# Fast Orthogonal Search For Training Radial Basis Function Neural Networks

By Wahid Ahmed

Thesis Advisor: Donald M. Hummels, Ph.D.

An Abstract of the Thesis Presented in
Partial Fulfillment of the Requirements for the
Degree of Master of Science (in Electrical Engineering).
August, 1994

This thesis presents a fast orthogonalization process to train a Radial Basis Function (RBF) neural network. The traditional methods for configuring the RBF weights is to use some matrix inversion or iterative process. These traditional approaches are either time consuming or computationally expensive, and may not converge to a solution. The goal of this thesis is first to use a fast orthogonalization process to find the nodes of the RBF network which produce the most improvement on a target function, and then to find the weights for these nodes. Three applications of RBF networks using this fast orthogonal search technique are presented. The first problem involves characterization of an analog-to-digital converter (ADC), where the goal is to build an error table to compensate the error generated by the physical conversion process of ADC. The second problem is a simple pattern recognition problem, where the goal is to classify a set of 2 dimensional and 2-class patterns. The final problem involves classification of human chromosomes, which is a highly complicated 30 dimensional and 24 class problem. Experimental results will be presented to show that the fast orthogonal search technique not only outperforms traditional techniques, but it also uses much less time and effort.

# Fast Orthogonal Search For Training Radial Basis Function Neural Networks

By

Wahid Ahmed

A THESIS

Submitted in Partial Fulfillment of the

requirement for the Degree of

Master of Science

(in Electrical Engineering)

The Graduate School

University of Maine

August, 1994

Advisory Committee:

Donald M. Hummels, Associate Professor of Electrical and Computer Engineering
Thesis Advisor

Mohamad T. Musavi, Associate Professor of Electrical and Computer Engineering,

Fred H. Irons, Castle Professor of Electrical and Computer Engineering,

Bruce E. Segee, Assistant Professor of Electrical and Computer Engineering.

# ACKNOWLEDGMENTS

Abundant thanks are due to those who, in one respect or another, have contributed to this project.

First, I am greatly indebted to my family for their continued support and encouragement. Without them I couldn't make it this far.

I especially wish to express my sincere gratitude to my thesis advisor, Dr. Donald M. Hummels for his patience, guidance, encouragement, and time. Without him this thesis would not have been completed. I envy his sincere, enthusiastic approach.

I express my sincere appreciation to Dr. M. T. Musavi. I owe him everything I know in neural networks. He gave me the first lesson on neural networks, and he has continued to support my neural network research until now. I have learned a lot just by working with him. I admire his unstoppable desire for improvement in every aspect. Lots of thanks goes to him.

My sincere appreciation goes to Dr. Fred H. Irons for his inspirational discussions, for his help in understanding the analog-to-digital converters, and for providing constructive criticism for this thesis.

I am grateful to Dr. Bruce E. Segee, for his constructive participation in my advisory committee.

I would like to thank everybody I worked with here in the Electrical and Computer Engineering Department. Thanks goes to, Bruce Littlefield for sharing his expertise with me in networking, Ioannis Papantonopoulos and Richard Cook for ADC research, and my coworkers this summer Norm Dutil, Jon Larabee for the greatest working environment and last but not least, Sekhar Puranapanda for a lot of laughs.

Finally, I wish to thank all my friends and faculty for making such a warm environment during my stay at the University of Maine.

# CONTENTS

# LIST OF FIGURES

# CHAPTER 1

# Introduction

## 1.1 Problem Statement

The neural network is the most exciting interdisciplinary field of this decade. The widespread applications of neural networks not only proves their success but also illustrates the necessity of finding the best network to solve a problem. It seems like the general agreement is that no single network can solve all our problems; we need different networks to solve different problems. Back Propagation (BP) [1], Radial Basis Function (RBF) [2, 3], Probabilistic Neural Network (PNN) [4], Kohonen's Self Organizing Map [5] are namely the few well known neural networks.

The growth of neural networks has been heavily influenced by the Radial Basis Function (RBF) neural networks. The application of the RBF network can be found in pattern recognition [3, 6, 7], function approximation [2, 8], signal processing [8, 9], system equalization [11] and more. The consistency and convergence of the RBF network for the approximation of functions has been proven [10]. A large amount of research has been also conducted on the architecture of the RBF network. The two most important parameters of a RBF node, the center and the covariance matrix, have been examined thoroughly [6, 11, 12]. A major issue handled by these researchers is the reduction of the number of nodes. This reduction involves clustering of the input samples without any consideration of the target function, or the convergence of the weights. The weights (the most significant component of any neural network) of the RBF network were left untouched by most of the researchers. This oversight is not ignorance but a confidence on the traditional approaches. Like many other things, the traditional approaches just do not work all the time. For RBF weights, the traditional approaches only work when the training samples are well behaved. In real life, the training samples

are not well behaved causing major problems for finding the RBF weights. The issue of this thesis is to find a set of most significant nodes and their weights for a given network, using a technique that considers both the structure of the input parameter space and the target function to which the network will be trained.

## 1.2 Review of Previous Work

The traditional approach to design an RBF network is to first select a set of network parameters (number of nodes, node centers, node covariances) and then find the weights by formulating the network by solving a least squares (LS) formulations of the problem. Various techniques may be used to solve the LS problems including Singular Value Decomposition (SVD), Cholesky Decomposition, and the Gradient Descent approach. These traditional methods have a variety of problems, which will be discussed later. Orthogonal decomposition techniques may be used to provide an orthogonal basis set for a LS problem. An orthogonal scheme was used by Chen et. al. [13] in RBF networks to simultaneously configure the structure of the network and the weights. The orthogonal search technique presented by Chen is cumbersome, and requires redundant calculations making it non-suitable for reasonable size networks.

A similar fast orthogonal search technique has also been developed by Korenberg et. al. [14, 15] for nonlinear system identification. This procedure also includes redundant calculations and was highly customized to the problem of finding the kernels for a nonlinear system with random inputs. This thesis presents an efficient fast orthogonal search eliminating the redundancy of [13, 14]. The resulting algorithm is directly applicable to RBF networks and a wide variety of other LS approximation problems.

## 1.3 Thesis Overview

This thesis has been organized in five chapters. Chapter 2 provides the RBF architecture along with some traditional approaches to configure the weights. In Chapter 3 the mathematical development, and a simple algorithm for the proposed fast orthogonal search technique will be presented. Several applications of the RBF network, using the fast orthogonal search technique, will be used to evaluate the technique in Chapter 4. Chapter 5 concludes the thesis by providing a brief summary of this research and suggesting some future research directions.

# CHAPTER 2

# Radial Basis Function Neural Network

## 2.1 RBF structure

The RBF Neural Network gained its popularity for its simplicity and speed. RBF is a simple feed forward neural network with only one hidden layer, and an output layer. The hidden layer is the backbone of the RBF structure. The hidden layer consists of a set of neurons or nodes with radial basis functions as the activation function of the neuron, hence the name Radial Basis Function Neural Network. A Gaussian density function is the most widely used activation function. The output layer is simply a summing unit. This layer adds up all of the weighted output of the hidden layer. Figure 2.1 illustrates the RBF network.

The following equation gives the output of the RBF network

$$\hat{y} = f(\vec{x}) = \sum_{k=1}^{N} w_k \phi_k(\vec{x}), \tag{2.1}$$

where

$$\phi_k(\vec{x}) = (2\pi)^{-p/2} |\Sigma_k|^{-\frac{1}{2}} e^{-\frac{1}{2}(\vec{x}-\vec{c}_k)\Sigma_k{}^{-1}(\vec{x}-\vec{c}_k)^T}. \tag{2.2}$$

Above, $N$ is the number of network nodes, p is the dimensionality of the input space $\vec{x}$, and $w_k$, $\vec{c}_k$, and $\Sigma_k$ represent the weight, center, and the covariance matrix associated with each node. In the above equation the output of the network is a scalar quantity for simplicity, but the network can have any number of outputs.

In supervised learning, where input output pairs $(\vec{x}, y)$ are presented to "teach" the network, the objective of training is to configure a set of weights, $w_k$, such that the network produces the desired output for the given inputs. In that case, we say that the network has learned. So if $(\vec{x}, y)$ is an input output pair, where $\vec{x}$ is the input and $y$ is the desired output, then the network should learn the

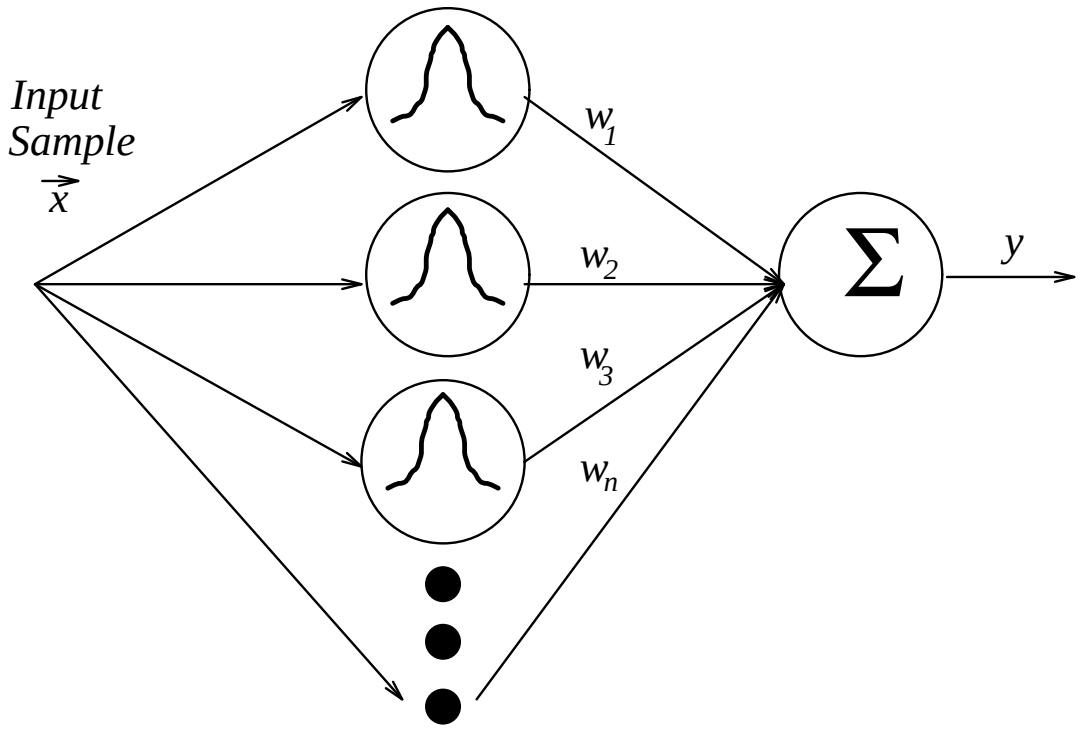


Figure 2.1: Network structure for the Radial Basis Function Neural Network.

mapping function $f$, where $y = f(\vec{x})$. The training is done using the $M$ training sample pairs $(\vec{x}_1, y_1), (\vec{x}_2, y_2), \cdots (\vec{x}_M, y_M)$. The output vector containing the $M$ outputs of the network can be written using the following matrix form,

$$\hat{\vec{y}} = \Phi \vec{w}, \tag{2.3}$$

where $\hat{\vec{y}}$ is an $M$ dimensional vector and $\vec{w}$ is the $N$ dimensional weight vector. The above quantities aregiven by

$$\begin{aligned}
\hat{\vec{y}} &= \begin{bmatrix} \hat{y}_1 & \hat{y}_2 & \cdots & \hat{y}_M \end{bmatrix}^T, \\
\vec{w} &= \begin{bmatrix} w_1 & w_2 & \cdots & w_N \end{bmatrix}^T, \\
\Phi &= \begin{bmatrix} \phi_1(\vec{x}_1) & \phi_2(\vec{x}_1) & \cdots & \phi_N(\vec{x}_1) \\ \vdots & \vdots & & \vdots \\ \phi_1(\vec{x}_M) & \phi_2(\vec{x}_M) & \cdots & \phi_N(\vec{x}_M) \end{bmatrix}.
\end{aligned} \tag{2.4}$$

$\phi_i(\vec{x}_j)$ is the output of $i^{th}$ node for the $j^{th}$ input vector $\vec{x}_j$. So each column of the $\Phi$ matrix contains the output of a node for all $M$ training samples.

The problem of finding the network weights reduces to finding the vector $\vec{w}$ which makes the network output $\hat{\vec{y}}$ as close as possible to the vector of desired network outputs $\vec{y} = [y_1 \ y_2 \ \cdots y_M]^T$. Generally, $\vec{w}$ is determined by finding the least square (LS) solution to

$$\Phi \vec{w} = \vec{y}. \tag{2.5}$$

The method for finding the solution to (2.5) depends in large part on the structure of the network being designed. One popular scheme is to center a node of the network on each of the input training samples $(\vec{c}_k = \vec{x}_k, k = 1, 2, \cdots, N)$. In this case the matrix $\Phi$ is square, and the weights are given by $\vec{w} = \Phi^{-1}\vec{y}$ provided that $\Phi$ is nonsingular. However, calculation of $\Phi^{-1}$ is often problematic, particularly for large networks.

Often, the number of nodes is much less than the number of training samples. In this case the system of equations (2.5) is overdetermined, and no exact solution exists. Various alternative methods of finding the weights in this case are discussed in the following sections.

## 2.2 Solving for the Weights

The training phase of any neural network is the most significant job for designing the network. The network must "learn" as accurately as possible and at the same time it should be able to generalize from the learned event. Training of a RBF neural networks involves setting up the node centers, the variance or widths, and most importantly the weight of each node. There has been extensive work on how to select the node centers and covariance [2, 3, 4, 6, 16, 17]. We assume in this section that these values have been selected, and that we must find an appropriate selection for $\vec{w}$. A number of traditional schemes for this selection are reviewed in this section, along with a brief summary of advantages and disadvantages.

### 2.2.1 Gradient Descent Solution

The first approach in this thesis to solve the weights is the Gradient Descent solution. This approach gained its popularity due to its similarity with algorithms used to train the Back Propagation (BP) neural networks [1, 18, 19]. The solution is found by feeding back the output of the network to update the weights. This updating continues until the network output meets some error criteria. The error criterion most widely used is the sum squared error (SSE) or the mean squared error (MSE) between the network outputs and the desired output. The following equation describes the weight updating rule.

$$w_k^{new} = w_k^{old} + \alpha \Delta w_k, \tag{2.6}$$

where $\alpha$ is a constant-known as the "learning rate"-between 0 and 1, and

$$\Delta w_k = \phi_k(\vec{x}_i)(y_i - \hat{y}_i). \tag{2.7}$$

In matrix form the updating rule is:

$$\vec{w}^{new} = \vec{w}^{old} + \alpha\Delta\vec{w}, \tag{2.8}$$

where

$$\Delta\vec{w} = \Phi^T(\vec{y} - \hat{\vec{y}}). \tag{2.9}$$

This rule is also widely known as the "delta rule."

