ORTHOGONAL LEAST SQUARES ALGORITHM FOR RADIAL BASIS FUNCTION NETWORKS AND ITS COMPARISON WITH MULTI-LAYER PERCEPTRON NETWORKS

By

LI ZHANG

Bachelor of Science
Peking University
Beijing, P. R. China
1986

Master of Arts in Mathematics
The University of Oklahoma
Norman, OK
1994

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1998

ORTHOGONAL LEAST SQUARES ALGORITHM FOR

RADIAL BASIS FUNCTION NETWORKS AND

ITS COMPARISON WITH MULTI-LAYER

PERCEPTRON NETWORKS

Thesis Approved:

J. Chandler

Thesis Advisor

Blayne E. Mayfield

M. Samadzadeh

Wayne B. Powell

Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my advisor Dr. John P. Chandler for his guidance, patience, kindness, and encouragement throughout my studies and finishing my thesis. I am deeply grateful to Dr. Blayne E. Mayfield and Dr. Mansur H. Samadzadeh for their serving on my graduate committee and providing me with invaluable advice and suggestions.

During my graduate study at Oklahoma State University, I benefited very much from the faculty and staff of the Computer Science Department. I would like to thank all the professors for their initiative and intuitive teaching. I also appreciate the opportunity that I could use the facilities of the computer lab at the Computer Science Department as well as other labs on campus.

I am greatly indebted to my parents for their unending love and care throughout my life. Special thanks go to my wife, Dongxiao (Janeen) Zhu, for her love, understanding encouragement, and support, both spiritual and financial. Without her love and support, I would not have completed my study.

I thank Joseph Wang for the program (FORTRAN) used in the last part of simulation (MLP trained by the Newton's method).

Finally, I thank the authors of Gnuplot (The software is copyrighted but freely distributed, see more details at http://www.cs.dartmouth.edu/gnuplot_info.html). They make it easy for me to finish all the graphs in this thesis.

TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION	1
1.1 What is an Artificial Neural Network	1
1.2 A Brief History of Neural Networks	5
1.3 Where Are Neural Nets Being Used?	6
1.4 Radial Basis Function Networks	8
1.5 RBF Networks vs. Multi-Layer Perceptron Networks	10
1.6 Organization of the Thesis	13
II. RADIAL BASIS FUNCTIONS	15
2.1 Interpolation Problem and RBF networks	15
2.2 Approaches for RBF Networks	17
2.3 Kernel Functions for RBF Networks	19
III. ORTHOGONAL LEAST SQUARES ALGORITHM	22
3.1 Least Squares Approximation	22
3.2 QR Factorization	22
3.3 OLS Solutions and QR Factorization	24
IV. OLS ALGORITHM FOR RBF NETWORKS	25
4.1 OLS Algorithm for RBF networks	25
4.2 OLS Implemented by Gram-Schmidt Orthogonal Method	26
4.3 OLS Implemented by Householder Orthogonal Transformation	28
V. SIMULATIONS OF RBF NETWORKS	30
5.1 OLS Procedure	31
5.2 Simulation I	32
5.3 Simulation II	37
5.4 Simulation with MLP Network	39
5.5 Discussion	43
VI. CONCLUSION	45
GLOSSARY	47
REFERENCES	50

LIST OF FIGURES

Figure Page

1. A Typical Neuron Cell........ 2
2. An Artificial Neuron 2
3. A Feed-forward Neural Network........ 4
4. Gaussian Function with Different Smooth Factors........ 20
5. Thin-plate Splines and Cubic Functions........ 21
6. Multiquadric and Inverse Multiquadric Functions........ 21
7. Shape of $F(x_1, x_2) = 1 / (x_1^2 + x_2^2 + 1)$ 32
8. 4 Neurons (Centers) in the Hidden Layer with SF = 1.5........ 33
9. 7 Neurons (Centers) in the Hidden Layer 33
10. NMSE vs. Number of Centers Selected........ 34
11. NMSE vs. Number of Centers Selected 36
12. Shape of $F(x_1, x_2) = sin(x_1 x_2)\, exp(-| x_1 x_2 |)$ 37
13. 4 or 5 Neurons (Centers) in the Hidden Layer 38
14. 4 Neurons (Centers) in the Hidden Layer 38
15. 8 Neurons (Centers) in the Hidden Layer 38
16. MLP Network to Simulate $F_1(x_1, x_2)$ 40
17. MLP Network to Simulate $F_1(x_1, x_2)$ 41
18 MLP Network to Simulate $F_2(x_1, x_2)$ 41
19. MLP Network to Simulate $F_2(x_1, x_2)$ 42

LIST OF TABLES

Table Page

1. Input and Output Layers of RBF Network.. 10
2. RBF Networks vs. MLP Networks.. 12

CHAPTER I INTRODUCTION

1.1 What is An Artificial Neural Network

A *neural network* (NN), more properly referred to as an "*artificial neural network* (ANN)", is a collection of mathematical models which consist of a large number of densely connected simple processing units (called *artificial neurons* or *nodes*). *Links* are the connections between nodes. A *synapse* is a link with a weight. An artificial network uses electronic circuits or software to model biological neurons and their interaction. These ANNs use or simulate many simple interconnected electronic processing elements (nodes) to emulate the interconnected neurons in the brain.

Neural networks are based on simulated nerve cells or neurons that are joined together in a variety of ways to form networks. These networks have the capacity to learn, memorize, and create relationships among data. Although they are described in many different ways, neural networks resemble a very small portion of the human brain in the following two respects [Aleksander and Morton 1990]:

> Knowledge is acquired by the network through a learning process. Interneuron connection strengths known as synaptic weights are used to store the knowledge.

The fundamental structure of computers is based on sequential processing, which has little in common with the human nervous system. It has been known that the human nervous system consists of an extremely large number of nerve cells or neurons (10 billion or 10^{10} neurons in the human cortex, and each of these neurons is connected to about 10^4 others via synapses [Beale and Jackson 90]). These nerve cells operate in parallel to process various kinds of information. Despite the fact that individual neurons operate much slower than today's silicon logical units, the human nervous system can handle complicated tasks such as image processing much more efficiently.

Artificial neurons simulate the function of biological neurons in the human nervous system. Figure 1 shows a simple neuron from the human cortex. Signals enter the neuron from other neurons through the *dendrite* (the input channel of a neuron). If the sum of input signals at a given moment exceeds a certain threshold value, the cell *firing*

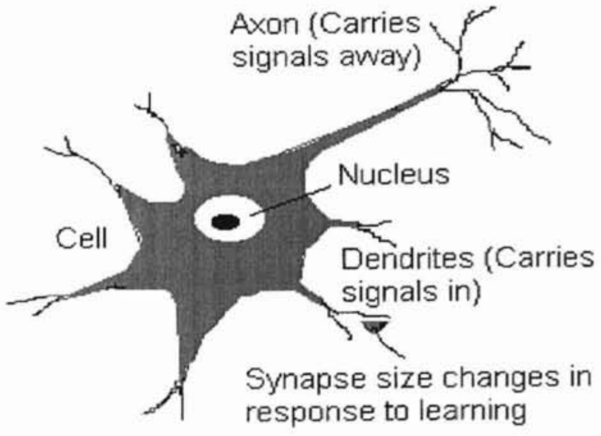

Axon (Carries signals away)
Nucleus
Cell
Dendrites (Carries signals in)
Synapse size changes in response to learning

Figure 1. A typical neuron cell (source: [NewWave 98])

produces an output signal which travels along the *axon* (the output channel of the neuron) and is passed on to other neurons. A synapse connects the axon with the dendrite of another cell, i.e., transfers a signal from one neuron to another.

