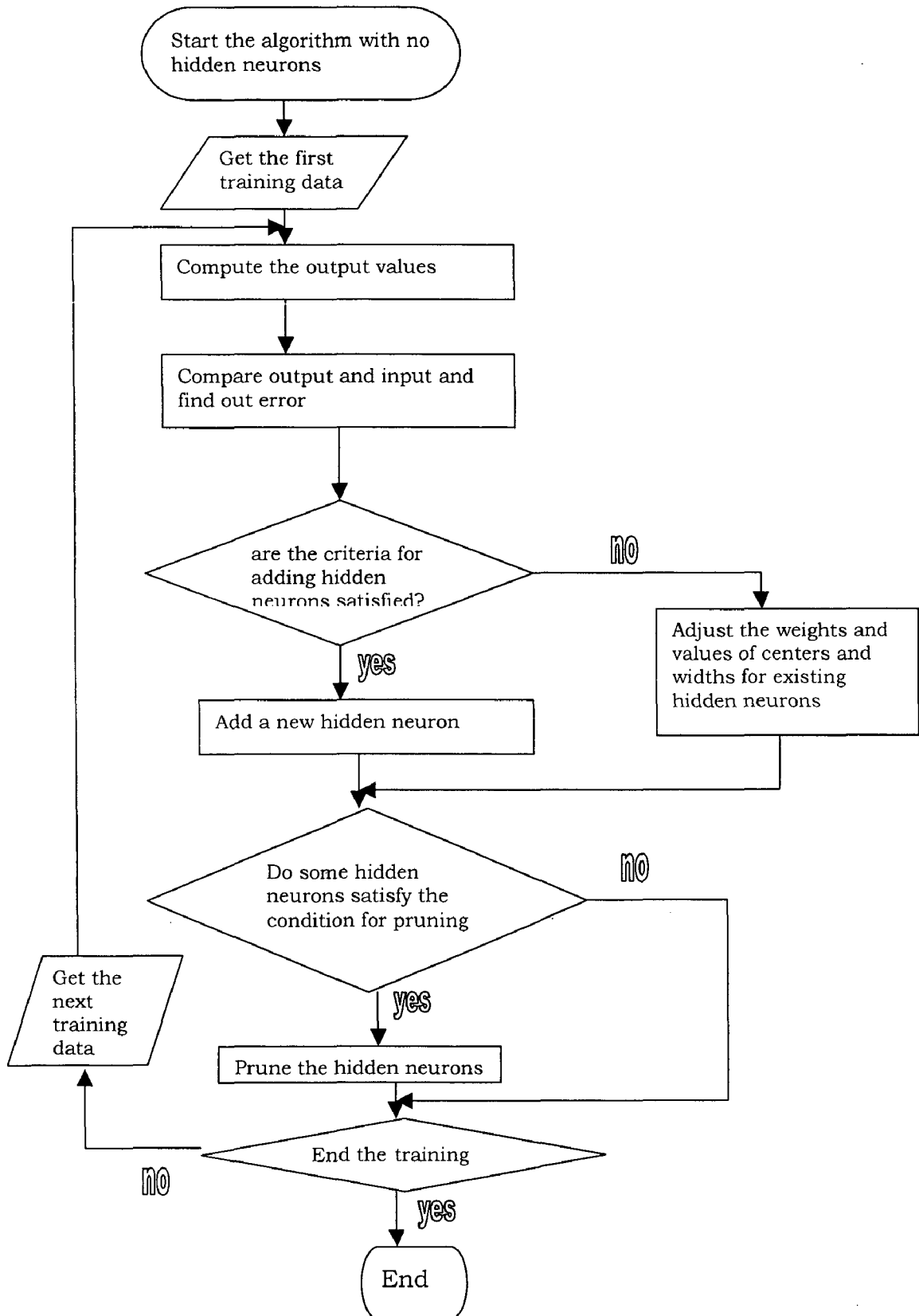
### 4.1 Introduction

Design and Implementation of MRAN sequential algorithm is presented here. Here implementation of RANEKF algorithm along with pruning strategy and sliding window RMS criterion are presented. Here in MRAN, not only the addition of hidden neurons but removal of superfluous hidden neurons which do not contribute much to the output for a certain period of time, is also done. This makes the network minimal.

### 4.2 Design

For MRAN, in the constructive stage of MRAN network, three criteria for adding hidden units are employed, which are presented in the previous chapter. Among them, the first two are identical to those in RANEKF for adding hidden units. The third criterion, called as RMS criterion is newly developed for MRAN. The aim of the criterion is to reduce the effect of noise in the network growth and also to make the transition of the number of hidden neurons smooth. Like RANEKF, Extended Kalman Filter algorithm is also employed in MRAN as the method for adaptation of existing network parameters. However, in RANEKF, hidden units are only allowed to be added in the network. There is no pruning in RANEKF. In order to remove the superfluous hidden neurons possible generated during network constructive phase and to make network more compact, a pruning strategy is developed in MRAN. Hence, for each observation, besides examining the innovation of the input data for network growth, the contribution of each hidden unit output also has to be observed for the pruning stage. Those hidden units with insignificant contribution will be removed from the network.

The flow diagram of MRAN algorithm is shown in Fig 4.1 below.



**Fig 4.1: Flow Diagram of MRAN Algorithm**

### 4.3 Implementation

Here I used MATLAB 6.1 version to perform channel equalization using minimal radial basis function. The MRAN is a sequential learning algorithm for minimum RBF neural network. Only a brief description of the algorithm is given here.

The output of the RBF network is of the form:

$$\Psi_j(X) = \lambda_0 + \sum_{j=1}^L \lambda_j \Phi_j(X) \quad \text{-----} \quad 4.1$$

Where  $\Psi_k(X)$  is the response of the  $j^{\text{th}}$  hidden neuron to the input  $x$ , and  $\lambda_j$  is the weight connecting the  $j^{\text{th}}$  hidden unit to the output unit.  $\lambda_0$  is the bias term. Here,  $j$  represents the number of hidden neurons in the network.  $\Phi_j(X)$  is a Gaussian function given by

$$\Phi_j(X) = \exp[-(\|X - \mu_j\|^2 / \sigma_j^2)] \quad \text{-----} \quad 4.2$$

where  $\mu_j$  is the centre and  $\sigma_j$  is the width of the Gaussian function.  $\| \cdot \|$  denotes the Euclidean norm.

In the algorithm, the network begins with no hidden units. As each input-output training data is received, and processed, the network is built up based on certain growth criteria. The algorithm adds hidden units, as well as adjusts the existing network, according to the data received. The criteria that must be met before a new hidden unit is added are:

$$\|x_n - \mu_{nr}\| > \epsilon_n \quad \text{-----} \quad 4.3$$

$$e_n = y_n - f(x_n) > e_{min} \quad \text{-----} \quad 4.4$$

$$e_{rmsn} = [(\sum_{i=n-(M-1)}^n e_i^2) / M]^{1/2} > e_{minl} \quad \text{-----} \quad 4.5$$

where  $\mu_{nr}$  is the centre (of the hidden unit) which is closest to  $x_n$ .  $\epsilon_n$ ,  $e_{min}$  and  $e_{minl}$  are thresholds to be selected appropriately. Equation 4.3 ensures that the new node to be added is sufficiently far from all the existing nodes. Equation 4.4 decides if the existing

nodes are insufficient to obtain a network output that meets the error specification. Equation 4.5 checks that the network has not met the required sum squared error specification for the past  $M$  outputs of the network. Only when all these criteria are met, is a new node added to the network.