An advantage of this method is that it can be used without storing the complete $\Phi$ matrix. For a large network where memory to store $\Phi$ can be an issue, this technique can be used easily by generating only one row of the $\Phi$ matrix at a time.

A major disadvantage of this approach is that it's very slow since an iterative process is involved. The solution may not even converge. The initial weights are also a major factor, and for many cases there may be no procedure for correctly choosing a set of initial weights that will provide a solution. If the $\Phi$ matrix is singular, which is always possible if the network is large, the procedure may not converge at all, or converge to a erroneous solution. The singularity of the $\Phi$ matrix occurs when the selection of nodes are either separated too closely or the variances are too large. So choosing the nodes, their variance, and the initial weights becomes a major problem.

### 2.2.2 Cholesky Decomposition

Cholesky Decomposition or factorization is also a widely used technique for finding the least squares solution to a system of linear equations. The vector $\vec{w}$ which gives the least squares solution to (2.5) must satisfy the "normal equation"

which is given by

$$\Phi^T \vec{y} = \Phi^T \Phi \vec{w}. \tag{2.10}$$

The symmetric matrix $(\Phi^T \Phi)$ is commonly refered to as the Gram matrix. The symmetry of the Gram matrix may be exploited to give a numerically efficient technique for solving (2.10) without explicitly calculating $(\Phi^T \Phi)^{-1}$. Cholesky factorization accomplishes this by decomposing the Gram matrix. The decomposed form of the Gram matrix is

$$\Phi^T \Phi = U^T U, \tag{2.11}$$

provided that the Gram matrix is nonsingular. Here U is a right triangular or upper triangular matrix [20, 22]. Now by substituting equation (2.11) into equation (2.10), we obtain,

$$U^T U \vec{w} = \Phi^T \hat{\vec{y}}, \tag{2.12}$$

The above equation can be written as,

$$U^T \vec{z} = \Phi^T \hat{\vec{y}}, \tag{2.13}$$

where

$$\vec{z} = U \vec{w}. \tag{2.14}$$

Equation (2.13) can be easily solved by the method of forward substitution since $U^T$ is a lower triangular matrix, and (2.14) can be easily solved by the method of back substitution since $U$ is an upper triangular matrix [20]. Algorithms for finding the Cholesky Decomposition are given in [23].

The major advantage of this technique is that it is very fast compared to solving by inverting the Gram matrix and then doing several matrix multiplications. However, this technique has several significant drawbacks.

The first assumption for the Cholesky Decomposition is that the Gram matrix is nonsingular, in other words $\det(\Phi^T \Phi) \neq 0$. If the number of nodes are

large the Gram matrix could be very close to being singular. The singularity of the Gram matrix introduces instability for the weights, and may result in approximations that do not generalize well to inputs near the training samples. When the weights for the RBF neural network are selected in this manner, the assignment of the node centers and the covariance matrices becomes crucial, since these parameters determine the structure of the $\Phi$ matrix.

A second problem is the complexity of the resulting network. The resulting network still has the same structure and size as the original ($\Phi$) network; that is all the nodes of the network are used. This procedure does not eliminate the redundant nodes from the network making the resulting network computationally inefficient. One would prefer a procedure to select only nodes which contribute to the approximation.

### 2.2.3 Singular Value Decomposition

The Singular Value Decomposition (SVD) provides a method of alleviating the numerical instability which exists for nearly singular matrices. For any $M \times N$ matrix, $\Phi$, the SVD is given by

$$\Phi = U \Lambda V^T, \tag{2.15}$$

where $U$ is an $M \times N$ matrix, $\Lambda$ is a $N \times N$ diagonal matrix, $V$ is an $N \times N$ matrix, and $U$ and $V$ are orthonormal ($U^T U = I, V^T V = I$). The columns of $U$ are eigenvectors of ($\Phi \Phi^T$), and are called the left singular vector of $\Phi$. An important property of the SVD is that the matrix $U$ provides an orthonormal basis set for the $\Phi$ matrix. The columns of $V$ are the eigenvectors of the Gram matrix ($\Phi^T \Phi$), and called the right singular vectors. $\Lambda$ is a diagonal matrix, where the diagonal elements are the square root of the eigenvalues of the Gram matrix ($\Phi^T \Phi$). These diagonal elements are called the singular values.

By substituting the SVD of $\Phi$, the equation (2.10) can be rewritten as

$$V\Lambda U^T U\Lambda V^T \vec{w} = V\Lambda U^T \vec{y}. \tag{2.16}$$

By using the fact that $U^T U = V^T V = I$, the solution for the weights is

$$\vec{w} = V\Lambda^{-1} U^T \vec{y} = \sum_{i=1}^{N} \vec{v_i} \left( \frac{\vec{u_i}^T \vec{y}}{\lambda_i} \right); \qquad \lambda_i \neq 0. \tag{2.17}$$

$\vec{u_i}$ and $\vec{v_i}$ are the $i^{th}$ column vector of the $U$ and $V$ matrix respectively, and $\lambda_i$ is the $i^{th}$ singular value (diagonal element of $\Lambda$). The term $V\Lambda^{-1}U^T$ is most commonly denoted by $\Phi^{\#}$, and called the pseudo inverse of $\Phi$. Note that in (2.17) the summation is carried out only over non-zero singular values. If the original $\Phi$ matrix is singular, one or more of the singular values will be zero. In this case (2.5) does not have a unique solution - there are an infinite number of solutions $\vec{w}$ which will provide the same (minimum) mean-square error. By carrying out the sum only over non-zero singular values, we are adopting the solution for $\vec{w}$ which has minimum length [23].

The singular values of a matrix are used to determine its proximity to singularity. Equation (2.17) shows how a nearly singular matrix (small $\lambda_i$'s) can make the solution for $\vec{w}$ blow up. One method of improving the numerical stability of the solution is by setting a threshold on the singular values. One can then throw away the components of the solution contributed by low singular values, ensuring that the weights do not blow up. The matrix which is obtained by setting the smallest values to zero may be shown to be the best low rank approximation to the original $\Phi$ matrix [20]. By dropping the smallest singular values from one solution, we are replacing an exact (but unstable) solution to the least squares problem by an approximate (but well-behaved) solution.

Although the SVD provides the best low-rank least square approximation for the weights, there exists some disadvantages. The following provides a list of

disadvantages of the SVD procedure.

1. SVD is slow and computationally expensive. It requires computing eigenvalues and eigenvectors for large matrices.
2. The threshold for elimination of singular values is hard to set. There are no clear approaches to select the threshold.
3. The approximation which is being made depends only on the structure of the network ($\Phi$) and is completely independent of the vector of training target values $\vec{y}$. One would expect an improved procedure could be developed which takes into account the function to be approximated.
4. Although the solution is obtained by approximating $\Phi$ by a singular matrix (implying fewer than $N$ nodes) the vector $\vec{w}$ generally does not contain any zero elements. Hence, all $N$ nodes of the network must be implemented. The network complexity is virtually identical to that obtained using the Cholesky Decomposition or gradient descent methods.

### 2.2.4 Orthogonal Search

Another technique for solving the least squares problem is the orthogonal search technique. The main part of the orthogonal search technique involves decomposing the $\Phi$ matrix to an orthogonal basis set, and then using the decomposition to solve the normal equation (2.10) to find the weights. The decomposition of $\Phi$ is given by

$$\Phi = QR. \tag{2.18}$$

$Q$ is a $M \times N$ matrix with column vectors given by the orthogonal basis vectors of $\Phi$, and $R$ is an $N \times N$ right or upper triangular matrix. The orthogonalization of $\Phi$ can be found by using the Householder transformation [20, 24], or the Gram-Schimidt orthogonalization procedure [13, 20, 21, 25].

The weights can be found by substituting (2.18) into equation (2.10), and using the fact that $Q^T Q = I$ we get

$$R^T R\vec{w} = \Phi^T \vec{y}. \tag{2.19}$$

The above equation can be solved for the weight vector by the method of forward substitution, and backward substitution [20] as described in section 2.2.2.

This procedure does not work when $\Phi$ is singular, and it does not give any insight about the network structure. A more useful approach is to use the orthogonal basis vectors to choose a set of nodes which reduces some error criteria for solving the least squares problem. Chen [13] presents one such algorithm, derived from the Gram-Schimidt procedure, to select the most 'significant' nodes one by one. In [14] a similar algorithm was also presented to select basis function one by one, with emphasis on the characterization of nonlinear systems with random inputs. Korenberg et. al. [14] presents an improved, fast version of the orthogonal search technique described in [26].

During each step of Chen's algorithm [13], the following procedure is used to search a set of candidate nodes to determine which node will be added to the currently selected set.

1. For each candidate, find the component of the basis vector associated with that node which is orthogonal to all of the currently selected nodes. Either the Gram-Schimidt or the Householder technique may be used. Call this component $\vec{q_i}$.

2. For each candidate, evaluate the projection of the target vector $\vec{y}$ onto the unit vector in the direction of the orthogonal component

$$p_i = \vec{y}^T \frac{\vec{q_i}}{\|\vec{q_i}\|}. \tag{2.20}$$

   This component gives the length of the change in $\vec{y}$ which will result if the $i^{th}$ vector is added to the currently selected set.

3. Choose the node which gives the largest value of $p_i$ - this is the node which will provide the greatest reduction in the mean-square error.

The importance of choosing the nodes one by one is significant in several aspects. The following is a list of some advantages.

1. Selection of nodes one by one provides an insight to the approximation problem, and the network structure. This insight can be used to further modify the network.
2. This selection procedure will let us meet some physical limitations. Nodes mean connections, so a reduction of nodes will provide a corresponding reduction of connections which can be very useful for hardware applications.
3. This procedure does not involve the selection of an arbitrary thresholding like the SVD method which can affect our solution. The desired number of nodes can be easily chosen just by looking at the error behavior.

The orthogonal search method is very useful, but not so practical. The method has not been widely accepted primarily because of the computational complexity. The procedure is not generally practical for networks of reasonable size. However, the above algorithm may be shown to be extremely redundant making it computationally expensive. In chapter 3 an algorithm will be presented to implement the orthogonal search technique efficiently. The algorithm will perform the same orthogonalization procedure without explicitly calculating the orthogonal set making it much easier to implement.

# CHAPTER 3

# The Fast Orthogonal Search

In chapter 2 some traditional approaches to solve the least square problem were discussed. The techniques discussed have a variety of problems for finding the weights for a RBF network. A major issue that had been left untouched is that of how many nodes are really needed to train a RBF network and what is the justification. Several clustering algorithms [6, 7, 12, 27] have been developed which could reduce the number of nodes, but these clustering algorithms by no means suggest whether the network will converge to a stable set of weights or not. In this chapter a simple, fast algorithm will be developed to find a set of weights that are best for the given network, and this solution of weights will indeed suggest the number of nodes needed by the network. The procedure may be shown to be a computationally efficient procedure for implementing the orthogonal search technique of section 2.2.4. Similar fast orthogonal search techniques have been used by Korenberg [14, 15] for time-series analysis, system identification, and signal identification problems.

In section 3.1 the fast orthogonal search technique will be developed mathematically. In section 3.2 a simple algorithm will be presented to implement the technique. A summary of computational requirements will be given in section 3.3.

## 3.1 Mathematical Development

The problem of solving for the RBF weights from the equation (2.3) is the issue of this section. Like the orthogonal search technique, during each iteration of the algorithm a set of candidate nodes will be considered to identify which node will provide the best improvement to the approximation of $\vec{y}$. This node will be added to the network, and the procedure continues until either an error criterion

is met or the number of nodes in the network reaches a desired value. Unlike the orthogonal search technique, the orthogonal basis set associated with the selected set of nodes is never explicitly calculated, significantly reducing the computational burden of the procedure. The Cholesky decomposition discussed in section 2.2.2 is the backbone of the proposed fast orthogonal search. Let's define $\vec{\phi}_j$ as the $M$ element column vector representing the output of the $j^{th}$ node for $M$ number of samples. If the job is to consider each node individually, let's also define $\left\{\vec{\phi}_j\right\}$ as the set of node functions to be considered, and let $\Phi$ denote the matrix containing the $\vec{\phi}_j$ for the currently selected nodes. Let $\Phi^T\Phi$ have Cholesky decomposition $\Phi^T\Phi = U^TU$. If a candidate function $\vec{\phi}_j \in \left\{\vec{\phi}_j\right\}$ is added to the $\Phi$ matrix, the Cholesky decomposition will be modified by

$$\begin{bmatrix} \Phi^T \\ \vec{\phi}_j^T \end{bmatrix} \begin{bmatrix} \Phi & \vec{\phi}_j \end{bmatrix} = \begin{bmatrix} U^T & 0 \\ \vec{x}_j^T & \xi_j \end{bmatrix} \begin{bmatrix} U & \vec{x}_j \\ 0 & \xi_j \end{bmatrix}, \tag{3.1}$$

where $\begin{bmatrix} \vec{x}_j \\ \xi_j \end{bmatrix}$ denotes the new column of the $U$ matrix. For the time being, assume that $\vec{x}_j$ and $\xi_j$ are known for each $\vec{\phi}_j \in \left\{\vec{\phi}_j\right\}$. (We must still show how $\vec{x}_j$ and $\xi_j$ should be updated as the $\Phi$ matrix changes.)