Figure 2 shows a diagram of an artificial neuron. The input x_i simulates the function of the dendrites to a cell body, and the output connection Y is used to simulate the axon of the cell body. The computation $Y = F(X)$ performed by the neuron simulates the transformation of signals from incoming dendrites to the outgoing axon.

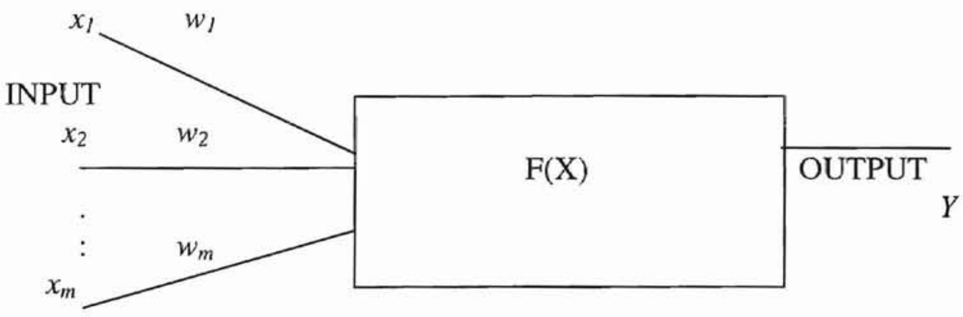


Figure 2. An artificial neuron

Usually there are two kinds of measures for determining the "Similarity" between two vector inputs. The first one is the inner product of the two vectors. The second one is the Euclidean distance. Therefore, the artificial neuron may compute a weighted sum of the input signals $F(X) = \sum_{i=1}^{n} x_i w_i$ (inner product) and pass it through a response function such as the sigmoid function. This type of neuron is usually used in *Multi-Layer Perceptron* (MLP). The neuron may also compute a distance between the input vector and weights vector, $F(X) = [\sum_{i=1}^{n} (x_i - w_i)^2]^{1/2}$ (Euclidean distance) and pass it through a non-linear response (kernel) function. *Radial basis function* (RBF) networks typically use this kind of hidden layer neuron with Gaussian response function [Haykin 94].

The effects of the synapses are represented by "weights", which simulate the effects of the corresponding input signals. The weight w_{ij} connects from neuron j to neuron i. If $w_{ij} > 0$, the firing of neuron j encourages neuron i to fire. If $w_{ij} < 0$, the firing of neuron j discourages neuron i from firing. The greater the magnitude of a weight, the greater the corresponding encouragement or discouragement effect. The weights in a network determine the computational properties of the network. The training of the network is achieved by modifying the weights appropriately.

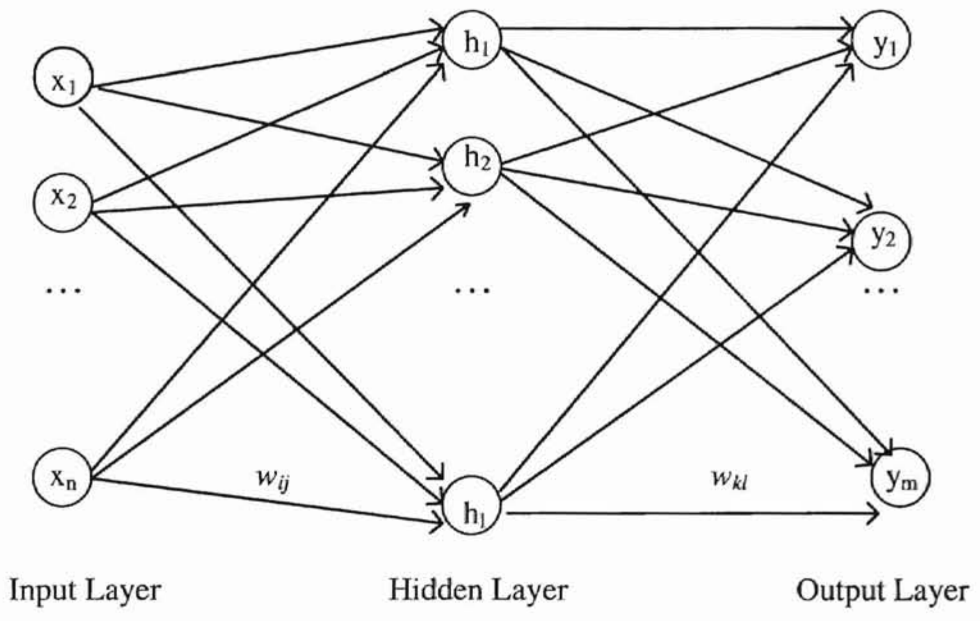


Figure 3. A Feed-forward neural network

A response function represents the nonlinear characteristics of neurons. The neuron impulse is computed as the weighted sum of the input signals, transformed by the transfer function. The learning capability of an artificial neuron is achieved by adjusting the weights in accordance with the chosen learning algorithm.

In feed-forward networks, there are connections only between adjacent layers. Figure 3 shows a typical structure of a feed-forward network, formed by an input layer, one hidden layer, and an output layer. It has two kinds of weights, the weights w_{ij} between the input layer and the hidden layer, and the weights w_{kl} between the hidden layer and the output layer.

The structure of a network includes the number of layers, the number of nodes in each layer, and how nodes are connected to each other.

1. 2 A Brief History of Neural Networks

McCulloch and Pitts [McCulloch and Pitts 43] and Hebb [Hebb 49] first introduced perceptrons to begin neural networks research. Perceptrons were popular in the 1950's and 1960's. But Minsky and Papert [Minsky and Papert 69] demonstrated that perceptrons have very serious limitations when used to compute. In the next two decades, the limitations of neural networks were overcome due to a number of reasons:

- the introduction of hidden layers, and nonlinear activation functions;
- Random, probabilistic, or stochastic methods (e.g., Boltzmann machines) have been introduced [Ackley et al. 85].

Lippmann [Lippmann 87] wrote a fundamental paper to help researchers apply neural networks in their fields. Lippmann gave various types of neural networks for classification problems. A more comprehensive review of neural networks can be found in many neural network books, such as Haykin's [Haykin 94]. The *Multi-Layer Perceptron* (MLP) might be the most popular and extensively studied network. Recently, Radial basis function networks, introduced by Broomhead and Lowe [Broomhead and Lowe 88], have been developed very fast.

1.3 Where Are Neural Nets Being Used?

The study of neural networks is an interdisciplinary area, both in its development and in its application. Here we give the most successful areas of neural networks.