When a new hidden neuron is added to the network the parameters associated with the unit are assigned as follows:

$$\lambda_{k+1} = e_n \quad \text{-----} \quad 4.6$$

$$\mu_{K+1} = x_n \quad \text{-----} \quad 4.7$$

$$\sigma_{K+1} = K \left\| x_n - \mu_{nr} \right\| \quad \text{-----} \quad 4.8$$

where  $k$  is overlap factor which determines the overlap of the response of a hidden unit in the input spaces. When the observations  $(x_n, y(n))$  does not meet the criteria for a new hidden unit to be added, Extended Kalman Filter (EKF) is used to adjust the parameters of the network,  $w = [\lambda_0, \lambda_1, \mu_1^T, \sigma_1, \dots, \lambda_k, \mu_k^T, \sigma_k]^T$ , given by

$$w(n) = w(n-1) + k_n e_n \quad \text{-----} \quad 4.9$$

where  $k_n$  is called the kalman gain vector given by

$$k_n = p_{n-1} a_n [R_n + a_n^T p_{n-1} a_n]^{-1} \quad \text{-----} \quad 4.10$$

where  $a_n$  is the gradient vector and has the following form

$$a_n = \Delta_w f(x_n) = [1, \Phi_1(x_n), \Phi_1(x_n)(2\lambda_1/\sigma_1^2)(x_n - \mu_1)^T, \Phi_1(x_n)(2\lambda_1/\sigma_1^3) \|x_n - \mu_1\|^2, \dots, \Phi_k(x_n), \Phi_k(x_n)(2\lambda_k/\sigma_k^2)(x_n - \mu_k)^T, \Phi_k(x_n)(2\lambda_k/\sigma_k^3) \|x_n - \mu_k\|^2]^T \quad \text{-----} \quad 4.11$$

$R_n$  is the variance of the measurement noise ;  $p_n$  is the error covariance matrix which is updated by

$$P_n = [I - k_n a_n^T] P_{n-1} + QI \quad \text{-----} \quad 4.12$$

Where  $Q$  is scalar that determines the allowed random step in the direction of gradient vector. Assume that the number of parameters to be adjusted is  $N$ ,  $P_n$  is therefore  $N \times N$  positive definite symmetric matrix. When a new hidden unit is allocated the dimensionality of  $P_n$  increases by

$$P_n = \begin{pmatrix} P_{n-1} & 0 \\ 0 & P_0 I_0 \end{pmatrix} \quad \text{-----} \quad 4.13$$

And the new rows and columns are initialized by  $P_0$ .  $P_0$  is an estimate of the uncertainty in the initial values assigned to the parameters, which in this algorithm is also the variance of the observations  $x_n$  and  $y(n)$ . The dimension of identity matrix is equal to the number of new parameters introduced by adding new hidden unit.

The algorithm also incorporates a pruning strategy, which is used to prune hidden nodes that do not contribute significantly to the outcome of the network, or are too close to each other. The former is done by observing the output of each of the hidden nodes for a period of time, and then removing the node that has not been contributing a significant output for that period. Also, if two hidden units are found to be closer than a threshold value, and the values of the parameters ( $\lambda, \sigma$ ) associated with the nodes are close, the two nodes are combined to form a single node, and the dimensions of the EKF are reduced.

#### 4.4 Summary

Here MATLAB 6.1 version has been used for implementation of Minimal Resource Allocation Network (MRAN) algorithm. This learning algorithm not only allocates a new hidden neuron but also prune those hidden neurons which have insignificant contribution for a certain period of time to the outputs.



## SIMULATION & RESULTS

---

### 5.1 Introduction

The performance of MRAN algorithm on a number of applications is presented here. These applications consist of problems which are both static and dynamic. The static problems include channel equalization and dynamic problems include nonlinear dynamic system identification. Here in channel equalization, different types of channels are considered. In nonlinear system identification, both nonlinear system with fixed dynamics and nonlinear system with varying dynamics are considered and the simulation results are presented here.

### 5.2 Results

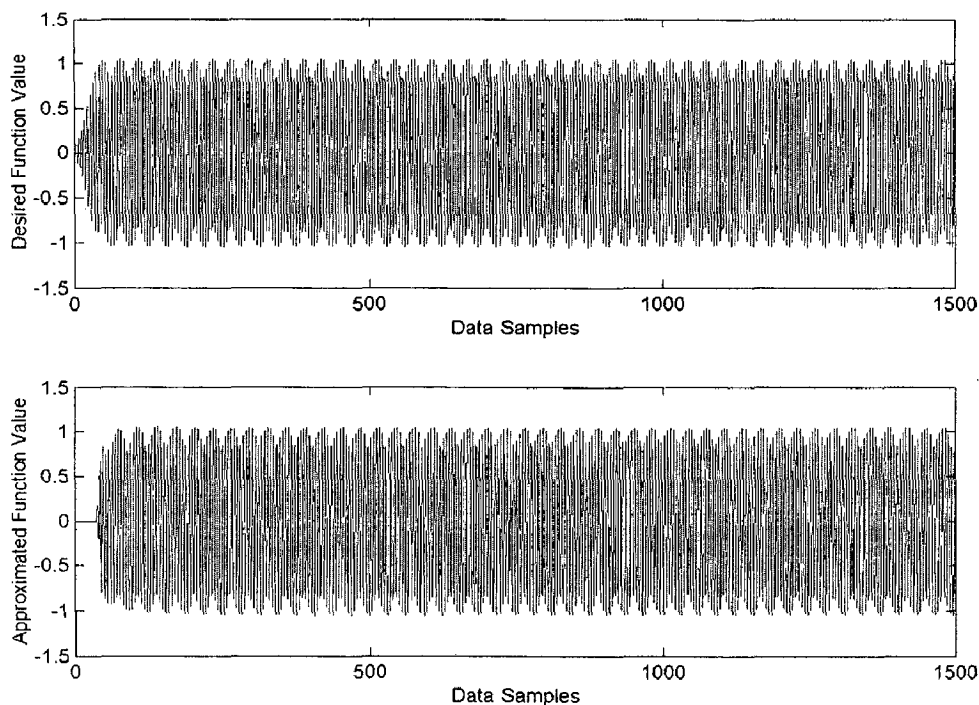
#### Model 1: Non Linear System Identification (Fixed Dynamics Case)

Here the algorithm is tested on same nonlinear dynamic systems discussed in Chen and Billings [11] and the results of this algorithm are compared with the results obtained in [11] using the Hybrid learning algorithm. The nonlinear system is of the form

$$y(n) = (0.8 - 0.5 \exp(-y(n-1)^2)) y(n-1) - (0.3 + 0.9 \exp(-y(n-1)^2)) y(n-2) + 0.1 \sin(3.1415928y(n-1)) + \text{noise}(n) \quad \text{-----} \quad 5.1$$