To select the appropriate $\vec{\phi}_j$, we need to look at how the selection of a node changes the estimate of $\vec{y}$, $\hat{\vec{y}}$. We prefer to select nodes which produce large changes in $\hat{\vec{y}}$. For the modified $\Phi$ of the above equation, the normal equation of the form (2.10) can be written as

$$\begin{bmatrix} \Phi^T \\ \vec{\phi}_j^T \end{bmatrix} \begin{bmatrix} \Phi & \vec{\phi}_j \end{bmatrix} \vec{w}_j = \begin{bmatrix} \Phi^T \\ \vec{\phi}_j^T \end{bmatrix} \vec{y}. \tag{3.2}$$

By using the decomposition of equation (3.1) in the above equation, we obtain

$$\begin{bmatrix} U^T & 0 \\ \vec{x}_j^T & \xi_j \end{bmatrix} \begin{bmatrix} U & \vec{x}_j \\ 0 & \xi_j \end{bmatrix} \vec{w}_j = \begin{bmatrix} \Phi^T \vec{y} \\ \vec{\phi}_j^T \vec{y} \end{bmatrix}. \tag{3.3}$$

As before in Cholesky decomposition equation (3.3) can be solved for $\vec{z}_j$ by the method of forward substitution where $\vec{z}_j$ is

$$\vec{z}_j = \begin{bmatrix} U & \vec{x}_j \\ 0 & \xi_j \end{bmatrix} \vec{w}_j. \tag{3.4}$$

Applying forward substitution in equation (3.3), we obtain

$$\vec{z}_j = \begin{bmatrix} \vec{z}_0 \\ \frac{1}{\xi_j}(\vec{\phi}_j^T y - \vec{x}_j^T \vec{z}_0) \end{bmatrix}, \tag{3.5}$$

where $\vec{z}_0$ (from $U^T \vec{z}_0 = \Phi^T \vec{y}$) is the old solution for $\vec{z}$ and is the same for all the $\vec{\phi}_j$ candidates. So the only change in $\vec{z}_j$ due to adding $\vec{\phi}_j$ is the addition of the the last component of the vector. Notice that this term is a scalar quantity and for simplicity let's define

$$\alpha_j = \vec{\phi}_j^T y - \vec{x}_j^T \vec{z}_0. \tag{3.6}$$

Now the estimate for the weights can be found by solving

$$\begin{bmatrix} U & \vec{x}_j \\ 0 & \xi_j \end{bmatrix} \vec{w}_j = \begin{bmatrix} \vec{z}_0 \\ \frac{\alpha_j}{\xi_j} \end{bmatrix}. \tag{3.7}$$

We define the changes in $\vec{w}_j$, $\Delta\vec{w}_j$, using the equation

$$\vec{w}_j = \begin{bmatrix} \vec{w}_0 \\ 0 \end{bmatrix} + \begin{bmatrix} \Delta\vec{w}_j \end{bmatrix}, \tag{3.8}$$

where $\vec{w}_0$ is the solution to $\Phi^T \Phi \vec{w}_0 = \Phi^T \vec{z}_0$ before $\Phi$ is modified ($U\vec{w}_0 = \vec{z}_0$). This

gives

$$\begin{bmatrix} U & \vec{x}_j \\ 0 & \xi_j \end{bmatrix} \Delta\vec{w}_j = \begin{bmatrix} \mathbf{0} \\ \frac{\alpha_j}{\xi_j} \end{bmatrix}. \tag{3.9}$$

The new estimate of $\vec{y}$ is then,

$$\hat{\vec{y}}_j = \begin{bmatrix} \Phi & \vec{\phi}_j \end{bmatrix} \vec{w}_j = \hat{\vec{y}}_0 + \begin{bmatrix} \Phi & \vec{\phi}_j \end{bmatrix} \Delta\vec{w}_j, \tag{3.10}$$

where $\hat{\vec{y}}_0 = \Phi\vec{w}_0$ is the old estimate. Let's define $\Delta\vec{y}_j = [\Phi \quad \vec{\phi}_j]\Delta\vec{w}_j$. The best node to select out of $\left\{\vec{\phi}_j\right\}$ is the node which contributes the most to the $\hat{\vec{y}}$ (has the largest $\Delta\vec{y}$). The length squared of $\Delta\vec{y}$ is

$$\|\Delta\vec{y}\|^2 = \Delta\vec{y}_j^T\Delta\vec{y}_j = \Delta\vec{w}_j^T \begin{bmatrix} \Phi^T \\ \vec{\phi}_j^T \end{bmatrix} \begin{bmatrix} \Phi & \vec{\phi}_j \end{bmatrix} \Delta\vec{w}_j. \tag{3.11}$$

Now, by substituting the modified Cholesky decomposition of equation (3.1) in the above equation we get,

$$\|\Delta\vec{y}_j\|^2 = \Delta\vec{w}_j^T \begin{bmatrix} U^T & 0 \\ \vec{x}_j^T & \xi_j \end{bmatrix} \begin{bmatrix} U & \vec{x}_j \\ 0 & \xi_j \end{bmatrix} \Delta\vec{w}_j. \tag{3.12}$$

Taking a step farther and applying equation (3.9) to the above equation we get

$$\|\Delta\vec{y}\|^2 = \left\| \begin{bmatrix} U & \vec{x}_j \\ 0 & \xi_j \end{bmatrix} \Delta\vec{w}_j \right\|^2 = \left\| \begin{bmatrix} 0 \\ \frac{\alpha_j}{\xi_j} \end{bmatrix} \right\|^2 = \frac{\alpha_j^2}{\xi_j^2}. \tag{3.13}$$

Rewritting the above equation,

$$\boxed{\|\Delta\vec{y}_j\|^2 = \frac{\alpha_j^2}{\xi_j^2}.} \tag{3.14}$$

The equation (3.14) is very important since it determines how much improvement we get by adding the new node $\vec{\phi}_j$. This equation will be the decision

criterion for including a node. So if we have to choose from the collection $\{\vec{\phi}_j\}$, the wisest choice would be to choose the node that gives the largest value of $\alpha_j^2/\xi_j^2$. Now we need to find how all the $\alpha_i$, $\xi_i$, and $x_i$ changes as $\Phi$ gets updated by a node.

To describe the updating rules, we introduce the following notation. Any quantity with $\tilde{}$ on top denotes the updated version of that quantity. That is, $\tilde{\star}$ is the value of $\star$ after $\vec{\phi}_k$ has been added to the $\Phi$ matrix. Here $\star$ can be a matrix, a vector or a scalar quantity, and $\vec{\phi}_k$ is the $k^{th}$ node that has been selected to be the best choice from the set $\{\vec{\phi}_j\}$. So using this notation we have,

$$\begin{aligned} \tilde{\Phi} &= \begin{bmatrix} \Phi & \vec{\phi}_k \end{bmatrix}, \\ \tilde{U} &= \begin{bmatrix} U & \vec{x}_k \\ 0 & \xi_k \end{bmatrix}. \end{aligned}$$

We need to find $\tilde{\alpha}_j$, $\tilde{\xi}_j$, and $\tilde{\vec{x}}_j$ in terms of $\alpha_j$, $\xi_j$, $\vec{x}_j$, $\alpha_k$, $\xi_k$, and $\vec{x}_k$.

The terms $\tilde{\xi}_j$ and $\tilde{\vec{x}}_j$ form the Cholesky Decomposition of the Gram matrix after the modification: that is by multiplying out the form of equation (3.1)

$$\begin{bmatrix} \tilde{\Phi}^T\tilde{\Phi} & \tilde{\Phi}^T\vec{\phi}_j \\ \tilde{\Phi}^T\vec{\phi}_j & \vec{\phi}_j^T\vec{\phi}_j \end{bmatrix} = \begin{bmatrix} \tilde{U}^T\tilde{U} & \tilde{U}^T\tilde{\vec{x}}_j \\ \tilde{\vec{x}}_j^T\tilde{U} & \tilde{\xi}_j^2 + \tilde{\vec{x}}_j^T\tilde{\vec{x}}_j \end{bmatrix}. \tag{3.15}$$

To get $\tilde{\vec{x}}_j$ we must solve, $\tilde{U}^T\tilde{\vec{x}}_j = \tilde{\Phi}^T\vec{\phi}_j$, that is,

$$\begin{bmatrix} U^T & 0 \\ \vec{x}_k^T & \xi_k \end{bmatrix} \tilde{\vec{x}}_j = \begin{bmatrix} \Phi^T \\ \vec{\phi}_k^T \end{bmatrix} \vec{\phi}_j. \tag{3.16}$$

By using the methods of forward substitution and using the fact that $\vec{x}_j$ is the

solution to $U^T\vec{x}_j = \Phi^T\vec{\phi}_j$, we get

$$\boxed{\tilde{\vec{x}}_j = \begin{bmatrix} \vec{x}_j \\ \frac{1}{\xi_k}(\vec{\phi}_k^T\vec{\phi}_j - \vec{x}_k^T\vec{x}_j) \end{bmatrix}}. \tag{3.17}$$

For simplification the last component of $\tilde{\vec{x}}_j$ will be called $\beta_j$, i.e. $\beta_j = \frac{1}{\xi_k}(\vec{\phi}_k^T\vec{\phi}_j - \vec{x}_k^T\vec{x}_j)$. We obtain $\tilde{\xi}_i$ by using the last equation ($\tilde{\xi}_j^2 + \tilde{\vec{x}}_j^T\tilde{\vec{x}}_j = \vec{\phi}_j^T\vec{\phi}_j$) of (3.15),

$$\begin{aligned} \tilde{\xi}_j^2 &= \vec{\phi}_j^T\vec{\phi}_j - \tilde{\vec{x}}_j^T\tilde{\vec{x}}_j \\ &= \vec{\phi}_j^T\vec{\phi}_j - \vec{x}_j^T\vec{x}_j - \beta_j^2 \end{aligned}$$

$$\boxed{\tilde{\xi}_j^2 = \xi_j^2 - \beta_j^2,} \tag{3.18}$$

where $\xi_j^2 = \vec{\phi}_j^T\vec{\phi}_j - \vec{x}_j^T\vec{x}_j$ is the value before $\vec{\phi}_k$ is added to $\Phi$.
Finally $\tilde{\alpha}_i$ can be found from equation (3.6):

$$\begin{aligned} \tilde{\alpha}_j &= \vec{\phi}_j^T\vec{y} - \tilde{\vec{x}}_j^T\tilde{\vec{z}}_0 \\ &= \vec{\phi}_j^T\vec{y} - \begin{bmatrix} \vec{x}_j^T & \beta_j \end{bmatrix}\begin{bmatrix} \vec{z}_0 \\ \frac{\alpha_k}{\xi_k} \end{bmatrix} \\ &= \vec{\phi}_j^T\vec{y} - \vec{x}_j^T\vec{z}_0 - \frac{\alpha_k\beta_j}{\xi_k} \end{aligned}$$

$$\boxed{\tilde{\alpha}_j = \alpha_j - \frac{\alpha_k\beta_j}{\xi_k},} \tag{3.19}$$

where $\alpha_j = \vec{\phi}_j^T\vec{y} - \vec{x}_j^T\vec{z}_0$.

The development of the fast orthogonal search comes down to only four important equations: (3.14, 3.17, 3.18, 3.19), these equations are boxed above. To select nodes one by one, we look first at the equation (3.14) to get the highest contributor, and then update the important terms of equations (3.17, 3.18), which

are also terms in $U$, due to inclusion of the most important node, and finally update (3.19) so that the next important node can be selected. This selection process can keep repeating until the desired number of nodes has been selected, or an error criterion has been met for the estimate. In the next section a very simple algorithm will be presented to implement it to find the weights.

## 3.2 Fast Orthogonal Search Algorithm

In section 3.1 the mathematical background was given for the fast orthogonal search technique. Even though the development of the technique looks complicated, the implementation of the technique is not so. In this section an algorithm will be presented for the technique, which will make it much easier to implement. The most important equations that we need to worry about are the ones boxed in section 3.1. Let's look at the RBF equation (2.3) of the matrix form again,

$$\hat{\vec{y}} = \Phi\vec{w}. \tag{3.20}$$

We need to solve this equation for weights $\vec{w}$. The matrix $\Phi = [\vec{\phi}_1\vec{\phi}_2\cdots\vec{\phi}_N]$ is the output of all $N$ nodes for $M$ samples. For the orthogonal search the nodes (or the columns $\vec{\phi}_i$) will be added one by one in an order such that the node which gives highest improvement, as in equation (3.14), is given the highest preference. The following algorithm presents the appropriate steps to implement the technique.

1. Put all the columns of $\Phi$ ($\vec{\phi}_i$'s) in the set $\{\vec{\phi}_j\}$.

   Initialization of some variables by setting,

$$\begin{aligned} \alpha_i &= \vec{\phi}_i^T\vec{y}, \\ \xi_i^2 &= \vec{\phi}_i^T\vec{\phi}_i, \\ x_i &= [0\times 1 \quad vector], \\ U &= [0\times 0 \quad matrix], \end{aligned} \tag{3.21}$$

where, $i = 1, 2, \cdots, N$,

Set $Error = \vec{y}^T\vec{y}$, and $Number\_node\_selected = 0$.

2. The iteration begins here.

   Find the maximum value of $\alpha_i^2/\xi_i^2$ for all i. Let's say the maximum is at $i = k$.

3. Now set,

$$\tilde{U} = \begin{bmatrix} U & \vec{x}_k \\ 0 & \xi_k \end{bmatrix}. \tag{3.22}$$

4. Now we update the $\vec{x}_i$ as in equation (3.17), by finding $\beta_i$ first,

$$\beta_i = \frac{1}{\xi_k}(\vec{\phi}_k^T\vec{\phi}_i - \vec{x}_k^T\vec{x}_i) \tag{3.23}$$

   and then,

$$\tilde{\vec{x}}_i = \begin{bmatrix} \vec{x}_i \\ \beta_i \end{bmatrix}. \tag{3.24}$$

5. We also need to update $\xi_i$ as in equation (3.18),

$$\tilde{\xi}_i^2 = \xi_i^2 - \beta_i^2. \tag{3.25}$$

6. Finally we update $\alpha$ as in equation (3.19) by,

$$\tilde{\alpha}_i = \alpha_i - \frac{\alpha_k\beta_i}{\xi_k}. \tag{3.26}$$

   In the above, from step 4 to step 6 updating is only required when $i \neq k$.

7. Keep repeating from step 4 through step 6 until all the nodes in the set $\{\vec{\phi}_j\}$ have been updated.

8. Delete the $k^{th}$ node from the set $\{\vec{\phi}_j\}$.

9. Increment the *Number_node_selected* by one, and set $Error = Error - \alpha_k^2/\xi_k^2$. If $Error$ is less than some error threshold, or the *Number_node_selected* is equal to the desired number of nodes then go to step 10, otherwise go through the steps 2 - 8 again.

10. At this stage we have a $U$ matrix giving the Cholesky decomposition of $\Phi$

$$\Phi^T\Phi = U^TU. \tag{3.27}$$

Note that here $\Phi$ does not contain the response from all nodes as in section 2.2.2, $\Phi$ is formed only from the set of nodes that reduce the sum squared error of $\hat{\vec{y}}$. We can use the equation above in the normal equation (2.10), and then solve for the weights of the selected nodes by the method of forward substitution, and backward substitution.

The algorithm presented in this section not only implements the technique to find the weights of a given network, but also allows the user to select the number of nodes. This selection has physical significance since there may exist hardware or software limitations on implementing nodes. The algorithm finds the best fixed number of nodes rather than just an arbitrary choice of nodes. If the number of nodes is not an issue than one may be able to find a better network by choosing a sum square error threshold. Also we can look at how the error is behaving as the nodes have been added to the network. This provides an indication of whether addition of a node really makes a difference or not. A subroutine which implements the above algorithm is included in Appendix A.1.

## 3.3 Computational Evaluations

This section will compare the procedure of fast orthogonal search technique presented in this chapter to the orthogonal search techniques presented in section 2.2.4 in terms of computational complexity. Here, the complexity of an algorithm may be evaluated by counting the number of multiplications required to implement the training procedure. The number of multiplications required for the fast orthogonal search described by Korenberg [14] is given by $O(SNM + S^3N)$. Here $S$ is the total number of nodes selected, $N$ is the total number of initial nodes, and $M$ is the total number of training samples. For the Orthogonal search technique of section 2.2.4 (developed by Chen [13]) the total number of multiplications required is $O(S^2NM)$. The fast orthogonal search technique developed in this chapter requires $O(SNM+S^2N)$ multiplications. So the fast orthogonal algorithm presented in section 3.2 is not only simple but also fast. It requires a factor of $S$ fewer computations than the method presented in [13], and is either comparable to or faster than Korenberg's algorithm of [14].

In the next chapter some applications of the RBF network will be given. The technique will also be evaluated against standard techniques, and an analysis of different aspects of the network will be presented.