1. Signal Processing: There are many applications of neural networks in the area of signal processing. The neural network is used for suppressing noise on a telephone line in a device known as an ADALINE. The adaptive noise cancellation idea is quite simple. At the end of a long-distance line, the incoming signal is applied to both the telephone system component (hybrid) and the adaptive filter. The ADALINE is trained to remove the noise from the hybrid's output signal [Widrow and Stearns 85].
2. Pattern Recognition: One specific area in which many neural networks applications have been developed is the automatic recognition of handwritten characters. Multilayer networks have been used for recognizing handwritten zip codes [Le Cun et al. 90].
3. Neurocomputers: Neurocomputer designers are exploring the use of virtually every known computing hardware idea, including digital, analog, and hybrid approaches using electronic, optical, acoustic, mechanical, and chemical technologies and

combinations of those technologies. There are several kinds of neurocomputers such like electro-optic, optical, and digital neurocomputers (Mark IV) [Hecht-Nielsen 90].

4. Neurosoftware: Neurosoftware languages are typically built around a model of neural networks. It is used with digital electronic neurocomputer coprocessors operating as process servers to host digital computers. It also can be used to provide neurosoftware descriptions of neural network architectures [Hecht-Nielsen 90].
5. Control: An example of the application of neural networks to control problems, is the task of training a neural "truck backer-upper" to provide steering directions to a trailer truck attempting to back up to a loading dock [Miller et al. 90]. The neural network is able to learn how to steer the truck in order for the trailer to reach the dock, starting with the truck and trailer in any initial configuration that allows enough clearance for a solution to be possible. To make the problem more challenging, the truck is allowed only to back up.
6. Medicine: "Instant Physician" [Hecht-Nielsen 90] is a method to train an autoassociative memory neural network to store a large number of medical records, each of which includes information on symptoms, diagnosis, and treatment for a particular case. When a particular set of symptoms occurs frequently in the training set, together with a unique diagnosis and treatment, the net will usually give the same diagnosis and treatment.
7. Speech Production: one of the most widely known examples of a neural network approach to the problem of speech production is NETtalk [Sejnowski and Rosenberg 86], a multi-layer neural network. NETtalk only requires a set of examples of the written input together with the correct pronunciation for it. The written input includes

both the letter that is currently being spoken and three letters before and after it. After training, the net can read new words with very few errors and the intelligibility of the speech is quite good.

8. Speech Recognition: Several types of neural networks have been used for speech recognition, including Multi-layer networks or Multi-layer networks with recurrent connections [Lippmann 89].
9. Business: Artificial neural networks are applied successfully in many business areas [Harston 90], such as the mortgage assessment work by Nestor, Inc. [Collins and Scofield 88].

1.4 Radial Basis Function Networks

The method of Radial Basis Functions (RBF), a technique for interpolation in a high-dimensional space, has developed fast in the past several years. The basic idea was originally proposed by Bashkirov et al. [Bashkirov et al. 1964]. This method also had been explored by Duda and Hart [Duda and Hart 1973] with a different name - potential function. The first users of the RBF technique in neural networks are Powell and Micchelli [Powell and Micchelli 1985].

RBF provides an alternative tool to learning in neural networks. The major idea is as following:

- avoid unnecessarily lengthy calculations, and
- reduce hardware cost when attempting VLSI implementation of such artificial networks for specific problem solving.

Radial functions are radially symmetric functions: $\boldsymbol{R} \rightarrow \boldsymbol{R}$, defined as:

$$\phi(r) = \phi(\|\boldsymbol{x} - \mathbf{c}_i\|), \qquad \boldsymbol{x},\ \boldsymbol{c}_i \in \boldsymbol{R}^n \text{ and } r \geq 0$$

where $\phi(r)$ is continuous on $[0, \infty)$ and its k^{th} derivative is completely monotonic on $(0, \infty)$ for some integer k; $\boldsymbol{x}$ is the input vector, and $\boldsymbol{c}_i$ is the center of the radial basis function.

The most widely used radial basis function in RBF networks is the Gaussian function:

$$\phi(r) = \exp(-\|\boldsymbol{x} - \boldsymbol{c}_i\|^2 / \sigma^2)$$

where σ is the smoothing factor. The value of σ can affect the training of the RBF networks (See Chapter V for more details).

RBF networks are feed-forward networks with one hidden layer (Figure 3). Typically, an RBF network consists of three layers: an input layer, a hidden layer, and an output layer. The input layer nodes propagate the input values to the hidden layer nodes. The input layer nodes are fully connected to the hidden layer nodes. The connections between input layer and hidden layer specify the set of function centers, which are denoted by w_{ij} in Figure 3. Each hidden layer node computes a distance measure (Euclidean norm) between the input vector $\boldsymbol{x}$ (input node) and the vector $\mathbf{c}_i$ (hidden node). The response function of the hidden neuron is a non-linear transformation ϕ -- the radial basis function. Each hidden layer node is also fully connected to each output layer node. The training of the network will determine the weights w_{kl} between the hidden layer and the output layer. Each output layer node will compute a weighted sum of the outputs (linear transformation) from all the hidden layer nodes.

In order to apply an RBF network for solving a problem, we need to give the structure of the network first. In the other words, we have to specify the number of nodes in each layer and the response function for the hidden layer nodes. It is easy to choose the number of input layer nodes and output layer nodes. For example, in a function approximation, the number of input nodes is equal to the dimension of the independent variable space. If we approximate the function F $(x_1, x_2) = 1 / (x_1^2 + x_2^2 + 1)$ by using a RBF network, the two nodes in the input layer are x_1 and x_2. The number of output nodes is equal to the number of dimensions in the target variable space. In the example above, the output of the node in the output layer is $y \in (0, 1]$. For classification problems, the number of input nodes is chosen to be the dimension of the patterns, and the number of output nodes is usually equal to the number of classes. The selection of the hidden nodes (centers) is the major problem for RBF networks and decides the structure of the networks, i.e., the training of the networks. Table 2 gives the summary of nodes on the input layer and output layer.

Problem	Example	Input Layer Nodes	Output Layer Node
Function Approximation	$F(x_1, x_2) = 1 / (x_1^2 + x_2^2 + 1)$	x_1, x_2	y
Classification	XOR	x_1, x_2	y_1, y_2

Table 1. Input and output layers of RBF network

1.5 RBF Networks vs. Multi-Layer Perceptron Networks

Neural networks have been used in various areas such as biological nervous systems, real-time adaptive signal processors, and physical or chemical processes. In many applications, neural networks were used as "black-box" modeling tools. A network or a number of alternative networks are selected as the candidate models of a process to be modeled. A set of data (training data) is used to determined the proper values of the weights, and then the network is applied to another set of data. A trained network is used as the model of the system for prediction when new inputs are given to the system.

The Multi-Layer Perceptron (MLP) networks are widely use in many areas for the following reasons:

- MLP networks can approximate any continuous function arbitrarily well provided that there are enough neurons [Irie and Miyake 88] [Cybenko 89];
- MLP networks are used and implemented widely in many neural network softwares;
- MLP networks are applied in many areas.

However, there are some disadvantages of MLP networks:

- MLP networks usually require a large number of training data to achieve a certain precision [Baum and Haussler 89];
- MLP networks are non-parametric models, i.e., there is no interpretation of the model parameters (the weights and biases) in relation to the network;
- Algorithms used for training MLP networks are not guaranteed to find the global minimum of the error surface.