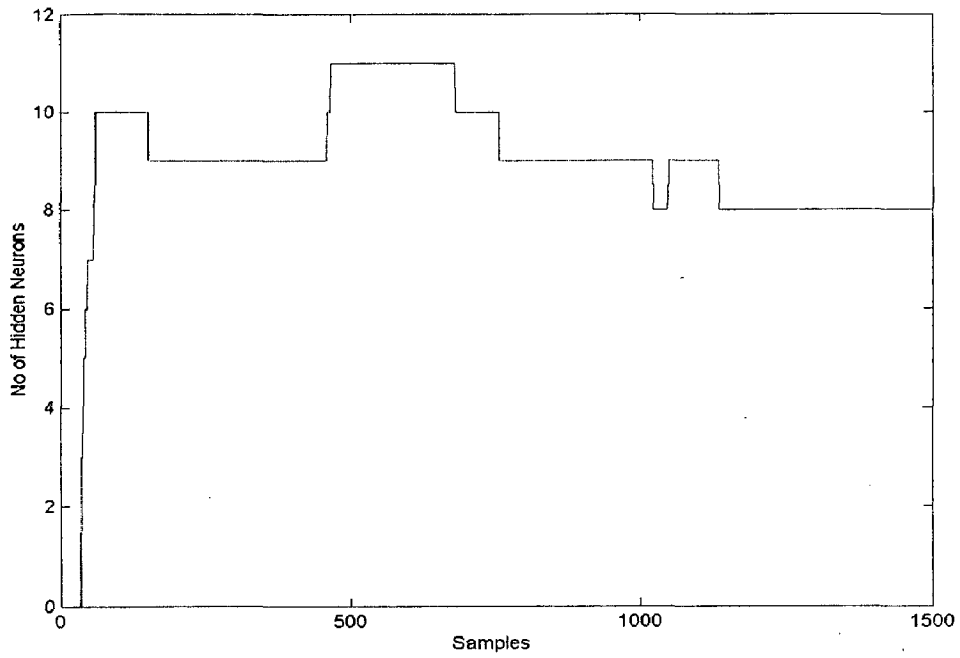
In eqn.5.1 noise(n) is a Gaussian white sequence. In [11], the hybrid learning algorithm is employed to identify the system. This algorithm applies n-means clustering algorithm for adjusting the RBF centers and least mean square algorithm for adjusting the RBF weights. The number of hidden neurons must be decided before the beginning of the identification process. The hidden neurons for algorithm can be neither added nor reduced during the learning process. 30 hidden neurons are employed in that network.

Here the network input vector is  $x = [y(n-2), y(n-1)]'$  and output  $y_n(n)$  is used to approximate  $y(n)$ . Thus, the network can be considered as one-step predictor. 1500 samples of data are generated by the above model and are used for identification process. The initial values  $y(0)$  and  $y(-1)$  are set as  $y(0)=0.1$ ,  $y(-1)=0.01$  respectively. Fig 5.2 shows the growth of hidden neurons, error i.e difference between desired and approximated values. Here the input and output vectors are same. The parameters required by the algorithm are selected as  $\epsilon_{\max} = 4.0$ ,  $\epsilon_{\min} = 0.4$ ,  $r = 0.96$ ,  $e_{\min} = 0.2$ ,  $K=0.87$ ,  $P_0 = R_n = 1.0$ ,  $Q=0.02$ ,  $\delta=0.0001$   $e'_{\min} = 0.4$ . As noise presents in the identification process, a data window with size of  $M= 25$  is employed here to avoid the problem of overfitting for the neural network. Here the number of hidden neurons achieved is much less than 30 required by the networks proposed in [11, 28]. It can be clearly seen that this algorithm not only reduces the size of the network but also reduces the network approximation errors.

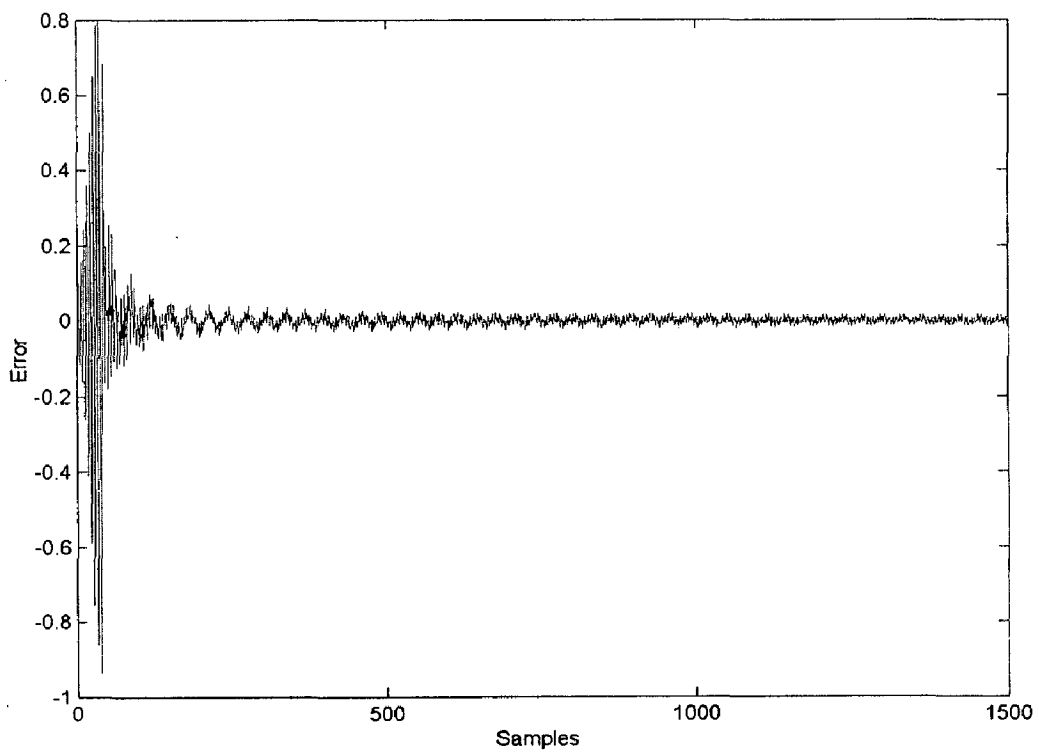


**Fig 5.1: Desired and Approximated function values for Model 1**





**Fig 5.2: Growth of Hidden Units as training progresses for Model 1**



**Fig 5.3: Error between Desired and Approximated as training progresses**

## Model 2: Non Linear System Identification (Varying Dynamics Case)

To study the adaptive capability of the minimal RBFNN algorithm, the dynamics of the system is deliberately changed during the simulation as follows:

$$y(n) = (0.8 - c_1(n) \exp(-y(n-1)^2)) y(n-1) - (0.3 + c_2(n) \exp(-y(n-1)^2)) y(n-2) + c_3(n) \sin(3.1415928y(n-1)) + \text{noise}(n) \quad \text{-----} \quad 5.2$$