# CHAPTER 4

# Applications of the RBF Network

The application of RBF neural networks is widespread. RBF networks have been successfully applied in pattern recognition [3, 6, 7], function approximation [8], time series prediction [8], signal detection [9, 28] and many other important problems. In this chapter, several applications of the RBF network using the fast orthogonal search for training will be given, and an analysis of the performance of the network will be provided.

In section 4.1 the application of RBF networks in analog to digital converter (ADC) error compensation will be presented. In section 4.2 a simple pattern recognition problem will be handled, and finally in section 4.3 the application to chromosome classification will be presented.

## 4.1 Analog to Digital Converter Error Compensation

The error compensation of ADC's has been undergoing research for several years [29]. Although the error introduced by an ADC is much smaller than the signal amplitude in general, the compensation of an ADC may be used to improve the linearity of the converter. For many spectral analysis and signal processing applications, the linearity of the converter is a critical specification. Despite the heavy growth of neural networks, few have attempted compensation using neural networks.

### 4.1.1 Problem Description

The error correction for an ADC is to eliminate frequency dependent distortion introduced by the converter [30]. To cope with this frequency dependent distortion Hummels et. al. [30] present a dynamic compensation procedure. This dynamic compensation was based on the the converter output and some estimate

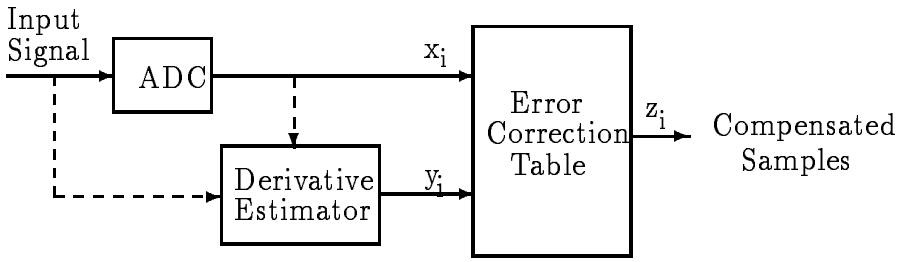


Figure 4.1: ADC Compensation Architecture

of the slope of the input signal. Figure 4.1 illustrates the compensation procedure where the values of the error function stored in memory, are accessed by the state and slope of the input signal.

In figure 4.1, $x_i$ is the $i^{th}$ output of the ADC, and $y_i$ is the estimate of the derivative of the input signal at the time that the sample $x_i$ is taken. $z_i$ represents the compensated signal given by

$$z_i = x_i - e(x_i, y_i), \tag{4.1}$$

where $e(x_i, y_i)$ is the unknown error function to be tabulated.

The job of the RBF neural network here is to determine the error function, $e(x_i, y_i)$, given the output of the converter and the estimated slope of the input signal. To train the RBF neural network an ADC calibration procedure is required to get input-output pairs for training. In [30] the detailed calibration procedure is presented. In brief, an ADC is excited by pure sinusoidal calibration signals of known frequency. The output of the converter is collected as a calibration data $x_i$, and an estimate of the derivative, $y_i$, is found. Figure 4.2 shows the calibration data versus the derivative of the input signal, in other words the domain of the error function. In this figure each ellipse is constructed by a single signal frequency.

Since the error is due to the frequency dependent distortion, the target (or the desired output of the RBF network) is the function which exactly compensates

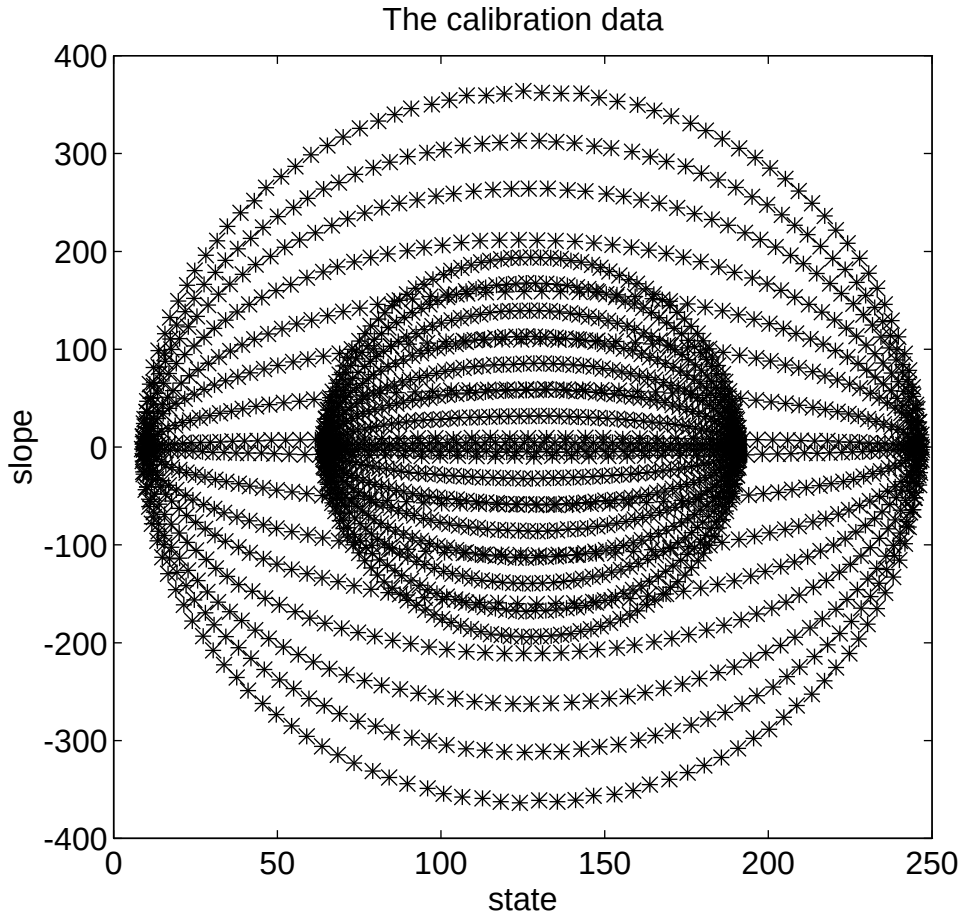


Figure 4.2: The domain of the error function.

the harmonics which exist in the calibration data. The above described problem can be represented by

$$F\Phi(\vec{x},\vec{y})\vec{w} = F\vec{x}. \tag{4.2}$$

Here, $F$ represent a Fourier Transform (FT) matrix, $\Phi(\vec{x},\vec{y})$ is the matrix form of the output of the RBF nodes, and $\vec{x}$ and $\vec{y}$ are the calibrated state data and the estimate of the derivative respectively. $\vec{w}$ is the weight vector of the RBF network. For a more detailed discussion on equation 4.2, see [30].

### 4.1.2 Results

Calibration data was collected from a Tektronix AD-20 8-bit converter sampling at 204.8 megasamples per second (MSPS). The data was then presented to a RBF network to learn the harmonics. The RBF nodes of the network were placed on training samples which were separated by a fixed minimum distance. The covariance for each node is chosen to be diagonal with diagonal elements proportional to the separation distance of the nodes. The $\Phi$ matrix of the network was constructed as in equation (4.2) with 300 nodes. Finally the weights of the network were found by the fast orthogonal search technique presented in Chapter 3. The technique of orthogonal search was used to find the best 50 nodes and their weights.

After training, an error table was constructed using the estimated function $e(x,y)$. Figure 4.3 shows the constructed error table. To evaluate the performance of the table a measure of Spurious Free Dynamic Range (SFDR) was evaluated. The SFDR was found by measuring the dB difference between the height of the fundamental and the height of the highest spurious signal in the spectrum of the ADC output as illustrated in Figure 4.4.

This measure is not a simulation, it is an actual measure done using a high performance test bed and the same AD-20 converter. Raw data was collected by driving the converter with a pure sinusoidal signal, and was compensated using

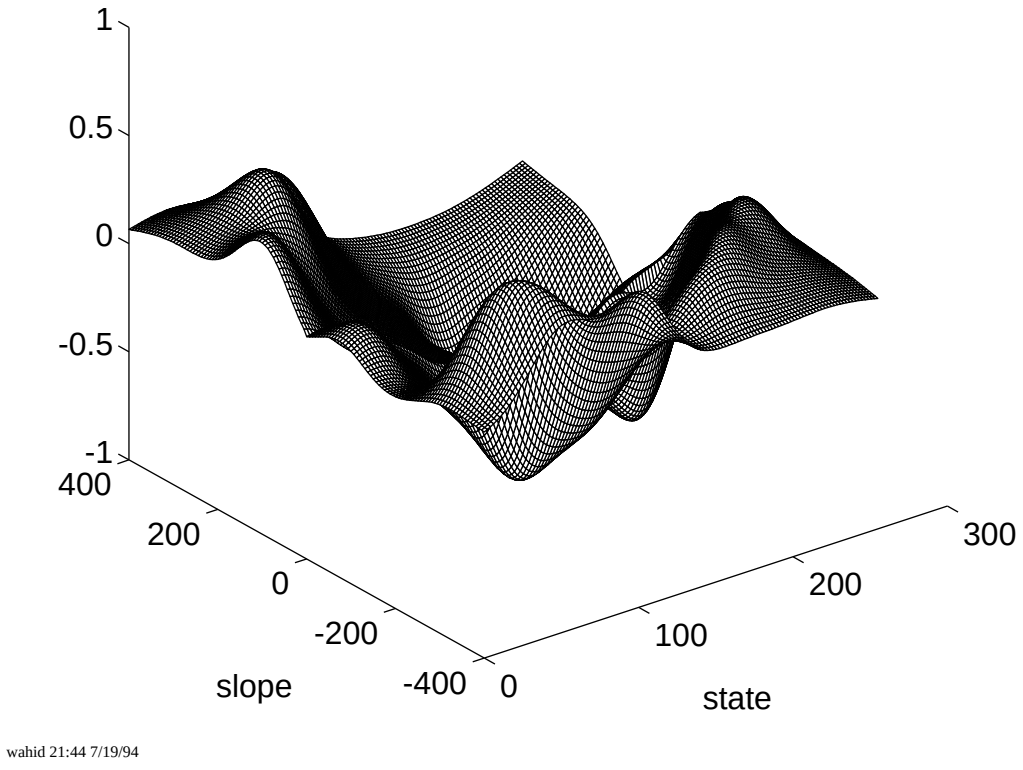


Figure 4.3: An error table.

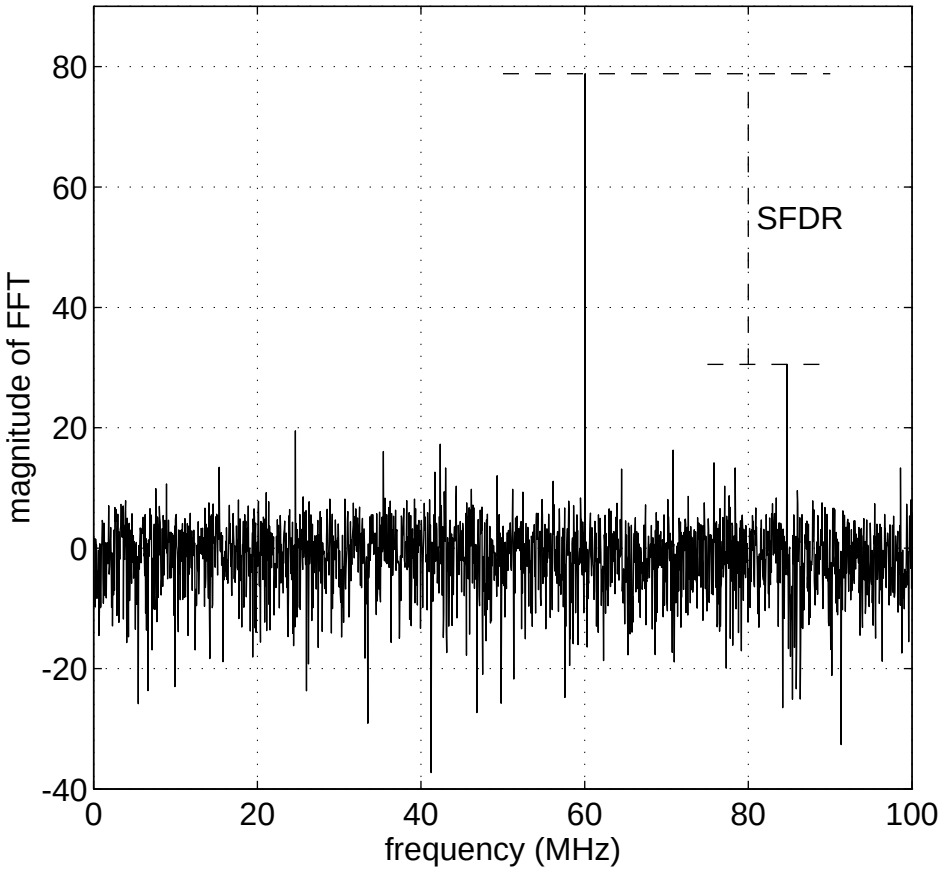


Figure 4.4: Measure of SFDR.

the table built by the RBF network as in (4.1).

Figure 4.5 shows the SFDR versus various test frequencies. The table is compensating the error by 10 dB or better throughout the Nyquist band. For comparison, a table was built using the same RBF network structure, but the weights were found using the method of Singular Value Decomposition (SVD) discussed in Section 2.2.3. The SFDR for the SVD technique is also shown in the figure 4.5. Even though the SFDR for both SVD technique and orthogonal search technique are similar, the orthogonal technique has some added advantages. The orthogonal technique is less time consuming than the SVD technique since the SVD technique requires finding the eigenvalues and eigenvectors of large matrices. The orthogonal technique gives the same performance with fewer nodes, providing a way to discard the unimportant nodes. If we look at the sum squared error as a node gets added in, we can make an intelligent decision about whether to add more nodes or not.

Figure 4.6 shows how the error function decreases as each node is added into the network. From this plot it is obvious that it may not improve the performance at all after the $30^{th}$ node had been added. In comparison, use of the SVD technique requires inclusion of all 300 nodes.

For many problems, insight into the importance of various nodes may be gained by examining the sequence with which nodes are added to the network. Figure 4.7 illustrates this selection process. The initial nodes are presented in the figure by 'o' and the selected nodes are presented by '$*$'. Next to each selected node, a number is printed which indicates the order in which the nodes are selected. This figure demonstrates the most important regions of the domain of this problem.