The advantages of an RBF network:

- In practice, an RBF network is much easier to train than an MLP network. The RBF networks establish the RBF parameters directly from the data. The weights between the input layer and the hidden layer represent data points in the input space. The weights between the hidden layer and the output layer control the contributions of each hidden node to the output node. For an MLP with one hidden layer, selecting the proper number of nodes in the hidden layer is the only mechanism to regulate the complexity of the model;

Networks	RBF	MLP
Number of hidden layers	One layer	One or more layers
Neurons in hidden layer and output layer	Different model	Same model
Output layer	Linear combination	Nonlinear combination
Global minimum	Guaranteed	Not guaranteed
Bias	No	Yes
Kernel function	Gaussian, Thin plate splines	Sigmoid, Logistic
Implemented by	OLS, Random fixed centers	Newton's method Back-propagation, etc.
Training method	Supervised,	Supervised,
Combination function	Euclidean norm	Inner product

Table 2. RBF networks vs. MLP networks

- While increasing the number of hidden nodes, the complexity of a neural network increases. In RBF networks, optimizing the smoothing factor is easier, faster and more effective than optimizing the number of nodes of a network. In the other words, optimizing the smoothing factor does not need to change the structure of the network and achieves better results (see Chapter V for more details);
- It is known that RBF networks can be trained much faster than MLP networks, and the global minimum is guaranteed in the error surface.

RBF networks still have their own problems. Their performances are different, depending on the architectures and the type of RBF kernels. Finally, we summarize the difference between RBF and MLP networks in Table 2.

1.6 Organization of the Thesis

This thesis consists of six chapters: Overview of neural networks which gives the basic knowledge of the neural networks; The details of RBF networks; Orthogonal Least Squares (OLS) Algorithm; The details to implement OLS by the Gram-Schmidt method or Householder Transformation; Simulations and the comparisons among OLS of RBF networks and MLP networks; and the summary of the thesis.

Chapter I explains the overview of neural networks, its history, where to use it, and the basic idea of RBF networks. Also, it compares RBF networks with MLP networks and gives the advantages and disadvantages of each .

Chapter II discusses the radial basis function and interpolation problem, how to use it to approach neural network problem, i.e. Radial Basis Function (RBF) Networks. Before the end of the chapter, it gives several commonly-used radial basis functions.

Chapter III briefly describes the least squares approximation, QR factorization, and the details about QR decomposition implemented by Gram-Schmidt method and Householder Transformation.

Chapter IV shows the Orthogonal Least Squares (OLS) method, i.e., how to use Gram-Schmidt orthogonalization or Householder Transformation to implement the RBF networks.

Chapter V gives the several interpolation applications solved by OLS for RBF networks and back propagation for MLP networks (programs implemented in C). This chapter also discuss how the smoothing factor affects Normalized Mean Square Error (NMSE) and the centers selection. Finally, comparison between the OLS for RBF networks and the Newton's method for MLP networks is given by solving the same function approximation problems.

Chapter VI briefly summarizes this thesis.

CHAPTER II RADIAL BASIS FUNCTION

A nonlinear function $\phi\,(\boldsymbol{x}, \boldsymbol{c})$ is called a Radial Basis Function (RBF) if it depends only on the radial distance $r = \| \boldsymbol{x} - \boldsymbol{c} \|$, where $\boldsymbol{x}, \boldsymbol{c} \in \boldsymbol{R}^n$ and the norm $\| \bullet \|$ is Euclidean norm. $\boldsymbol{x}$ is the independent variable, and $\boldsymbol{c}$, called center, is a constant.

2.1 Interpolation Problem and RBF Networks

The RBF network is designed to perform a nonlinear mapping ϕ from the input space to the hidden space. Then it performs a linear mapping $\sum_{i=1}^{N} w_i \phi\,(\| \boldsymbol{x} - \boldsymbol{c}_i \|)$ from the hidden space to the output space. Therefore it depends on the theory of multivariable interpolation in high-dimensional space [Davis 63]. The interpolation problem is stated as follows:

Interpolation Problem: Given N different points $\boldsymbol{x}_i \in \boldsymbol{R}^n$ in an n-dimensional real space, and N corresponding numbers $y_i \in \boldsymbol{R}$, find a function $F : \boldsymbol{R}^n \rightarrow \boldsymbol{R}$, that satisfies the interpolation conditions:

$$F(\boldsymbol{x}_i) = y_i, \qquad i = 1, 2, \ldots, N \tag{2.1}$$

In the other words, we are going to construct an interpolating surface $\Gamma \subset \boldsymbol{R}^{n+1}$ passing through all the training data points $\{\boldsymbol{x}_i \in \boldsymbol{R}^n \mid i = 1, 2, \ldots, N\}$. Usually, the surface Γ is unknown and the training data are contaminated with noise. In these cases the function F should pass near the data points, not through them. Accordingly, the training phase and generalization phase of the learning process may be viewed as follows [Broomhead and Lowe 88]:

> The training phase constitutes the optimization of a fitting procedure for the surface Γ , based on known data points presented to the network in the form of input-output examples. The generalization phase is synonymous with interpolation between the data points, with the interpolation being performed along the constrained surface generated by the fitting procedure as the optimum approximation to the true surface Γ .

RBF approach: find a function F

$$F(\boldsymbol{x}) = \sum_{i=1}^{N} w_i \phi(\|\boldsymbol{x} - \boldsymbol{c}_i\|) \tag{2.2}$$

satisfying interpolation condition (2.1), where $\{\phi(\|\boldsymbol{x} - \boldsymbol{c}_i\|) \mid i = 1, 2, \ldots, N\}$ is a set of N nonlinear functions, known as radial basis functions, and the norm $\| \bullet \|$ is the Euclidean norm. The centers $\{\boldsymbol{c}_i \in \boldsymbol{R}^n \mid i = 1, 2, \ldots, N\}$ of the radial basis functions and the weights $\{w_i \in \boldsymbol{R} \mid i = 1, 2, \ldots, N\}$ are to be decided by training.

We can use interpolation condition (2.1) to determine the unknown weights w_i. First, we define interpolation matrix $\Phi \in \boldsymbol{R}^{N \times N}$ as:

$$\Phi = (\phi_{ij})_{N \times N}, \qquad \text{where} \qquad \phi_{ij} = \phi(\|\boldsymbol{x}_i - \boldsymbol{c}_j\|) \tag{2.3}$$

from the above definition, we know that each column ϕ_j of Φ has a fixed center. Then, we can rewrite (2.1) and (2.2) as the following linear system:

$$\Phi \boldsymbol{w} = \boldsymbol{y} \tag{2.4}$$

or

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} \phi_{11}\phi_{12}\cdots\phi_{1N} \\ \phi_{21}\phi_{22}\cdots\phi_{2N} \\ \vdots\vdots\ddots\vdots \\ \phi_{N1}\phi_{N2}\cdots\phi_{NN} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{pmatrix}$$

where $\boldsymbol{w}$ and $\boldsymbol{y}$ represent weight vector and target (output) vector, respectively.

There is a class of radial basis functions that has the following property [Light 92]:

> Let $\boldsymbol{x}_1, \boldsymbol{x}_2 \ldots, \boldsymbol{x}_N$ be distinct points in $\boldsymbol{R}^N$. Then the $N \times N$ interpolation matrix Φ whose element is $\phi_{ij} = \phi\,(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|)$ is positive definite.