Where  $c_1(n)$ ,  $c_2(n)$ ,  $c_3(n)$  are time varying coefficients and their expressions are shown as follows :

$$c_1(n) = 0 \quad \text{if } 2000 < n < 7000 \quad \text{-----} \quad 5.3$$

$$= 0.5 \quad \text{otherwise}$$

$$c_2(n) = 0 \quad \text{if } 3000 < n < 6000 \quad \text{-----} \quad 5.4$$

$$= 0.9 \quad \text{otherwise}$$

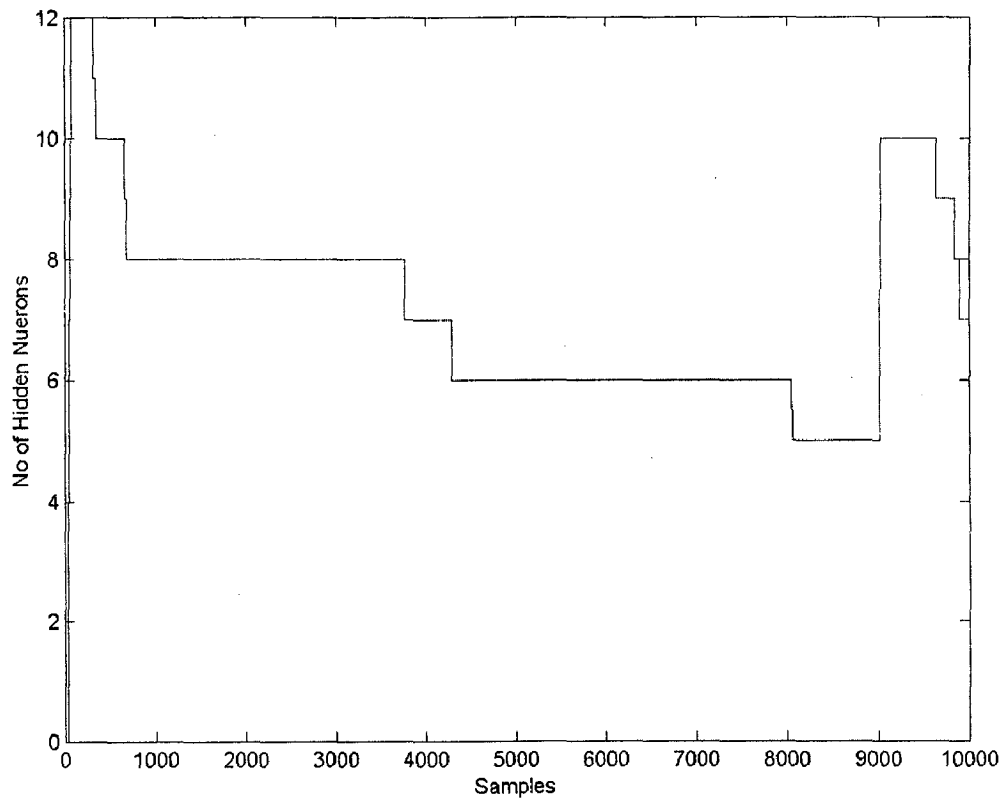
$$c_3(n) = 0 \quad \text{if } 4000 < n < 5000 \quad \text{-----} \quad 5.5$$

$$= 0.1 \quad \text{otherwise}$$

It can be seen that during the period  $n < 1500$   $c_1(n)$ ,  $c_2(n)$ ,  $c_3(n)$  are fixed at 0.5,0.9,0.1 respectively. After 2000 observations, the dynamics of this system changes in the way expressed by equations (5.3-5.5). The minimal RBF algorithm is used for online identification of this time varying non linear system.

Simulation results for a total of 10000 observations are presented in Fig 5.4. As  $c_1(n)$ ,  $c_2(n)$  and  $c_3(n)$  change to 0 and later recover to their original values sequentially, the minimal RBFNN algorithm is able to track the varying system dynamics and prune/add hidden neurons approximately to keep the size of RBFNN minimal. Some of the hidden neurons generated during the first 1500 samples appear to be superfluous and the minimal RBFNN algorithm prunes these neurons and reduces the size of the network to contain five hidden neurons. Finally, when all the values of  $c_1(n)$ ,  $c_2(n)$  and  $c_3(n)$  reach

the original values, the number of hidden units in the minimal RBF network reaches seven again. The number of neurons assumed for Hybrid algorithm [11] is 30 which is highly in excess of the seven required in the minimal RBF algorithm. Even the mean square error for the hybrid algorithm is higher. The parameters used here are same as in the above example.



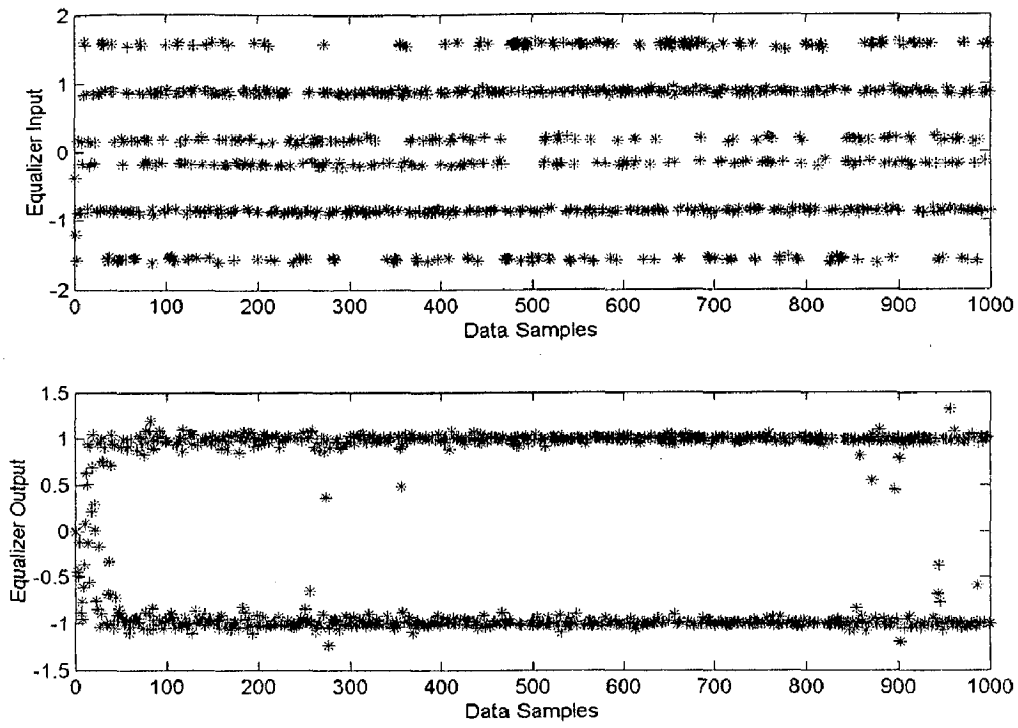
**Fig 5.4: Growth of Hidden Units as training progresses for Model 2**