To get a better insight into the problem, a different procedure was used to place the initial nodes. The node centers of a RBF network were placed in layers of arrays of nodes. The first layer includes only 4 nodes on 4 corners of the input space of the ADC calibration data. $4 \times 4$ array of nodes were placed on

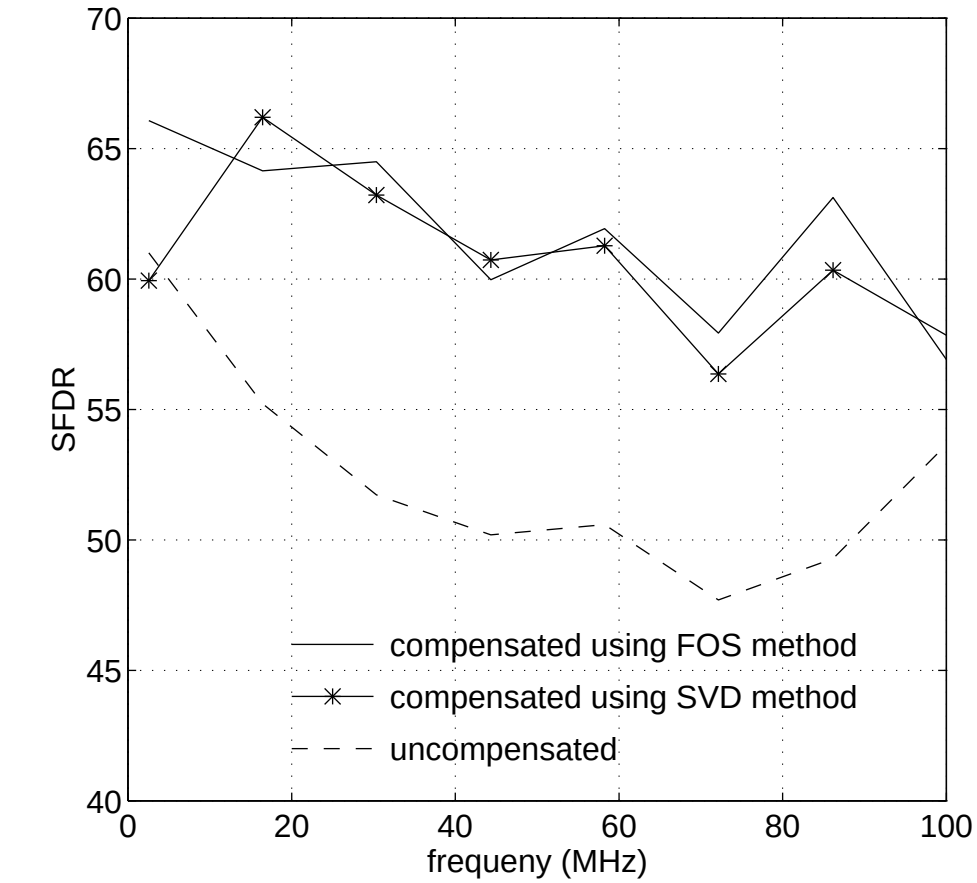


Figure 4.5: SFDR for the uncompensated, compensated by proposed technique, and compensated by singular value decomposition.

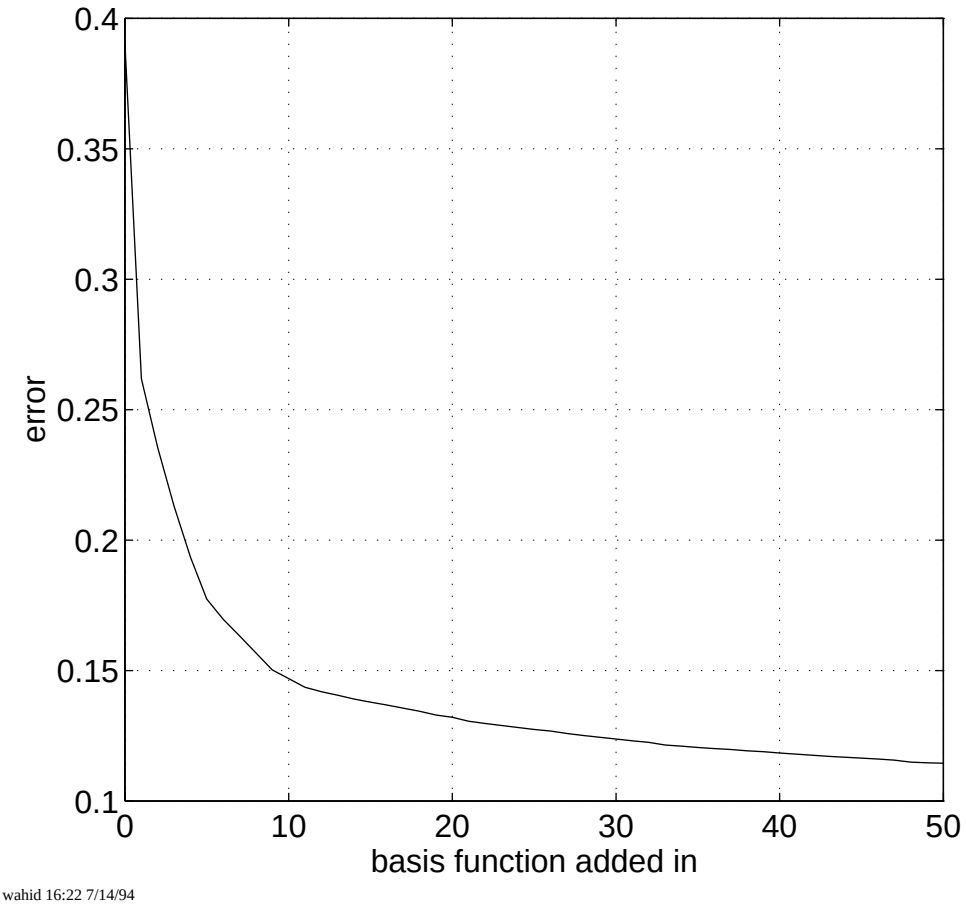


Figure 4.6: The sum squared error for the network as each node gets added in.

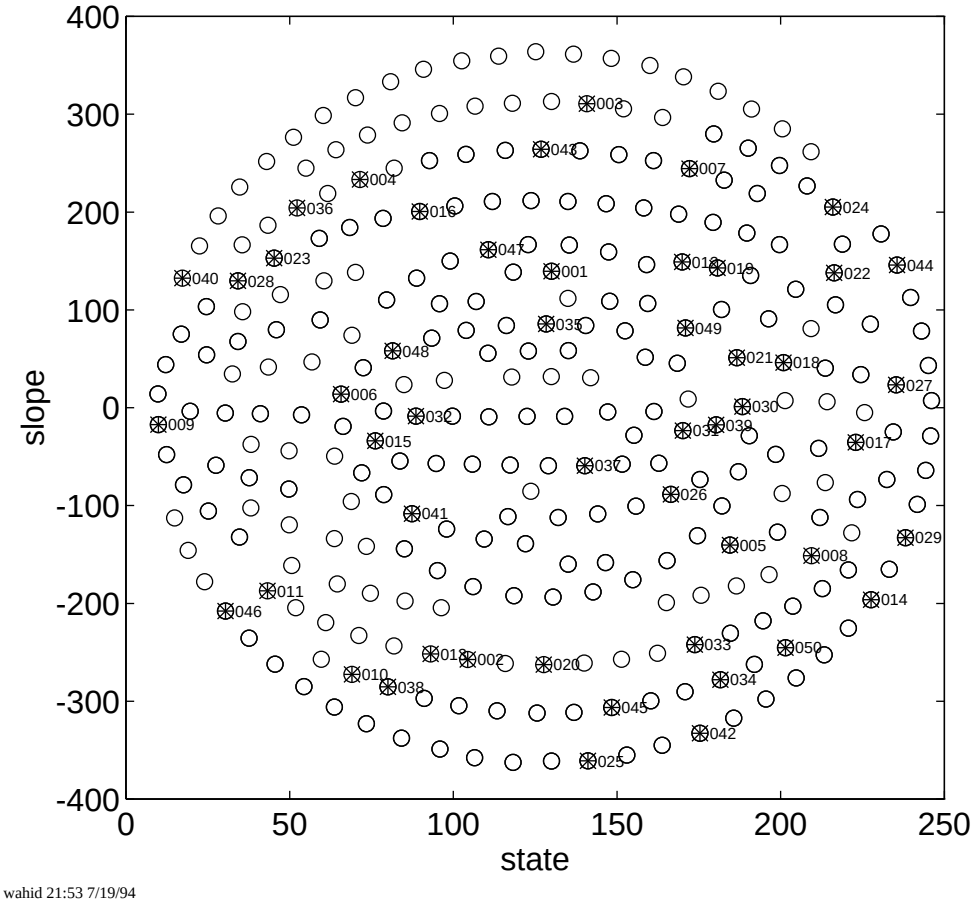


Figure 4.7: The selection of 50 nodes out of 500 initial nodes.

the second layer, $8 \times 8$ array of nodes were placed on the third layer, and finally $16 \times 16$ array of nodes were placed on the last layer. The covariance matrix of each node was made diagonal with the diagonal elements given by the distance between two neighboring nodes of the same layer. This node placement scheme provides the network with the opportunity to give the error table general shape using the widely spaced node of layers 1,2 and 3, and fine structure using the nodes of layer 4.

Figure 4.8 presents the above described scheme. Here the initial nodes are shown using '.', and the selected nodes are shown using 'o'. Notice that the first 11 nodes were selected from layers 2 and 3 to give the table a general shape, and then the remaining 39 nodes were selected from layer 4 to provide the required fine structure of the table. No nodes were selected from layer 1 since the nodes were outside the domain of ADC (figure 4.2).

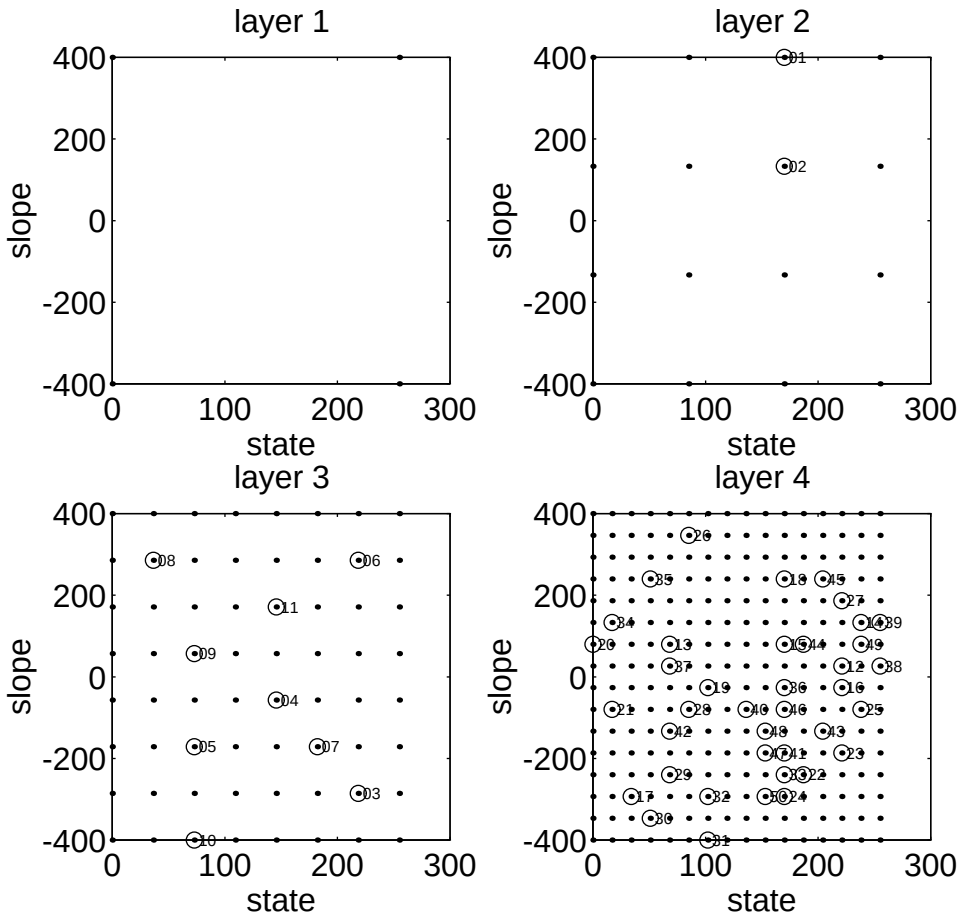


Figure 4.8: The selection of 50 nodes out of the 4 layered nodes.

## 4.2 A Simple Pattern Recognition Problem

From the signal processing application of the previous section, this section moves to a pattern recognition problem which is much more familiar territory for neural networks. The problem here is to classify simple two dimensional, and two class patterns. A training set of 400 samples were generated as Gaussian random vectors. The first class (class '0') are zero mean random vectors, with an identity ($I$) covariance matrix. The second class (class '1') has mean vector $[1 \quad 2]^T$, and a diagonal covariance matrix with diagonal elements 0.01 and 4.0. Figure 4.9 shows the training data set described above. Here 'o' is used for class 0, and a '+' is used for class '1'. A test set of 20000 samples was also generated using the same Gaussian density parameters. The reason for choosing this example is that it has been used by several researchers to evaluate various neural network classification schemes [6, 7, 16, 27, 31].

The RBF network presented here was trained using the 400 training samples. The nodes of the network were placed on the 400 training samples and all the nodes had the same covariance matrix $\sigma^2 I$, where sigma is the width chosen between 0.1 and 1. The network was asked to find 50 best nodes, and their weights. The desired network output was set to zero for any class '0' sample, and one for class '1' samples. After training, the network was tested with the 20000 test samples. Figure 4.10 gives the percentage error versus the width of a node $\sigma$. The minimum error was found to be 7.92% when the $\sigma$ was 0.4. The minimum error given by [6] was 9.26%, and the optimal error for this problem using a quadratic classifier was 6% [6]. So the network presented here not only outperforms the standard RBF approaches [6, 27], but also requires fewer nodes. In [6] the 9.26% error was obtained by using 86 nodes, and all the nodes had different covariance matrices. The Fast Orthogonal Search technique found the 50 best nodes and their weights in less than 2 minutes in a DEC3100. So the speed of training is also very good.