Therefore, a unique solution to the interpolation problem exists and the interpolation matrix depends only on ϕ and the centers $\boldsymbol{c}_i$. We can solve the undefined weights vector $\boldsymbol{w}$ as:

$$\boldsymbol{w} = \Phi^{-1}\boldsymbol{y} \tag{2.5}$$

2.2 Approaches for RBF Networks

From the above analysis, the following two problems are essential for RBF networks:

1. How to determine the structure of a neural network, i.e., the number of the hidden neurons and how to choose these neurons. In other words, how to choose small numbers of centers $\boldsymbol{c}_j$ to solve the linear system under tolerated error.
2. How to choose an RBF kernel function ϕ, such that the column vectors ϕ_j of Φ form a basis in $\boldsymbol{R}^N$.

There are different learning strategies that we can use to design an RBF network, depending on how the centers of the radial basis functions of the network are specified. There are three approaches specified as follows:

1. Fixed centers selected at random: use fixed radial basis functions defining the response functions of the hidden neurons and the location of the centers may be chosen randomly from the training data set [Lowe 89].
2. Self-organized selection of centers: the radial basis functions are permitted to move the locations of their centers in a self-organized fashion: the linear weights of the output layer are computed using a supervised learning method. The self-organized component of the learning process serves to allocate network resources by placing the centers in only those areas of the input space where significant data are present, such as by the k-nearest-neighbor rule [Moody and Darken 89].
3. Supervised selection of centers: the centers undergo a supervised learning process. A natural candidate for such a process is error correction learning, implemented using a gradient-descent procedure that represent a generalization of the least mean square algorithm [Poggio and Girosi 1990].

One method, called scale-based clustering (similar to method 2 above), aims at partitioning data into more or less homogeneous subsets when the priori distribution of the data is not known. Well-known clustering methods like the k-means algorithm use an implicit cost function to yield a desirable cluster [Dubes and Jain 79]. Recently, Chakravarthy and Ghosh [Chakravarthy and Ghosh 96] showed that scale-based clustering can be used in the RBF networks. In this thesis, we will focus on the orthogonal least squares algorithm [Chen et al. 1991].

Usually, dimensionality reduction is used to reduce the dimension from N to M ($< N$). In order to approximate Φ, M columns are selected from Φ and form a new matrix $\underline{\Phi} \in \boldsymbol{R}^{N \times M}$ (How to select the columns primarily depends on the method to implement the

OLS. The details are given in Chapter IV. Then $\underline{\Phi}^T \underline{\Phi} \in \boldsymbol{R}^{M \times M}$ is a basis in $\boldsymbol{R}^M$ and is invertible. Therefore the linear problem (2.4) is reduced to an approximation problem or optimal problem with reduced dimension.

Approximation Problem: Give $\underline{\Phi} \in \boldsymbol{R}^{N \times M}$ and $\boldsymbol{y} \in \boldsymbol{R}^N$ satisfying:

$$\boldsymbol{y} = \underline{\Phi} w + \boldsymbol{e} \tag{2.6}$$

choose an optimal coefficient vector $\boldsymbol{w} \in \boldsymbol{R}^M$ such that the error energy $\boldsymbol{e}^T\boldsymbol{e}$ minimized.

Optimal Problem: Find $\boldsymbol{w}^* \in \boldsymbol{R}^M$ such that for all $\boldsymbol{w} \in \boldsymbol{R}^M$, w^* satisfying

$$\min\, (y_i - \sum_{j=1}^{M} w_i \phi\, (\, \| \boldsymbol{x}_i - \boldsymbol{c}_j \| \,)\,)^2, \qquad i = 1, 2, \ldots, N \tag{2.7}$$

where $1 \leq M \leq N$, the reduced dimension, and N is the number of input data.

There are other approaches to determining the network size. Chen et al. [Chen et al. 1991] described a method to determine the number of neurons in RBF networks based on orthogonal least squares algorithm. The OLS will reduce the size of an RBF network and avoid the ill-condition in other training method [Haykin 94]

> the OLS procedure will generally produce an RBF network whose hidden layer is smaller than that of an RBF networks with randomly selected centers, ... the problem of numerical ill-conditioning encountered in the random selection of centers method is avoided.

2.3 Kernel Functions for RBF Networks

In order to specify the property of the kernel functions, Micchelli [Micchelli 86] gave two sufficient conditions for choosing an RBF kernel function. From the results of Micchelli, the functions given below satisfying the sufficient conditions:

1. Gaussian function: $\phi(r) = \exp(-(r/\sigma)^2)$

The Gaussian function is widely used in neural network research [Broomhead and Lowe 88]. σ, called the smoothing factor, decides the width of Gaussian function. The Gaussian function, among the radial basis functions, is the only one with the factorizable property [Poggio and Girosi 90]. In the other words, a multidimensional Gaussian function can be decomposed into a product of several lower-dimensional Gaussian functions. The Gaussian function with different smooth factors is showed in Figure 4.

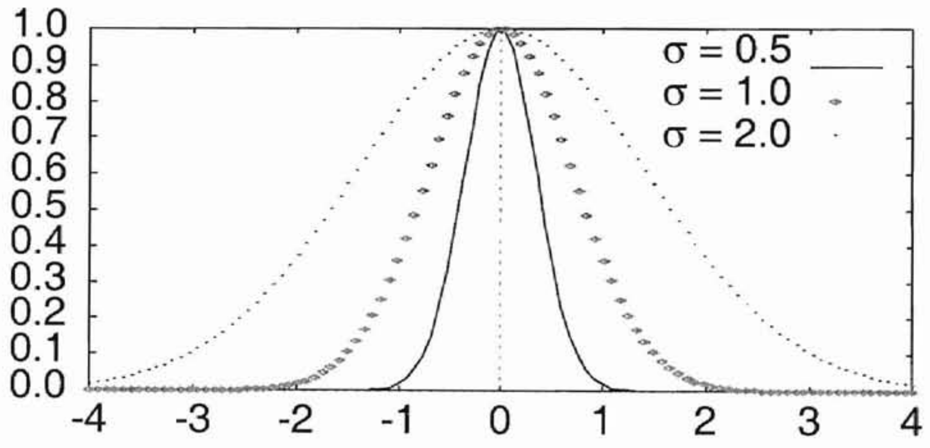


Figure 4. Gaussian function with different smoothing factors

2. The thin plate splines function: $\phi(r) = r^2 \log r$

This kind of function was first used to minimize the bending energy of a "thin plate" [Duchon 76]. There is no user-specified smoothing factor to be concerned with. The thin plate splines function is showed in Figure 5.