### Model 3: (Non Linear Channel 1)

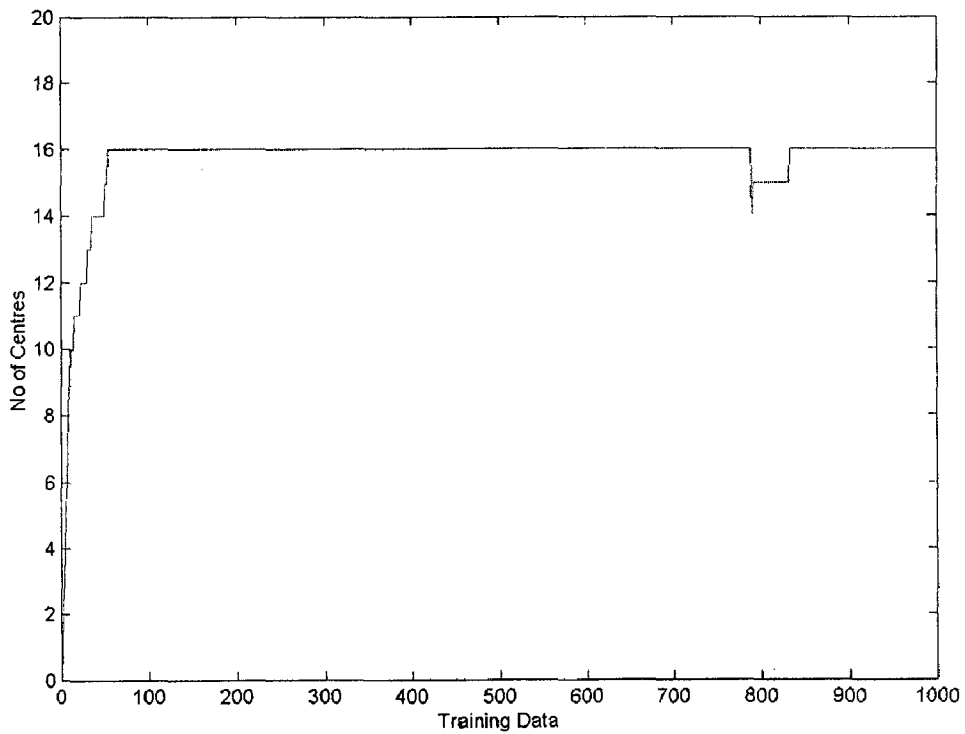
To test the algorithm for non-linear channels, the following non-linear channel was chosen:

$$H(z) = X(z)/S(z) = 0.3482 + 0.8704 z^{-1} + 0.3482z^{-2} \quad \text{-----} \quad 5.6$$

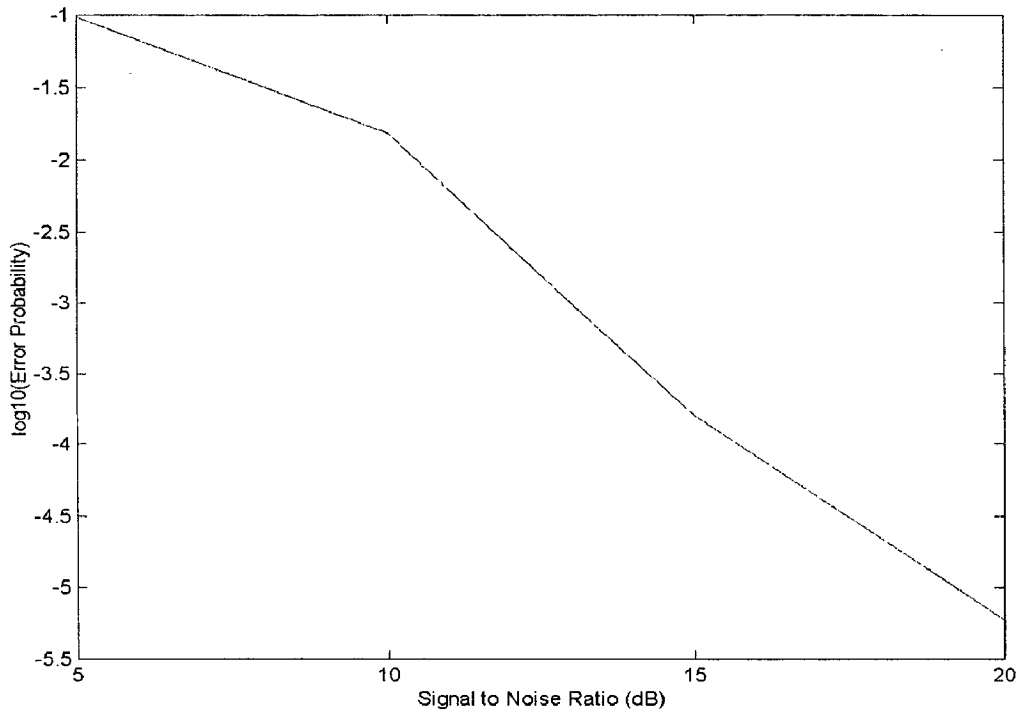
In the paper, Kechriotis et al [10] had used a recurrent neural network with multiple iterations to realize an equalizer network. For the purpose of graphical display, the equalizer order is chosen as  $m=2$ . Thus, a two dimensional plot can be made, to show the the most two recent input to the equalizer for each input data passed through the channel. In the example,  $n_h = 2$ . Thus, there will be 16 desired states for the channel output, ( $2^{n_h + m} = 16$ ). Here in MRAN algorithm, 1000 data samples at 25dB SNR, were used. The Equalizer input and output are shown in Fig 5.5. The network has build up 16 hidden nodes. Figure 5.6 shows the growth of RBF network, as training progresses. This is equal to the 16 desired channel states. Bit error rate (BER) is one method of testing to see if an equalizer is performing as required. The performance of the network as training progresses can be seen in Fig 5.7. The parameters required by the algorithm are selected as  $\epsilon_{\max} = 0.5$ ,  $\epsilon_{\min} = 0.2$ ,  $\gamma = 0.993$ ,  $e_{\min} = 0.1$ ,  $K = 0.85$ ,  $P_0 = R_n = 1.0$ ,  $Q = 0.002$ ,  $\delta = 0.0001$ ,  $e'_{\min} = 0.1$  and  $M = 30$ .



**Fig 5.5: Equalizer Input and Output for Model 3**



**Fig 5.6: Number of centres obtained as training progresses for Model3**



**Fig 5.7: Error Curves for RBF Networks for Model 3**

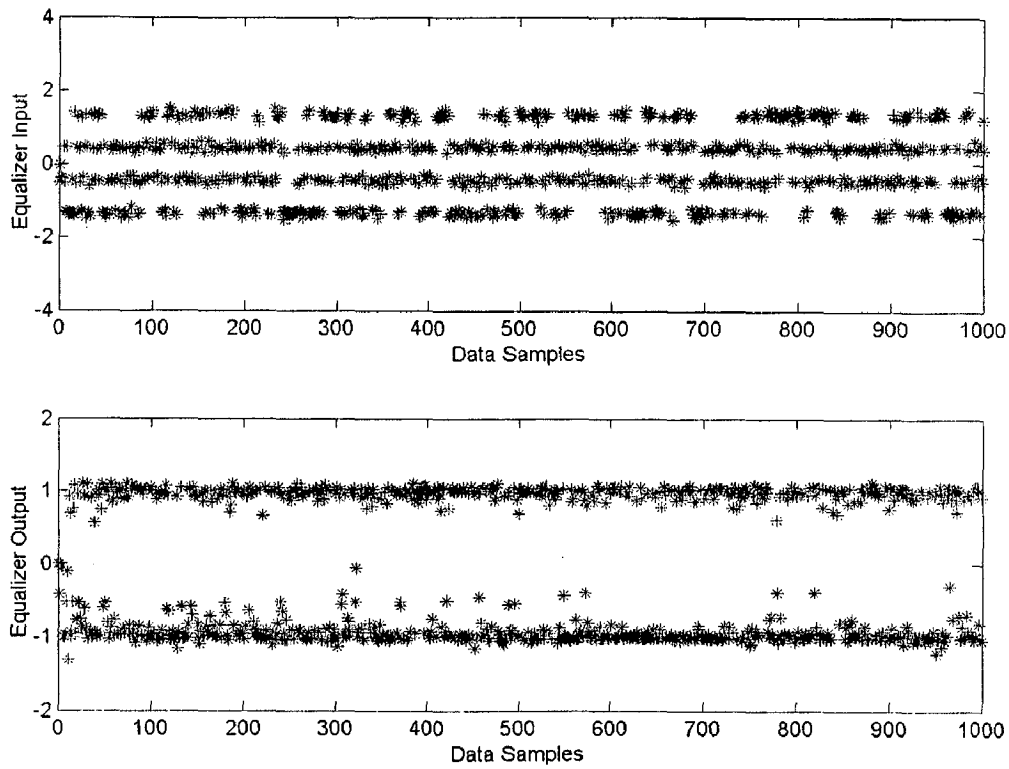
### Model 4: (Non Linear Channel 2)