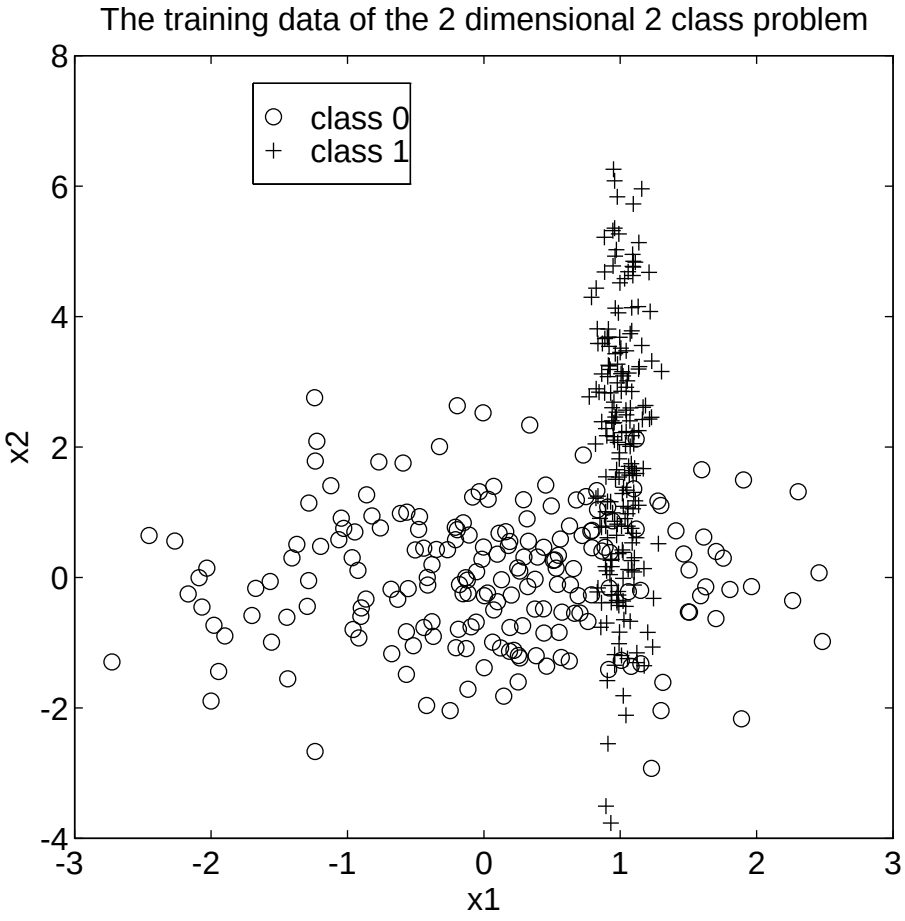


Figure 4.9: The training sample for the 2D 2-class problem.

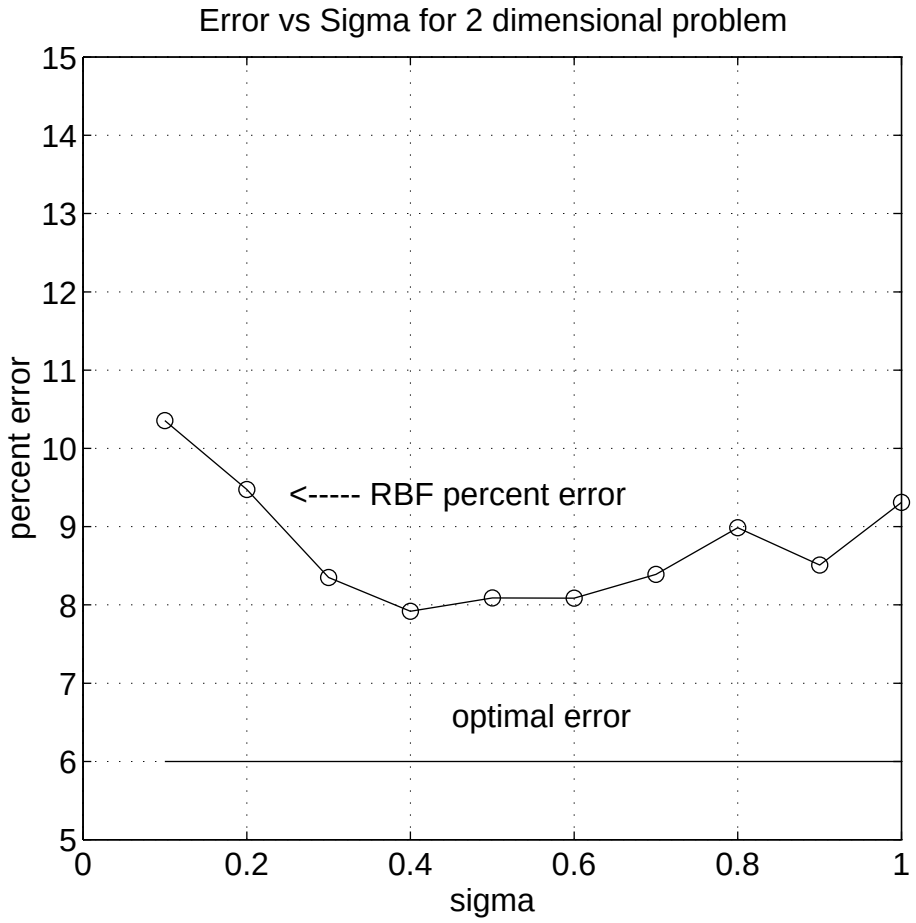


Figure 4.10: The percentage error vs $\sigma$ for the 2D 2-class problem.

To see how the RBF network is generalizing over the entire domain, figure 4.11 shows the output surface of the network over the entire domain of the input sample.

The behavior of the sum square error on the training sample is also an issue of our discussion. Figure 4.12 shows how the error is decreasing as each node gets added in. Again as in the previous section the sum square error on the training sample becomes almost flat as the 'significant' nodes have been added. Finally a plot is presented here which shows how the search technique selected the 'significant' nodes from the whole set. Figure 4.13 shows the 50 nodes selected by the network, and their order of selection. From this plot we can see that the first node was picked from a pure class 0 region, and the second node was picked from the pure class 1 region, which makes complete sense since we want the most important nodes to be where most of training samples are. Also from figure 4.12 we can see that the introduction of the first node reduces training error by 50%.

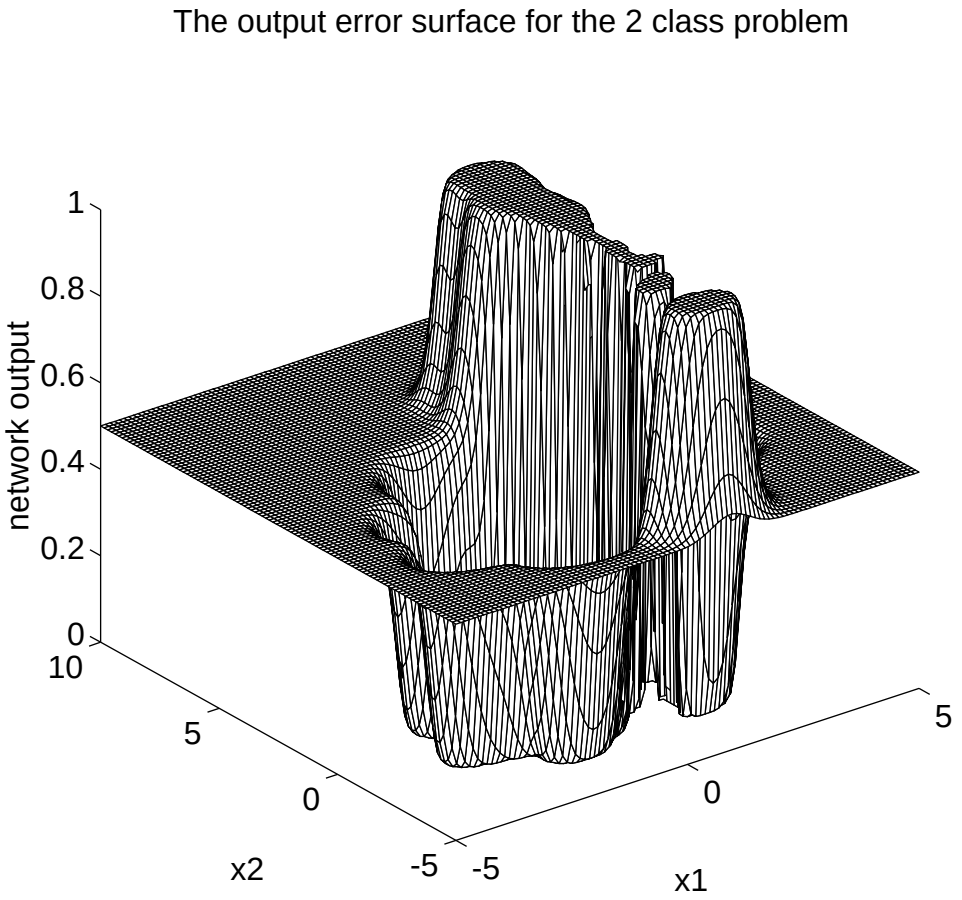


Figure 4.11: The output surface of the RBF for the 2D 2-class problem.

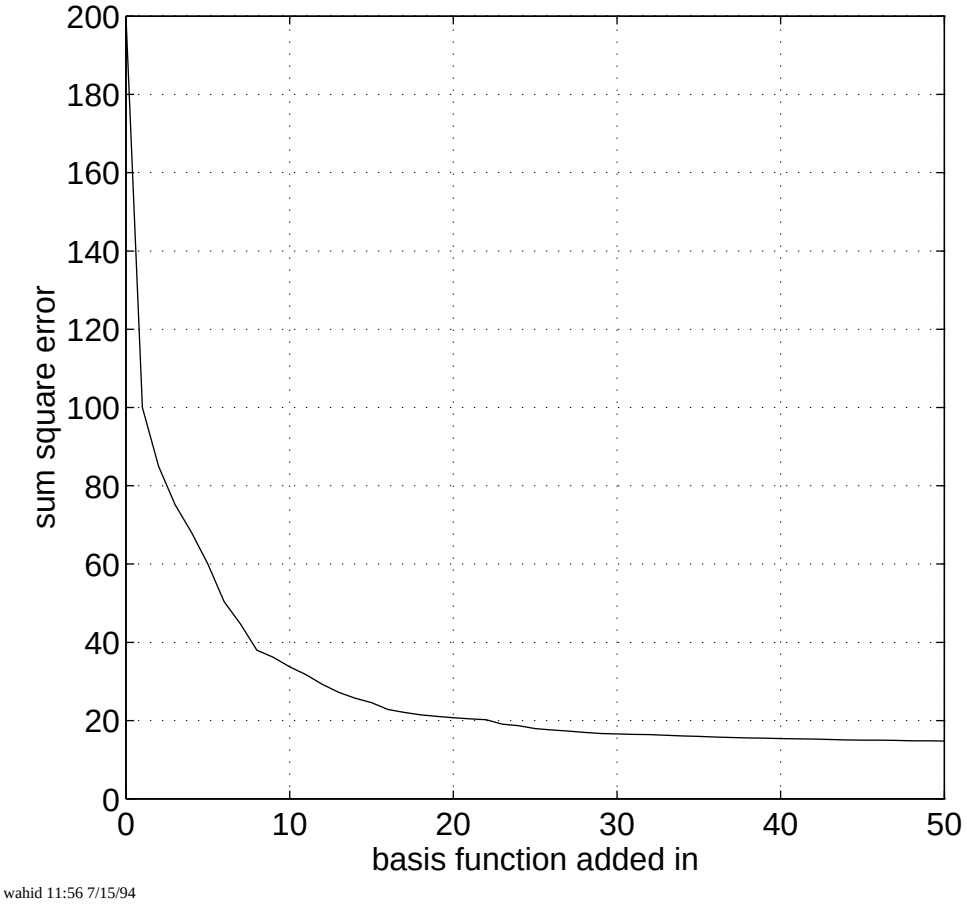


Figure 4.12: The sum squared error of the training samples for the 2D 2-class problem.

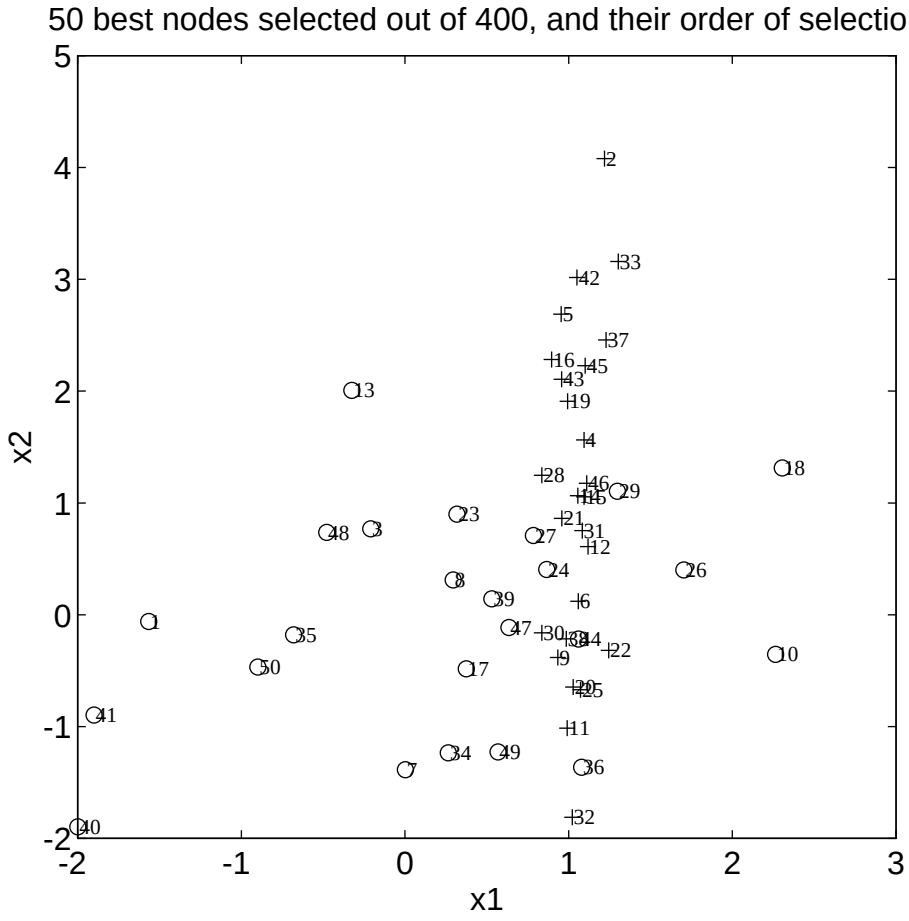


Figure 4.13: The node selection process of the orthogonal search technique.

## 4.3 Classification of Chromosome

A more complicated and challenging application of RBF will be discussed in this section. "Karyotyping", the classification of the chromosome in a metaphase into the 24 normal classes has been a very important issue in the medical field for many many years. However, the automation of karyotyping by computers has been in development only for about 25 years [32]. Classification of chromosomes involves finding a good set of features to describe a chromosome, and a classification technique to identify the chromosomes using the features. The RBF neural network developed in this report will be used for the classification of chromosome given a set of features.

### 4.3.1 RBF Network for Chromosome Classification

The problem of karyotyping involves classifying the chromosome of ee features (for the data base used here) into 24 different classes. The chromosomes in a cell consists of 22 pairs of autosomes, one of each pair inherited of each parent, and two sex chromosomes (an X and Y for male, and two X's for female). Classification of a cell correctly requires classification of all 24 classes of a cell. Rather than classifying a cell, this report will look at the classification per class. So here the problem is to only classify each pair of autosomes, and the sex chromosomes.