3. The cubic function: $\phi(r) = r^3$, shown in Figure 5 with larger dots.

4. Multiquadric function: $\phi(r) = \sqrt{r^2 + c^2}$, shown in Figure 6 (left).

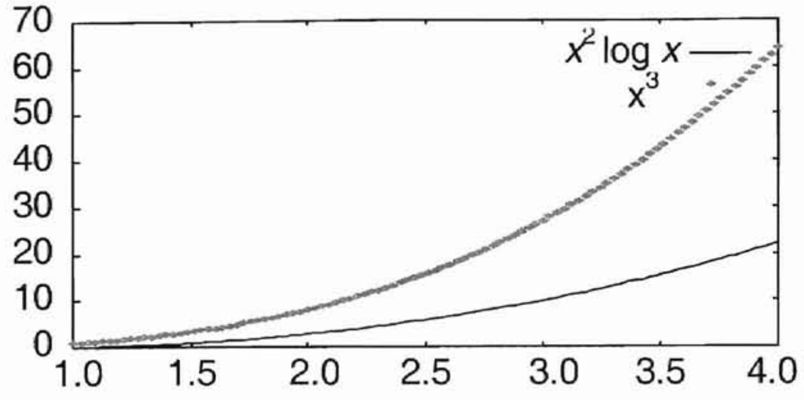


Figure 5. Thin-plate Splines and Cubic Functions

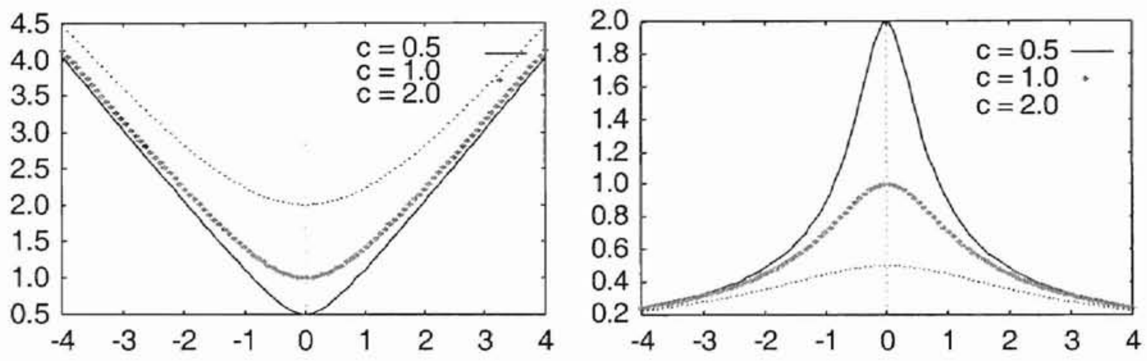


Figure 6. Multiquadric and Inverse Multiquadric (left) functions with different *c*

5. Inverse multiquadric function: $\phi(r) = 1 / \sqrt{r^2 + c^2}$ is shown in Figure 6 (right). The shape is rather similar to the Gaussian function, but it is more sensitive to the smooth factor *c*.

CHAPTER III ORTHOGONAL LEAST SQUARES ALGORITHM

The Orthogonal Least Squares algorithm (OLS) is a very popular method in science today. There are many textbooks [Jacob 95] [Gill et al. 91] giving the details of this method.

3.1 Least Squares Approximation

Least Squares Approximation is a very popular and powerful method to solve linear problem today.

Definition 3.1: Let A be an n × m matrix. A vector $\boldsymbol{x} \in \boldsymbol{R}^m$ is called a least squares solution to the system $A\boldsymbol{x} = \boldsymbol{b}$, if the distance

$$\|A\boldsymbol{x} - \boldsymbol{b}\| = (A\boldsymbol{x} - \boldsymbol{b}, A\boldsymbol{x} - \boldsymbol{b})^{1/2} \tag{3.1}$$

in $\boldsymbol{R}^m$ is a minimum among all possible choices for $\boldsymbol{x}$.

Theorem 3.1: Let A be an n × m rank n matrix, then the least squares solutions to the system $A\boldsymbol{x} = \boldsymbol{b}$ are the solutions to the system $A\boldsymbol{x} = A(A^TA)^{-1}A^T\boldsymbol{b}$. This equation has the unique solution $\boldsymbol{x} = (A^TA)^{-1}A^T\boldsymbol{b}$.

3.2 QR Factorization

There are several methods to decompose the matrix A into a product of an orthogonal matrix and an upper triangular matrix. Here two of them, the Gram-Schmidt orthogonalization method and the Householder Transformation method, are briefly introduced as follows.

Theorem 3.2 (Gram-Schmidt Theorem): Suppose that $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_m$ are mutually orthogonal and nonzero vectors in $\boldsymbol{R}^n$. Suppose $\boldsymbol{v} \notin \text{span}\{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_m\}$. Define

$$\boldsymbol{w}_{m+1} = \boldsymbol{v} - \sum_{j=1}^{m} \frac{(\boldsymbol{v}, \boldsymbol{w}_j)}{(\boldsymbol{w}_j, \boldsymbol{w}_j)} \boldsymbol{w}_j \tag{3.2}$$

Then the vectors $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_m, \boldsymbol{w}_{m+1}$ are mutually orthogonal. We have, moreover, $\text{span}\{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_m, \boldsymbol{v}\} = \text{span}\{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_m, \boldsymbol{w}_{m+1}\}$.

QR Factorization: Given an n×n rank n matrix A, we can use the Gram-Schmidt Theorem to decompose the matrix A into QR, where Q is an $n \times n$ orthogonal matrix, i.e., $Q^TQ = I$, and R is an $n \times n$ upper triangular matrix.

Denote the columns of A as $\boldsymbol{v}_1, \boldsymbol{v}_2 \ldots, \boldsymbol{v}_n$. Apply the Gram-Schmidt Theorem to the columns of A.

$$\{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_n\} = \{\boldsymbol{v}_1, \boldsymbol{v}_2 \ldots, \boldsymbol{v}_n\}\underline{R}$$

where $\underline{R}$ is an $n \times n$ upper triangular matrix from the coefficients from (3.1).

Therefore we have

$$A = QR$$

where $Q = \{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_n\}$ and $R = \underline{R}^{-1}$.

Theorem 3.3 (Householder Transformation): Let A be an $n \times n$ matrix with rank n. There exists n -1 Householder matrices $H_1, H_2 \ldots, H_{n-1}$, such that

$$H_{n-1}\, H_{n-2} \cdots H_1 A = R$$

where R is an n × n upper triangular matrix. Let $Q^T = H_{n-1}H_{n-2} \cdots H_1$, then Q $= H_1 H_2 \cdots H_{n-1}$. Therefore we have $A = QR$, where Q is an n × n orthogonal matrix.

The Householder matrix is symmetric ($H = H^T$) and orthogonal ($HH^T = H^T H = I$). The Householder matrix only depends on the direction of $\boldsymbol{u}$, the Householder vector. The Householder Transformation has the following format:

$$H = I - \frac{2\boldsymbol{u}\boldsymbol{u}^T}{\|\boldsymbol{u}\|^2} = I - \frac{2\boldsymbol{u}\boldsymbol{u}^T}{\beta}, \qquad \text{where } \beta = \frac{1}{2}\|\boldsymbol{u}\|^2$$

where $\| \bullet \|$ is the Euclidean norm.

3.3 Least Squares Solutions and the QR Factorization

If we apply the above *QR* factorization to an n × m matrix *A*, where m < n, we won't obtain an orthogonal matrix. The n × m matrix *Q* consists of orthonormal columns, and the m × m matrix *R* is an upper triangular matrix.