A second Non Linear Channel with the following model was used:

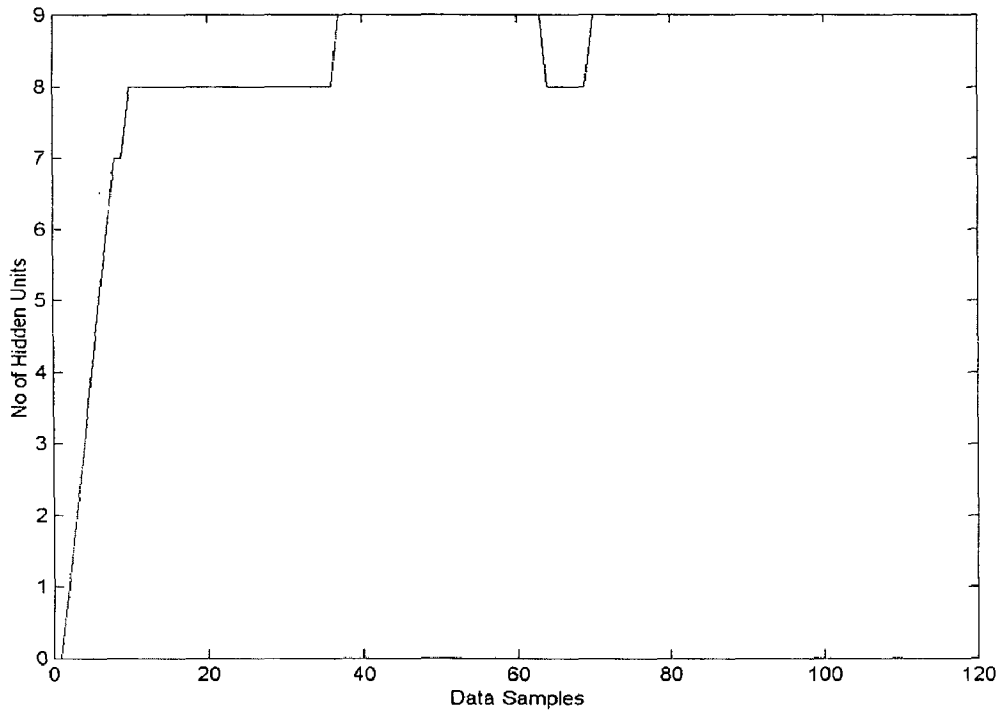
$$H(z) = X(z)/S(z) = 1 + 0.7 z^{-1} \quad \text{-----} \quad 5.7$$

Such channel models are frequently encountered in data transmission over digital satellite links, especially when the signal amplifiers operate in their high-gain limits. The equalizer order was chosen to be  $m=2$ . 1000 training data bits were used. They were mixed with low noise to get 30 db SNR. The parameters  $e_{\min}$  and  $e_{\min 1}$  were both set to 0.9. The other parameters were,  $\epsilon = 0.5$  and Sliding Window  $M = 10$ . The resulting network had 9 Hidden units, as compared with 8 desired channel states.

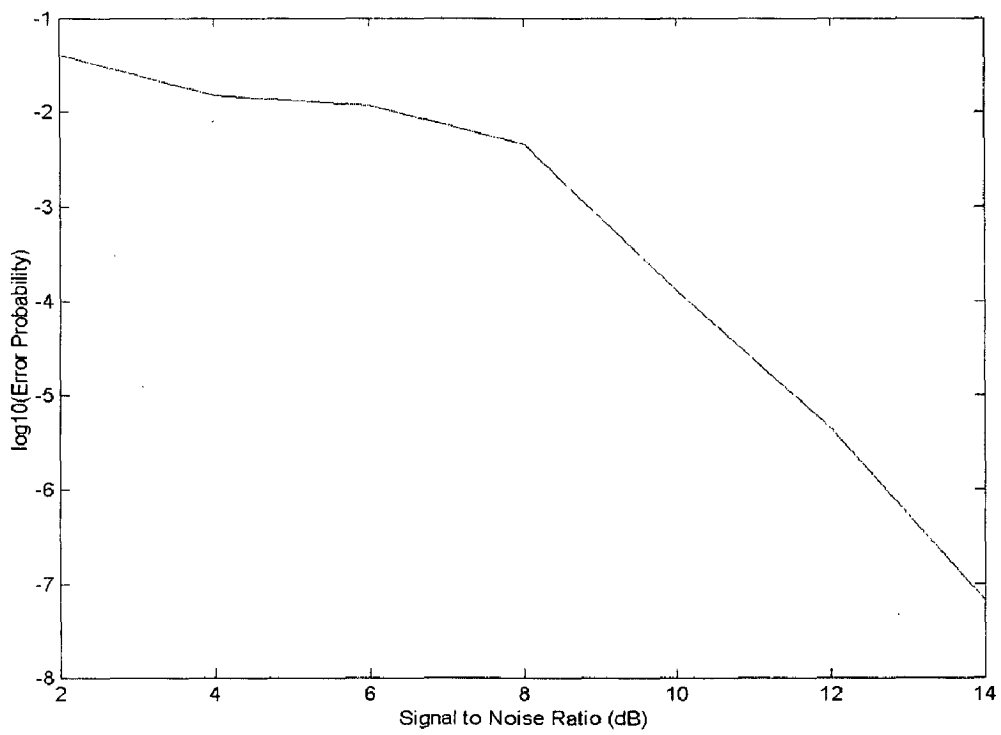
The Equalizer input and output are shown in Fig 5.8. Figure 5.9 shows the growth of RBF network, as training progresses. Fig 5.10 shows the variation of error probability curve as SNR changes.