The RBF structure for the chromosome classification is slightly different than the one given in Section 2.1. The output layer of the network consists of 24 output nodes to classify the 24 classes. The decision of the network is the node that gives highest output. One major advantage of the fast orthogonal search technique will be evident here. Figure 4.14 illustrates the RBF network for chromosome classification. In standard RBF networks, all the nodes of the hidden layer are connected to all the nodes of the output layer. The fast orthogonal search will be able to reduce the insignificant connections. The reduction of the nodes can be of very large scale for a multiclass problem like the chromosome classification. The

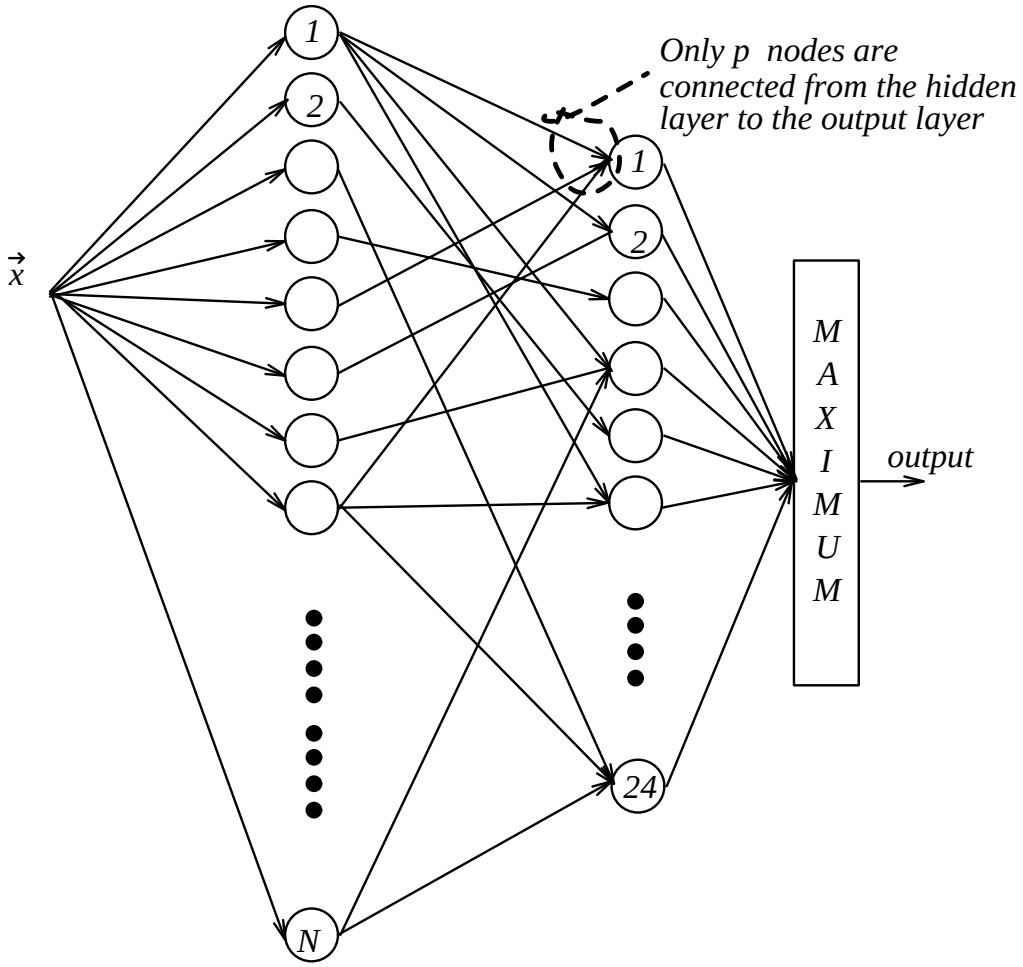


Figure 4.14: Network structure for the RBF Network for Chromosome Classification.

following result and analysis section will show this phenomenon.

### 4.3.2 Results and Analysis

The chromosome database used for evaluating the method is the Philadelphia database [32, 33, 34]. Each pattern of this database is an autosome, the X sex chromosome, or the Y sex chromosome. Each pattern consists of a set of 30 different features, which are the measurements of the normalized area, size, density, normalized convex hull perimeter, normalized length, area, centromeric index, mass centromeric index, length centromeric index, the weighted density distribution density, and others [34].

The RBF neural network was trained with 1000 training patterns. The initial nodes of the network were placed on these 1000 patterns. The covariance of each node was chosen to be diagonal with initial diagonal elements equal to the estimated variance of the class the node belongs to, that is

$$\sigma_{ik}^2 = E\left\{(x_{jk} - c_{ik})^2\right\} \qquad \vec{x}_j, \vec{c}_i \ \in \{class\ l\}. \tag{4.3}$$

Where $\sigma_{ik}^2$ denotes the $k^{th}$ diagonal element of the covariance matrix $\Sigma_i$ for the $i^{th}$ node, and $c_{ik}$ is the $k^{th}$ component of the $i^{th}$ node center, and $x_{jk}$ is the $k^{th}$ feature of the pattern $\vec{x}_j$. The diagonal elements of the covariance matrix were adjusted to separate the closest discriminating nodes. The adjustment was made by multiplying the diagonal element of the covariance matrix by a fraction of the distance between two overlapping classes, that is

$$\Sigma_i = \gamma\Sigma_i(\vec{c}_j - \vec{c}_i)^T\Sigma_i^{-1}(\vec{c}_j - \vec{c}_i), \qquad \vec{c}_j, \vec{c}_i \ \notin \{class\ l\}. \tag{4.4}$$

Where $\gamma$ is a constant less than 1.0. The fast orthogonal search was used to only find the best 40 nodes and their weights. The search technique successfully found the desired nodes. Figure 4.15 shows the training error for class 1 as each node

was added in.

Notice here that only 40 out of 400 RBF nodes are connected to the output. By looking at figure 4.15, we see that the training error for class 1 has leveled off by the introduction of the $40^{th}$ node, implying that introducing another node may not improve the performance at all. Similar conclusions can be made for training other classes.

After training the network, it was tested by a test set of 3000 patterns different from the training set. The percent error for this test set was 20.87%, which was slightly (1.73%) lower than the ones presented by [32, 33, 34].

The number of initial nodes from which to select a set of nodes of the network can influence the performance. Figure 4.16 illustrates the results of some simple tests where the number of initial node assignments was changed for the training procedure described above. The plots in figure 4.16 give the percent error (y-axis) for a network constructed by selecting a set of nodes per class from a larger set of initial nodes (x-axis). The test was performed to select 20, 30, 40, 50 and 60 nodes per class from the initial sets. The percentage error was generally lower than the error given by [32, 33]. From Figure 4.16 we can also see that the percent error for 20 selected nodes per class is higher than all other selections. This suggests that 20 nodes per class is not necessarily the best. On the other hand, selection of 60 nodes does not improve the percent error at all suggesting overtraining. The smaller initial sets of (100-300) nodes are good, but not the best-too little to choose from, and the percent error increases as the initial nodes increases-too many to choose from. One other important issue is that even though the percent error is changing due to the selection process, the difference between the best performance and the worst performance of all the tests is merely 3.2%, suggesting that the fast orthogonal search procedure is finding the best possible selection for a given network.

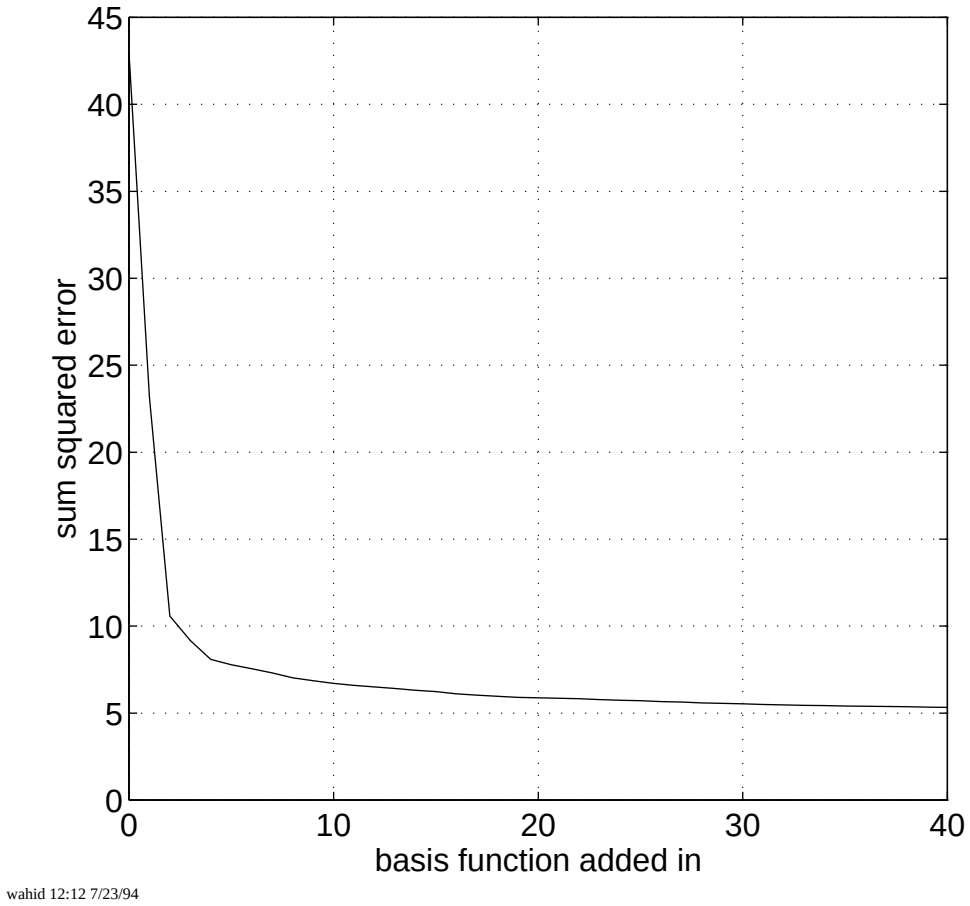


Figure 4.15: The sum squared error for training class 1 of the Chromosome problem.

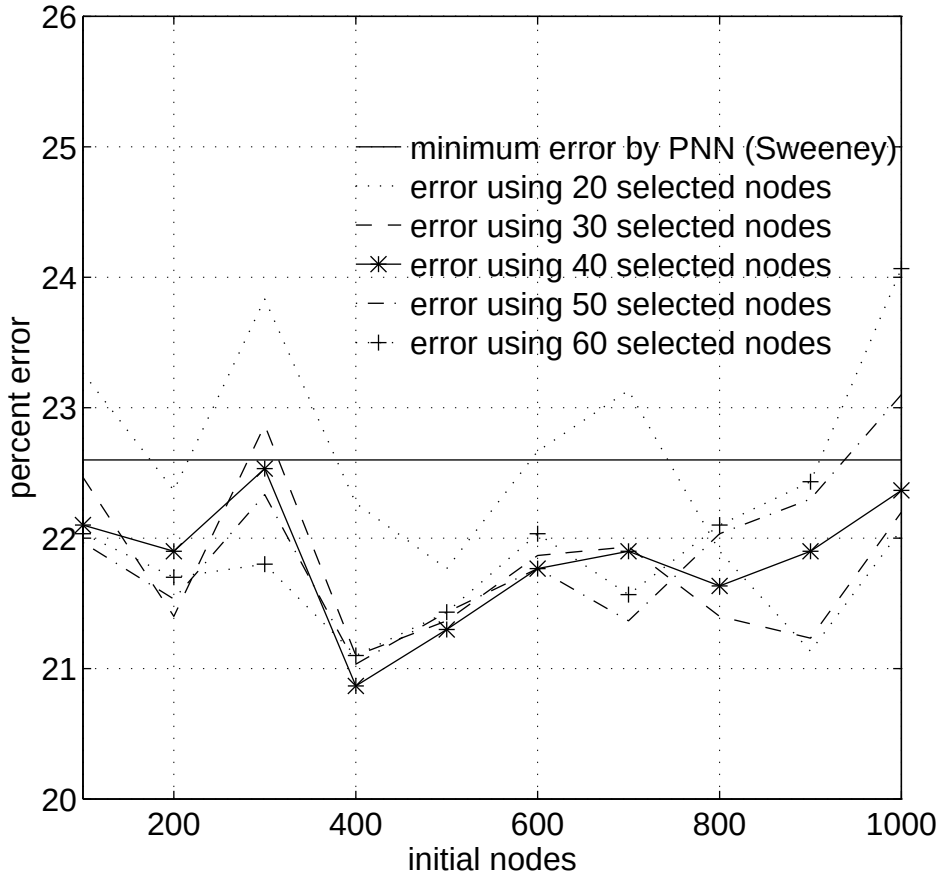


Figure 4.16: The percent error of the network for different numbers of the initial node selection

# CHAPTER 5

# Conclusion

## 5.1 Summary

This thesis presented an approach to configure the most significant component of the RBF neural networks, the weights.

1. The method provides a simple way to find the most significant nodes of the network and their weights.

2. The technique of fast orthogonal search is implementable using a simple 10 step algorithm. Traditional approaches require significantly more computations.

3. The technique provides a solution regardless of the network parameters. The provided solution is the best to match the target function. The traditional approaches may not converge, or may produce an erroneous solution.

4. The solution is correlated with the target function, so scaling of any node will not affect the network output. This is in contrast to many of the traditional approaches, where node activation functions must be carefully normalized.

5. The approach gives a clear indication of the number of nodes to be used. Nodes should be added only until addition of a node does not improve the output significantly.

6. Several important applications of the RBF network have been provided. The results show that the orthogonal search technique gives better performance than that of other approaches.

## 5.2 Future Work

The fast orthogonal search technique is a significant improvement over existing RBF training techniques. The method provides good insight into the concerned problem, and the size of the network required to address the problem. However, additional research could make the procedure more practical to implement, or more adaptable to complex problems.

The first issue is that if the problem is too big and requires a large network, we must contend with memory limitations. For example, if we want to train the chromosome problem of chapter 4 with 2000 training samples, and 1000 initial nodes, then the size of the $\Phi$ matrix is very large ($2000 \times 1000 \times 8 bytes \approx 16 MB$). It's almost impossible to implement a problem of this size. To solve this problem we have to look at how we can implement this technique without having to store the large matrix. It may be necessary to generate the columns of $\Phi$ again and again to solve the problem.

The second issue is that after we find the most significant node $\vec{\phi}_k$ from the set $\{\Phi_j\}$, is there anything we can do to make it even better? One possibility would be to optimize the node by changing its width $\sigma$. The work presented in this thesis may provide some insight into this problem, since specific equations are given which relate the vector $\vec{\phi}$ to the improvement in LS solution.

Finally, some more applications of the RBF network can be used to evaluate the procedure. For example Optical Character Recognition (OCR) is a good problem for neural network. The problem of handwritten zipcode recognition [35] can be a challenge for the fast orthogonal search technique.

# REFERENCES

[1] D.E. Rumelhart, J.L. McClelland, *Parallel Distributed Processing*, MIT Press, Cambridge, MA 1986.

[2] D.S. Broomhead, D. Lowe, "Multivariable Functional Interpolation and Adaptive Networks," *Complex Systems*, vol. 2, pp. 321-355, 1988.

[3] J. Moody, C.J. Darken, "Fast Learning in Networks of Locally Tuned Processing Units," *Neural Computation*, vol. 1, pp. 281-294, 1989.

[4] D.F. Specht, "A Generalized Regression Neural Network" *IEEE Transactions of Neural Networks*, vol. 2, No. 6, pp. 568-576, 1991.

[5] T. Kohonen, "The Self Organizing Map", *Proceedings of IEEE*, vol.78, No. 9, pp.1464-1480, 1990.