Let $A = QR$ be a QR-factorization of A, we have

$$A^TA = (Q\,R)^T QR = R^T Q^T QR = R^T R$$

Therefore, the least square solution becomes

$$\boldsymbol{x} = (A^TA)^{-1} A^T\boldsymbol{b} = (R^TR)^{-1} (QR)^T \boldsymbol{b} = R^{-1} (R^T)^{-1} R^T Q^T \boldsymbol{b} = R^{-1} Q^T \boldsymbol{b}$$

The solution of the linear system $A\boldsymbol{x} = \boldsymbol{y}$ can be easily solved by a matrix-vector product, if we get the *QR* decomposition of *A*.

CHAPTER IV OLS ALGORITHM FOR RBF NETWORKS

Since M is very large, optimal subset selection techniques are computationally impractical. We have to use subset model selection. The Orthogonal Least Squares (OLS) algorithm [Chen et al. 91] is an efficient implementation of a subset selection method. When $M << N$, computational requirements can be reduced by a preprocessing scheme based on Gram-Schmidt orthogonalization procedure [Chng 94]. Without loss of generality, only one neuron of the output layer is considered in this thesis.

4.1 OLS Algorithm for RBF networks

The OLS (Orthogonal Least Squares) algorithm for the RBF networks is a supervised training algorithm. While back-propagation and other widely used supervised methods are forms of continuous training, OLS is a form of combinatorial optimization. Rather than treating the RBF centers as continuous values to be adjusted to reduce the training error, OLS begins with a large set of candidate centers (in the training data) and selects a limited numbers of centers that usually gives good training error. The vectors corresponding to the centers selected are orthogonal to each other. If we selected all the test data as centers, the vectors would form an orthogonal basis (the dimension of the space would be equal to the number of test data). For small training data sets, the candidates can include all of the training data. For large training data sets, it is more efficient to use a random subset of the training data or to do a cluster analysis and use the cluster means as candidates.

How to select the centers and how many of them for OLS depends on the method of implementing the QR decomposition and the stopping criteria. In this chapter, we will discuss OLS implemented by Gram-Schmidt Orthogonal Method and Householder Orthogonal Transformation.

4.2 OLS Implemented by the Gram-Schmidt Orthogonalization Method

From (2.6), we know that $\Phi \in \boldsymbol{R}^{N \times M}$, where N is the number of data and M is the number of neurons in the hidden layer. Let an orthogonal decomposition of Φ be QR. The model (2.6) can be rewritten as

$$\boldsymbol{y} = Q\boldsymbol{g} + \boldsymbol{e} \tag{4.1}$$

where

$$\boldsymbol{g} = R\,\boldsymbol{w} \tag{4.2}$$

Since (4.1) is a special case of the approximation problem, due to the orthonormality, its linear least square solution is

$$\boldsymbol{g} = Q^T\boldsymbol{y} \tag{4.3}$$

Denote $\boldsymbol{g} = (g_1, \ldots, g_M)^T$ and Q = $(\boldsymbol{q}_1, \ldots, \boldsymbol{q}_M)$, where $\boldsymbol{q}_i \in \boldsymbol{R}^N$. Since $Q^TQ = I$ and $g_j = \boldsymbol{q}_j^T\boldsymbol{y}$, we have

$$\boldsymbol{y}^T\boldsymbol{y} = \boldsymbol{g}^T\boldsymbol{g} + \boldsymbol{e}^T\boldsymbol{e} = \sum_{j=1}^{M} g_j^2 + \boldsymbol{e}^T\boldsymbol{e} = \sum_{j=1}^{M} (\boldsymbol{q}_j^T\boldsymbol{y})^2 + \boldsymbol{e}^T\boldsymbol{e} \tag{4.4}$$

The key difference between the OLS method and *QR* factoralization is the selection of the centers [Chen et al. 91]. The OLS method selects centers at each step to maximize $g_j^2 = (\boldsymbol{q}_j^T \boldsymbol{y})^2$. The OLS method selects from the remaining $N - i - 1$ choices the values j and ϕ_j such that the resulting $\boldsymbol{q}_j$ gives the largest energy g_j^2 at each step $i = 1, 2, ..., M$. The selection stops when the error energy has been reduced to the given tolerance.

Denoting $d = \boldsymbol{y}^T\boldsymbol{y}$, we have error energy from (4.4)

$$\boldsymbol{e}^T\boldsymbol{e} / d = 1 - \sum_{j=1}^{M} Err_j \tag{4.5}$$

where $Err_j = (\boldsymbol{q}_j^T \boldsymbol{y})^2 / d.$

The center selection procedure is given as following [Chen et al. 91]:

a. *Base Case*: for $1 \le i \le M,$ we normalize every column vector and calculate the error:

$$\boldsymbol{q}_1^{(i)} = \phi_i / \| \phi_i \|$$

$$g_1^{(i)} = (\boldsymbol{q}_1^{(i)})^T \boldsymbol{y}$$

$$Err_1^{(i)} = (g_1^{(i)})^2 / d$$

To find

$$Err_1^{(i_1)} = max\{Err_1^i,\ 1 \le i \le M\}$$

and select

$$q_1 = q_1^{(i_1)}$$

b. *Iteration*: at the kth step where $k \ge 2$, $1 \le i \le M$, $i \ne i_1, ..., i \ne i_{k-1}$. Orthogonalize the column with the previous selected columns.

$$\alpha_{jk}^{(i)} = \boldsymbol{q}_j \phi_i$$

$$q_k^{(i)} = (\phi_i - \sum_{j=1}^{k-1} \alpha_{jk}^{(i)} q_j) / \| \phi_i - \sum_{j=1}^{k-1} \alpha_{jk}^{(i)} q_j \| \tag{4.6}$$

$$g_k^{(i)} = (q_k^{(i)})^T y \tag{4.7}$$

$$Err_k^{(i)} = (g_k^{(i)})^2 / d$$

To find

$$Err_k^{(i_k)} = max \{Err_k^{(i)},\ 1 \le i \le M,\ i \ne i_1, \ldots, i \ne i_{k-1}\}$$

and select

$$q_k = q_k^{(i_k)}$$

c. *Stopping Criterion*: stop the center selection procedure when the following error condition is met.

$$1 - \sum_{j=1}^{M} Err_j < \rho \tag{4.8}$$

for some M , and given error tolerance ρ.

In the center selection, the error tolerance ρ controls how many centers will be selected, i.e., when we will stopping the center selection. As soon as the M centers have been selected and satisfied the stopping criterion (4.8), we have finished the selection of the centers. At the same time, we get the orthonormal matrix $Q = \{ q_1, q_2, \ldots, q_M \}$. On the next step, we will use Q to get the weight vector w by using (4.3) and (4.2):

$$w = R^{-1} Q^T y ;$$

the calculation of w is trivial.

4.3 OLS Implemented by Householder Orthogonal Transformation

In this section, the Householder Transformation (Theorem 3.3) is used to implement OLS.

From (2.7), choose $\underline{\Phi} \in \boldsymbol{R}^{N \times M}$ and $\underline{w} \in \boldsymbol{R}^{M}$ such that

$$\| y - \underline{\Phi}\, \underline{w} \| = \min \| y - \Phi\, w \| \quad \text{for all } \Phi \in \boldsymbol{R}^{N \times M} \text{ and } w \in \boldsymbol{R}^{M}$$

to get the approximation of y in $\boldsymbol{R}^{M}$. The minimum will be achieved if and only if Φ has full column rank, i.e., the columns of Φ are linearly independent to each other.