**Fig 5.8: Equalizer Input and Output for Model 4**



**Fig 5.9: Number of Centres obtained as training progresses for Model 4**



**Fig 5.10: Error Curves for RBF Networks for Model 4**



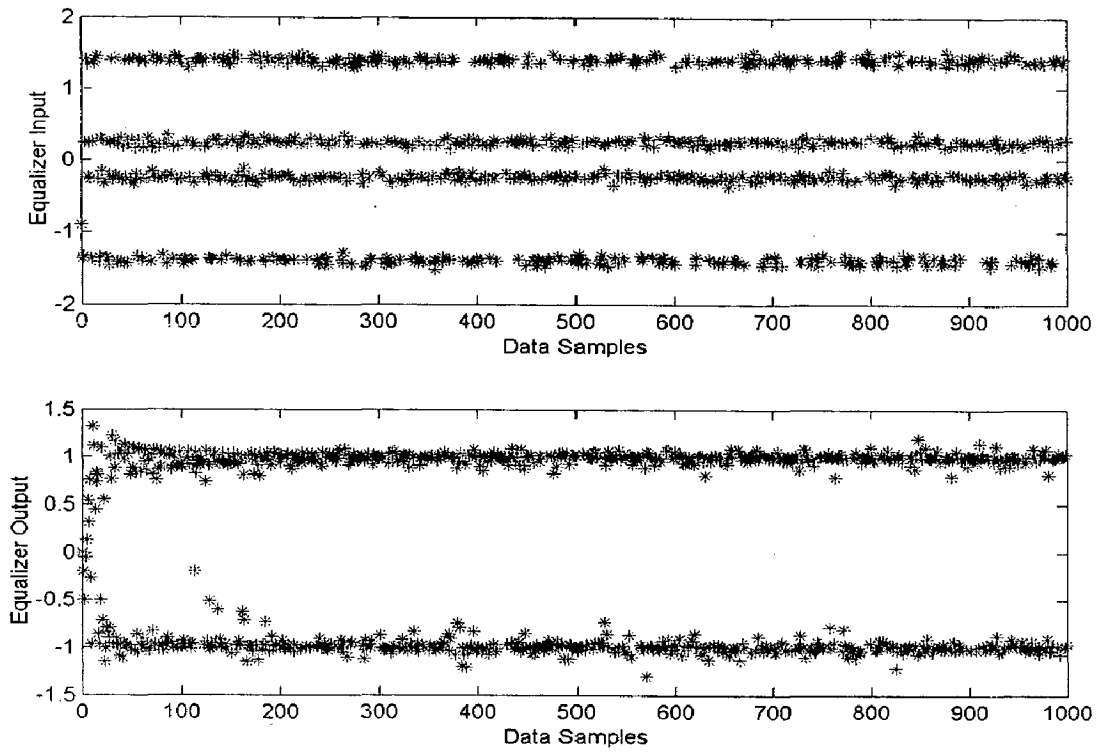
## Model 5: Linear Non-Minimum phase channel

Here a linear non-minimum phase channel was chosen with the following transfer function:

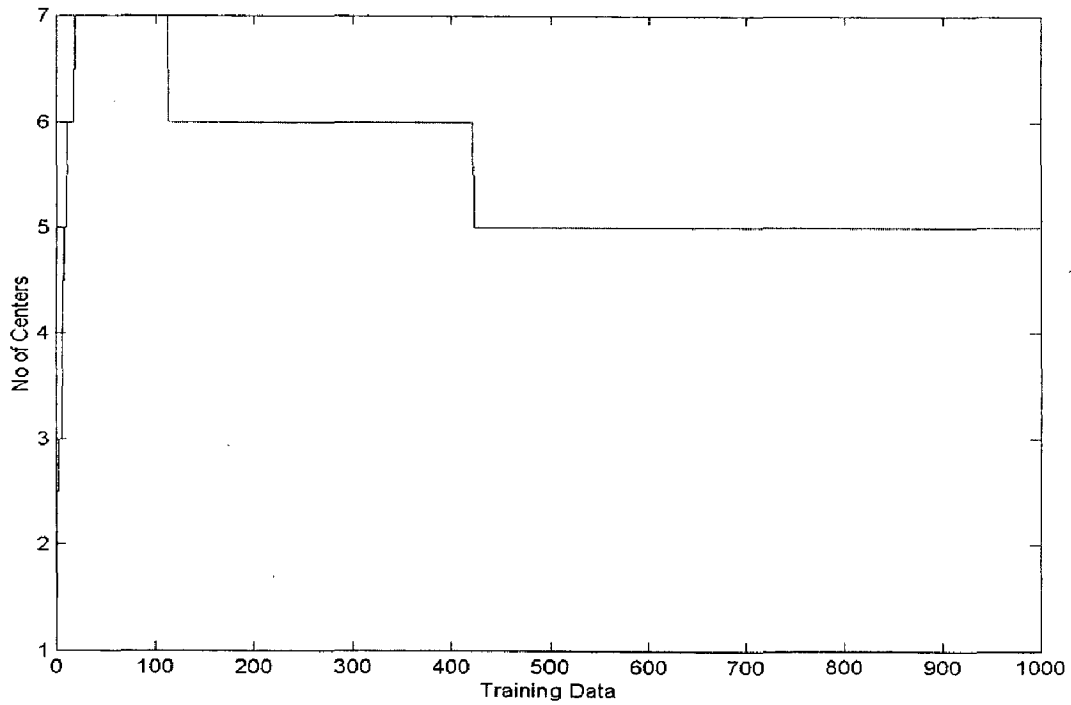
$$H(z) = X(z)/S(z) = 0.5 + 1.0 z^{-1} \quad \text{-----} \quad 5.8$$

This is the channel used by Chen et al [8] . In his paper a clustering algorithm for 1000 data samples of 10 dB SNR to obtain a RBF network was used. The method, however, had to have knowledge of which state was being transmitted, and the total number of states, so that the data belong to each state should be clustered together. In our method 1000 data samples of the data at 15dB SNR were used to train the network using the MRAN algorithm. There was no need for estimation of the channel order, to estimate the number of states. Neither was there a need to know which state the transmitted data belonged to.

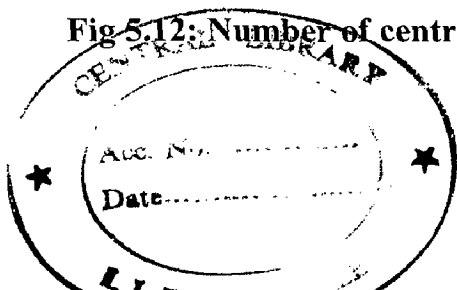
The parameters were  $e_{min} = 0.1$ ,  $e_{min1} = 0.3$ ,  $\epsilon_n = 0.5$ ,  $M=10$ . The results show that the MRAN algorithm is able to build up a network that can perform equalization comparable to that of a Bayesian equalizer, without having a need to estimate the channel order.



**Fig 5.11: Equalizer Input and Output for Model 5**



**Fig 5.12: Number of centres obtained as training progresses for Model 5**



### 5.3 Summary

Here from the above results, it is clearly shown that MRAN algorithm can be applied to several applications like function approximation, channel equalization, nonlinear system identification etc. In channel equalization, the advantage of using MRAN is firstly, there is no need to estimate the channel parameters or channel order. The second thing is, even though its performance is almost comparable to Bayesian equalizer, the number of hidden neurons required is far less than most of the previous methods. In nonlinear system identification, the number of hidden neurons required by MRAN is far less than the previous methods.