[6] M.T. Musavi, W. Ahmed, K.H. Chan, K.B. Faris, D.M. Hummels, "On the Training Algorithm for Radial Basis Function Classifiers," *Neural Networks*, vol. 5, pp. 595-603, 1992.

[7] K. Kalantri, *Improving the Generalization ability of Neural Network Classifiers* M.S Thesis, University of Maine, Orono, Maine, May 1992.

[8] R. D. Jones, Y. C. Lee, W. Barnes, G. W. Flake, K. Lee, P. S. Lewis, S. Qian, "Function approximation and Time Series Prediction with Neural Networks," *Proc. of the IJCNN*, pp 649-665, June 1990.

[9] W. Ahmed, D.M. Hummels, M.T. Musavi, "Adaptive RBF Neural Detection in Signal Detection," *Proccedings of International Symposium on Circuits and Systems (ISCAS '94)*, pp 265-268, May 29 - June 2, 1994, London, U.K.

[10] L. Xu, A. Krzyzak, A. Yuille, "On Radial Basis Function Nets and Kernel Regression: Statistical Consistency, Convergence Rates, and Receptive Field Size", *Neural Networks*, vol. 7, No. 4, pp. 609-628, 1994.

[11] S. Chen, B. Mulgrew, P.M. Grant, "A clustering technique for Digital Communications Channel Equalization Using Radial Basis Function Networks" *IEEE Transactions on Neural Networks*, vol. 4, No. 4, pp. 570-579, Mar. 1993.

[12] H. Spath, *Cluster Analysis Algorithms for Data Reduction and Classification of Objects*, Halstead Press, 1980, New York.

[13] S. Chen, C.F.N. Cowan, P.M. Grant, "Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks" *IEEE Transactions on Neural Networks*, vol. 2, No. 2, pp. 302-309, Mar. 1991.

[14] M.J. Korenberg, L.D. Paarmann, "Orthogonal Approaches to Time-Series Analysis and System Identification," *IEEE SP Magazine*, July 1991, pp. 29-43.

[15] L.D. Paarmann, M.J. Korenberg, "Accurate ARMA Signal Identification - Empirical Results," *Proceedings of Midwest Symposium*, pp 110-113,1991.

[16] M.T. Musavi, K.H. Chan, K. Kalantri, W. Ahmed, "A Minimum Error Neural Network," *Neural Networks*, vol. 6, pp. 397-407, 1993.

[17] K.H. Chan, *A Probabilistic Model For Evaluation of Neural Network Classifiers* M.S Thesis, University of Maine, Orono, Maine, May 1991.

[18] W.P. Jones, J. Hoskins, "Back Propagation - A General Delta Learning Rule," *Byte*, pp. 155-162, Oct. 1987.

[19] H. Guo, S.B. Gelfand, "Analysis of Gradient Descent Learning Algorithms for Multilayer Feedforward Neural Networks", *IEEE Transactions on Circuits and Systems*, vol. 38, No. 8, pp. 883-893, 1991.

[20] L.L. Scharf, *Statistical Signal Processing, Detection, Estimation, and Time Series Analysis*, Addision-Wesley Inc., 1991.

[21] R. A. Horn, C. A. Johnson, *Topics in Matrix Analysis*, Cambridge University Press, 1991.

[22] G. Sewell *The Numerical Solution of Ordinary and Partial Diffrential Equations*, Academic Press, INC., San Diego, CA, 1988.

[23] G. Strang, *Linear Algebra and Its Application*, Academic Press, 1976, New York.

[24] R. A. Horn, C. A. Johnson, *Matrix Analysis*, Cambridge University Press, 1985.

[25] N. Weiner, *Nonlinear Problems in Random Theory*, The Technology Press of MIT and John Wiley & Sons, Inc., New York, 1958.

[26] A.A. Desrochers, "On An Improved Model Reduction Technique for Nonlinear Systems" *Automatica*, vol. 17, pp. 407-409, 1981.

[27] M.T. Musavi, K.B. Faris, K.H. Chan, W. Ahmed, "A Clustering Algorithm for Implementation of RBF Technique," *International Joint Conference on Neural Networks, IJCNN '91*, Seattle, WA, July 1991.

[28] D.M. Hummels, W. Ahmed, M.T. Musavi, "Adaptive Detection of Small Sinusoidal Signals in Non-Gaussian Noise using a RBF Neural Network," to be published in 1994 in *IEEE Transactions of Neural Networks.*

[29] T.A. Rebold, F.H. Irons, "A Phase Plane Approach to Compensation of High Speed Analog to Digital Converters" *IEEE International Symposium on Curcuits and Systems*, pp. 455, Philadelphia, 1987.

[30] D.M. Hummels, F.H Irons, R. Cook, I. Papantonopoulos, "Characterization of ADCs Using a Non-Iterative Procedure," *Proccedings of International Symposium on Circuits and Systems (ISCAS '94)*, May 29 - June 2, 1994, London, U.K.

[31] M.T. Musavi, W. Ahmed, K.H. Chan, D.M. Hummels, K. Kalantri, "A Probabilistic Model for Evaluation of Neural Network Classifier," *Pattern Recognition*, vol. 25, pp. 1241-1251, 1992.

[32] W. P. Sweeney Jr., M.T. Musavi, J.N. Guidi
"Classification of Chromosomes Using Probabilistic Neural Network", *Cytometry* vol. 16 pp. 17-24, 1994.

[33] W. P. Sweeney Jr. *Classification of Human Chromosomes Using Probabilistic Neural Network*, M.S Thesis, University of Maine, Orono, Maine, August 1993.

[34] J. Piper, E. Granum, "On Fully Automatic Feature Measurement for Banded Chromosome Classification", *Cytometry* vol. 10 pp. 242-255, 1989.

[35] L. Sharma, *Recognition of Handwritten Zipcodes Using Probabilistic Neural Network*, M.S Thesis, University of Maine, Orono, Maine, May 1994.

# Appendix A
# PROGRAM LISTING

## A.1 Fast Orthogonal Search

```
/*****************************************************************
fast_ortho_search.c
Wahid Ahmed
Jun 26, 1994

This routine implements the Fast Orthogonal Search
for a input matrix.

the prototype of this function is:
    double *fast_ortho_search(MATRIX a, double *y,
                int TOTAL_BASIS, double err_thres);

    where, a is a matrix.
    y = aw;
    w is the unknown parameters returned by this routine.
      So w is a vector of weights, the unused nodes have weights 0.
    TOTAL_BASIS is number of nodes desired from the a matrix.
    err_thres is the sum squared training error threshold to quit.

*****************************************************************/

#include <stdio.h>
#include <math.h>
#include "vecmath.h"
#include "cmath.h"
#define SQ(x) ((x) * (x))               /* define square macro  */
#define MAX(a,b) ((a > b) ? a:b)        /* define maximum macro */

double *fast_ortho_search(MATRIX a,
     double *y,
                    int TOTAL_BASIS,
                    double err_thres)
{
     /* Define variables                                        */
     MATRIX U,L;          /* matrices for decomposition         */
     int i,j,k;
     double dummy;
```

```
double *p;           /* holds the correlation part         */
double *q;           /* holds the energy reduction part    */
double *C;           /* A pointer for the targets          */
double **x;          /* A double pointer for the additional
                        vector                              */
double *weight;      /* The result goes here               */
double *temp_vec1,
       *temp_vec2;   /* temporary vectors for various uses */
int *selected;       /* Array that holds the indeces for
   selected nodes */
int flag = 0;
double Maximum = -10;
int Max_index;
double error;        /* The error variable                 */

/* define the routines used                                */
MATRIX realloc_mat(MATRIX A,int new_rows,int new_cols);
double dot_prod(double *x, double *y, int length);

/* allocate memory for various pointers                    */
p = (double *)calloc(a.cols, sizeof(double));
q = (double *)calloc(a.cols, sizeof(double));

weight = (double *)calloc(a.rows, sizeof(double));

temp_vec1 = (double *)calloc(a.rows, sizeof(double));
temp_vec2 = (double *)calloc(a.rows, sizeof(double));

selected = (int *)calloc(a.cols, sizeof(int));
for (i=0;i<a.cols;i++) selected[i] = a.cols + 1;
x = (double **)calloc(a.cols, sizeof(double *));
for (i=0;i<a.cols;i++)
   x[i] = (double *)calloc(1,sizeof(double));

/* Initialize the weight matrix                            */
for (i=0;i<a.rows;i++) weight[i] = 0.0;

/* Initialize the matrix for the result                    */
U = initMatrix(1,1);

/* Initialize the correlation and energy values            */
for (i=0;i<a.cols;i++){
```

```
    for (j=0;j<a.rows;j++) temp_vec1[j] = a.el[j][i];
    p[i] = dot_prod(temp_vec1,y,a.rows);
    q[i] = dot_prod(temp_vec1,temp_vec1,a.rows);
}

/*  The initial sum squaerd error for the network          */
error = dot_prod(y,y,a.rows);
printf("initial Error = %1.6e\n",error);

/*  The first iteration                                    */
for (i=0;i<a.cols;i++){
    dummy = SQ(p[i])/q[i];
    Maximum = MAX(Maximum,dummy);
    if (Maximum == dummy) Max_index = i;
    /* printf("Energy = %1.4f for col = %d\n",dummy,i);    */
}

error -= Maximum;
/* printf("Error = %1.6e\n",error);                        */
selected[0] = Max_index;
U.el[0][0] = sqrt(q[Max_index]);
for (i=0;i<a.cols;i++){
    if (i == Max_index) continue;
    for (j=0;j<a.rows;j++){
    temp_vec1[j] = a.el[j][Max_index];
    temp_vec2[j] = a.el[j][i];
    }
    x[i][0] = dot_prod(temp_vec1,temp_vec2,a.rows)/U.el[0][0];
    q[i] -= SQ(x[i][0]);
    p[i] -= x[i][0] * (p[Max_index]/sqrt(q[Max_index]));
}



/* The rest of the iterations                              */
for (i=1;(i<TOTAL_BASIS) & (error > err_thres);i++){
    /* printf("iteration %d\n",i); */
    Maximum = -10;

    /* Find an unselected node that will provide highest
       improvement                                         */
    for (j=0;j<a.cols;j++){
```

```
flag = 0;
        for (k=0;k<i;k++)
    if (selected[k] == j){flag = 1; break;}
        if (flag) continue;
if (q[j] <= 1e-50) continue;
        dummy = SQ(p[j])/q[j];
        Maximum = MAX(Maximum,dummy);
        if (Maximum == dummy) Max_index = j;
    }

    error -= Maximum;
    /* printf("Error = %1.6e\n",error);                    */
    selected[i] = Max_index;

    /* First find the Orthogonal decomposition for the
       selection                                           */
    U = realloc_mat(U,i+1,i+1);
    for (j=0;j<i;j++) U.el[j][i] = x[Max_index][j];
    U.el[i][i] = sqrt(q[Max_index]);

    /* Update every thing else for selecting the next node */
    for (j=0;j<a.cols;j++){
        flag = 0;
        for (k=0;k<=i;k++)
             if (selected[k] == j){flag = 1; break;}
        if (flag) continue;
        x[j] = (double *)realloc(x[j],(i+1) * sizeof(double));
     for (k=0;k<a.rows;k++){
            temp_vec1[k] = a.el[k][Max_index];
            temp_vec2[k] = a.el[k][j];
     }
        x[j][i] = (dot_prod(temp_vec1,temp_vec2,a.rows)
                   - dot_prod(x[Max_index],x[j],i))/U.el[i][i];
        q[j] -= SQ(x[j][i]);
        p[j] -= x[j][i] * (p[Max_index]/sqrt(q[Max_index]));
    }
}
printf("final Error = %1.6e\n",error);

/* for (i=0;i<U.cols;i++) printf("%d\n",selected[i]);      */

/* The nodes are selected, now find the weights for these
```

```
      nodes. Leave the unselected nodes equal to zero         */

    /* This is the constant vector for the linear equation     */
    C = (double *)calloc(U.cols, sizeof(double));
    for (i=0;i<U.cols;i++){
        C[i] = 0.0;
        for (j=0;j<a.rows;j++)
           C[i] += a.el[j][selected[i]] * y[j];
    }

    /* printmatf("%1.4f",U);                                   */
    L = transpose(U);
    freeMatrix(U);

    /* Now solve the system of linear equations by method of
       forward substitution, and back substitution             */
    if (cholesky_solve(L,C))
           printf("\nERROR FINDING THE SOLUTION\n");
    for (i=0;i<L.rows;i++)
        weight[selected[i]] = C[i];

    /*  Free all the memories                                  */
    freeMatrix(L);
    for (i=0;i<a.cols;i++) free(x[i]);
    free(x);
    free(p);
    free(q);
    free(temp_vec1);
    free(temp_vec2);
    free(selected);
    free(C);

    return(weight);            /* returning the weights       */

}
/*************************************************************
realloc_mat.c

This routine reallocates a matrix to a new size, and returns the new
matrix.  The values in the old matrix are saved in the new matrix
in the same order
```

```
calling sequence:
NEW = realloc_mat(OLD, new_rows, new_cols);
here,
   OLD is the OLD Matrix, and NEW is the new one.
   new_rows and new_cols are the size of the new matrix

************************************************************/
MATRIX realloc_mat(MATRIX A,int new_rows,int new_cols)
{
   MATRIX result;
   int i,j;

   result = initMatrix(new_rows,new_cols);
   for (i=0;i<A.rows;i++)
       for (j=0;j<A.cols;j++)
           result.el[i][j] = A.el[i][j];
   freeMatrix(A);
   return(result);
}
```

# Appendix B
# Preparation of This Document

The author used LaTeX, the high quality typesetting software, to produce this thesis. Also the drawings were produced by XFIG. The XFIG files were then converted into postscript format or LaTeX format if possible to inlcude in the LaTeX files. The graphs of chapter 4 were generated using MATLAB.

# BIOGRAPHY OF THE AUTHOR

Wahid Ahmed was born in Dhaka, Bangladesh on November 30, 1969. He received his Higher Secondary Certificate (HSC) from Notre Dame College, Dhaka in 1986. In 1989 he entered University of Maine majoring in Electrical Engineering, and obtained his Bachelor of Science degree with high distinction in December 1992. He had done research in the field of neural networks, and pattern recognition during last two years of his undergraduate studies. He enrolled in the Master of Science program in Electrical Engineering at the University of Maine in January 1993. He has served as a Teaching/Research Assistant and Assistant Network Administrator in the department of Electrical and Computer Engineering. His current research interests are neural networks, image processing, signal processing, pattern recognition, and robotics. He has co-authored several articles in journals and conference proceedings. He is a candidate for the Master of Science degree in Electrical Engineering from the University of Maine, Orono, in August 1994.

## LIBRARY RIGHTS STATEMENT

In presenting this thesis in partial fulfillment of the requirement for an advanced degree at the University of Maine at Orono, I agree that the Library shall make it freely available for inspection. I further agree that premission for extensive copying of this thesis for scholarly purposes may be granted by the Librarian. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Signature _____

Date _____