# CONCLUSION AND FUTURE WORK

---

In the proposed dissertation work, a sequential learning algorithm for implementing minimal Radial Basis Function (RBF) neural networks has been developed. This learning algorithm for the network referred to as MRAN, not only allocates a new hidden neuron based on the novelty of observation but also prunes those hidden neuron units which have insignificant contribution to the outputs of the RBF network. Therefore, when applied to time-varying dynamic systems, the architecture of MRAN will be adjusted automatically on-line to fit closely the dynamics of the observation data.

This dissertation work describes in detail the performance of MRAN on a number of applications. These applications consist of problems which are both static and dynamic systems. The static problems include channel equalization. Here in channel equalization different types of channels which are used practically are considered. The simulation result clearly shows the advantage of MRAN when compared with previous methods. Even the error performance is also almost comparable with bayesian equalizers. The problem for dynamic type consists of nonlinear dynamic system identification with both fixed and varying dynamics case. Here also, MRAN shows better performance with minimal structure when compared with previous methods.

### **Future work**

The following are the possible areas of future works which emerge from this work

- ❖ There are some threshold values which must be chosen before MRAN is used. The selection of these parameters is now based on experience and the selected parameter combination may not be optimal. The methods for choosing optimal threshold values of the MRAN algorithm also need to be developed:

- ❖ Extended kalman Filter (EKF) is employed in MRAN as the strategy for network parameter adaptation. EKF is demonstrated to improve the rate of convergence of networks. However, when the input and output dimensions of the network increase, the memories occupied by the covariance matrix  $P$  in EKF will increase doubly. Furthermore, the inverse of  $P$  is required during the adaptation of the network parameters, the adaptation of the network parameters, the computation speed will thus be slowed down due to the large dimensions of  $P$ . A faster implementation version of EKF should be used in MRAN. Real time implementation version of EKF should be used in MRAN. Real time implementation issues of MRAN also have to be addressed for practical use.

## REFERENCES

---

- [1] J.C.Patra , R.N Pal, R.Baliarsingh, and G.Panda " Nonlinear channel equalization for QAM constellation using artificial neural networks," *IEEE Trans.syst., Man, cybern. B*, Vol.29, pp.262-271, Apr.1999
- [2] S.Chen, S.Mclaughlin, and B.Mulgrew, "Complex-valued radial basis function network, Part II: Application to digital communications channel equalization," *Signal Processing*, Vol 36, pp.165-188, Feb.1994
- [3] L.Cha, and S.Kassam "Channel equalization using adaptive complex radial basis function networks," *IEEE J.Select.Areas.commun.*, Vol.13, pp.122-131, Jan.1995.
- [4] L.Yingwei, N.Sundararajan, and P.Saratchandran, "Performance evaluation of a sequential Minimal Radial Basis Function (RBF) neural network learning algorithm," *IEEE Trans. Neural Networks*, Vol.9, pp.308-318, Mar.1998.
- [5] D..Jianping, N.Sundararajan, P.Saratchandran, "Complex-valued minimal radial basis function neural network for nonlinear system processing" *Int. J. Neural Syst.*, Vol.10, No.2, pp.95-106, Jan 2000.
- [6] Proakis, J. G., *Digital Communications*. New York Mcgraw-Hill,1983.
- [7] S.Chen, G.J.Gibson, C.F.N. Cowan, and P.M.Grant, "Adaptive equalization of finite non-linear channels using multilayer perceptrons," *Signal Processing*, Vol.20 , pp 107-119 , 1990.

- [8] S.Chen, Mulgrew B., Grant P.M, "A Clustering Technique for Digital Communications Channel Equalization Using Radial Basis Function Networks", *IEEE Transactions on Neural Networks* ,Vol 4, No.4, July 1993.
- [9] Y.H. Cheng and C. Shin Lin, "A learning algorithm for radial basis function networks: with the capability of adding and pruning neurons," *IEEE Int. Conf. on Neural Networks* , Vol.1, No.3, pp. 797-801 , 1994.
- [10] G.Kechriotis, E.Zervas and E.S.Manolakos , "Using Recurrent Neural Networks for Adaptive Communication Channel Equalization ", *IEEE Transactions on Neural Networks*, Vol 5, No.2 ,March 1994.
- [11] Chen, S., Billings, S.A., and Grant, P.M., "Recursive Hybrid Algorithm for non-linear system identification using radial basis function networks" , *Int. J. control*, 1992, 55, pp. 1051-1070.
- [12] K.Tao, " A closer look at the radial basis function (rbf) networks," in *Conference Record of 27<sup>th</sup> Asilomar Conference on Signals, System and Computers*, (Pacific Grove, CA, USA) , pp. 401-405,1993.
- [13] A.G.Bors and M.Gabbouj, "Minimal topology for a radial basis functions neural network for pattern classification," *Digital Signal Processing*, Vol. 4, pp.173-188, 1994.
- [14] S.Haykin, *Neural Networks: A Comprehensive Foundation*. Second Edition, Prentice-Hall International Inc., New Jersey, 1999.
- [15] C.M.Bishop, *Neural Networks for Pattern Recognition*. Oxford; Clarendon Press, UK, 1995.
- [16] Aleksander, I. and Morton, H. *An introduction to neural computing* ( 2nd edition).



- [17] Qureshi, S.U.H., "Adaptive Equalization", *Proc. IEEE*, Vol.73, No.9, pp. 1349-1387, Sept. 1985.
- [18] Tou, J.T., Gongalez, R.C.(1974) *Pattern Recognition*. Reading, MA:Addison-Wesley.
- [19] Park, J., Sandberg, J.W.,(1991), "Universal approximation using radial basis functions network," *Neural Computation*, Vol 3, pp .246-257.
- [20] Poggio, T., Girosi, F.,(1990) "Networks for approximation and learning," *Proc. IEEE*, Vol.78, No.9, pp.1481-1497.
- [21] Moody, J.,(1989) "Fast learning in networks of locally-tuned processing units," *Neural Computation* , Vol. 1 , pp.281-194.
- [22] Broomhead, D.S., Lowe, D.(1988) "Multivariable functional interpolation and adaptive networks," *Complex Systems*, Vol.2, pp.331-355.
- [23] Sanner, R.M., Slotine, J.E., (1994) "Gaussian networks for direct adaptive control," *IEEE Trans. On Neural Networks*, Vol.3, No.6, pp.837-863.
- [24] Bors, A.G., Gabbouj, G.,(1994) "Minimal topology for a radial basis function neural networks for pattern classification," *Digital Signal Processing: a review journal*, Vol.4, No:3, pp.173-188.
- [25] Chen, S., Cowan, C.F.N., Grant, P.M., (1991) "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Trans. On Neural Networks*, Vol.2, No.2, pp.302-309.

- [26] P. Chandra Kumar, "Radial Basis function networks for communication channel equalization", MEng Thesis, School of Electrical and Electronic Eng., Nanyang Technological University, Singapore, Nov 1998.
  
- [27] V. Kadiramanathan and M. Nirajan, "Application of an architecturally dynamic network for speech pattern classification," *Proceedings of the Institute of Acoustics*, Vol.14, pp.343-350,1992.
  
- [28] C. Chen, W. Chen, and F. Cheng, "Hybrid learning algorithm for Gaussian potential function network," *IEEE Proceedings-D*, Vol.140, pp.442-448, 1993.