Let $e \in \boldsymbol{R}^{M}$ and $e = y - \underline{\Phi}\, \underline{w}$, we have

$$\| e \|^2 = \| y - \underline{\Phi}\, \underline{w} \|^2 = \| Q\, y - Q\, \underline{\Phi}\, \underline{w} \|^2 \tag{4.9}$$

where Q is a $M \times N$ orthonormal matrix such that $Q\, \underline{\Phi} = R = \begin{pmatrix} \underline{R}_{M \times M} \\ 0_{(N-M) \times M} \end{pmatrix}$.

Therefore, we have

$$\| e \|^2 = \| z - \begin{pmatrix} \underline{R} \\ 0 \end{pmatrix} \underline{w} \|^2 = \| z - \begin{pmatrix} R\mathrm{w} \\ 0 \end{pmatrix} \|^2 = \| z_1 - \underline{R}\, \underline{w} \|^2 + \| z_2 \|^2$$

where $z = Q\, y$, $z = (z_1, z_2)^T$, $z_1 = (z_1, z_2, \ldots, z_{M-1})^T$ and $z_1 = (z_M, z_{M+1}, \ldots, z_{N-1})^T$. If we choose $\underline{w} = \underline{R}^{-1} z_1$, we have

$$\| e \|^2 = \| z_2 \|^2 = \sum_{j=M}^{N-1} z_j^2 \tag{4.10}$$

From (4.10), the error e depends only on the last $N - M$ elements in the vector $z = Q\, y$. Therefore, when applying the Householder Transformation to RBF networks, the next column is chosen among the unselected columns of Φ in order to maximize the reduction in $\| z_2 \|$.

CHAPTER V SIMULATIONS OF RBF NETWORKS

In order to show the capability of the RBF network modeling, the RBF network is used to solve function approximation problems. First of all, a small set of data from a known function is given to training the RBF network. Then a larger set of data are used to test the RBF network. Finally, the results from RBF network is compared with the actual function values using Normalized Mean Square Error (NMSE):

$$\mathrm{NMSE} = \frac{\sqrt{\sum_{j=1}^{L} (y_i - y'_i)^2}}{\sqrt{\sum_{j=1}^{L} y_i^2}}$$

where L is the number of test data $\boldsymbol{x}_i$ ($i = 1, \ldots, L$), y_i is the actual function (target) value at $\boldsymbol{x}_i$ and y'_i is the output value from the trained RBF network corresponding to the input $\boldsymbol{x}_i$.

The RBF networks can learn the shape of different functions with a small number of training data. The RBF approximations and the original functions will be showed in three-dimensional graphs respectively. We can compare the accuracy of the RBF approximation though these graphs. We also use NMSE on the test data to show how the smoothing factor affects the selection of centers.

5.1 OLS Procedure

In this thesis, the simulation program is written in C. Following the discuss in Chapter IV, the procedure OLS for RBF networks is given as following:

```
Step 0. Set the number of centers selected to $k = 0$.
Step 1. While stop criteria 1- $Error < \rho$ (4.8) is false, do Steps 2 - 13.
    Step 2. Select starting from the first column and set $ind = 0$. While not selecting all
    $M$ centers ($ind < M$), do Step 3 - 12.
        Step 3. If column No. $ind$ is not selected, do Steps 4 - 11.
            Step 4. Initialize the $k^{th}$ column of $Q$, $\boldsymbol{q} = \phi_{ind}$. Calculate the possible $k^{th}$
            column of $R$ and the $k^{th}$ column of $Q$; do steps 5 - 6 ( $i = 0,1,\ldots, k$ -1).
                Step 5. $\boldsymbol{r}[i] = \boldsymbol{q}_i^T \phi_{ind}$
                Step 6. $\boldsymbol{q} = \boldsymbol{q} - \boldsymbol{r}[i] \times \boldsymbol{q}_i$
            Step 7. Get the $k^{th}$ dialog element of $\boldsymbol{r}$, $\boldsymbol{r}[k] = \| \boldsymbol{q} \|$.
            Step 8. Normalize $\boldsymbol{q}$, $\boldsymbol{q} = \boldsymbol{q} / \| \boldsymbol{q} \|$.
            Step 9. Calculate the error $\boldsymbol{q}$ will reduce when it is selected,
            $Err_{ind} = ( \boldsymbol{q}^T \boldsymbol{y} )^2 / \boldsymbol{y}^T \boldsymbol{y}$ .
            Step 10. Set the $k^{th}$ column of $R$ and the $k^{th}$ column of $Q$ as $\boldsymbol{r}$ and $\boldsymbol{q}$
            respectively, corresponding to the biggest $Err_{ind}$.
            Step 11. Replace $ind = ind + 1$.
        Step 12. Else if column $ind$ is selected, set $ind = ind + 1$, and go back to Step 2.
Step 13. Add $Err_{ind}$ to $Error$, and set $k = k + 1$.
Step 14. Print the result into file.
```

From the above procedure, the OLS algorithm selects the number of hidden neurons dynamically, depending on the stop criterion (4.8). As soon as the stop criterion is satisfied, the procedure is finished and the number of neurons at hidden layer is determined.

5.2 Simulation I

In this section, the RBF network is used to simulate a two-dimensional function

$$F_1(x_1, x_2) = 1 / (x_1^2 + x_2^2 + 1) \tag{5.1}$$

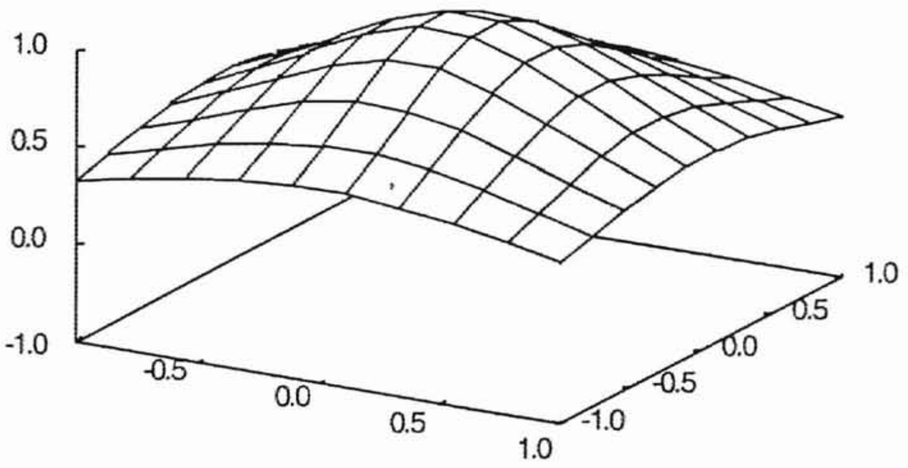


Figure 7. Shape of $F(x_1, x_2) = 1 / (x_1^2 + x_2^2 + 1)$

The shape of $F_1(x_1, x_2)$ is shown in Figure 7. The input layer has two nodes: x_1, x_2. The output layer has only one output $y = F_1(x_1, x_2)$. We test different numbers of nodes in the hidden layer during training. Figure 8 shows the figure for 100 test data after training the

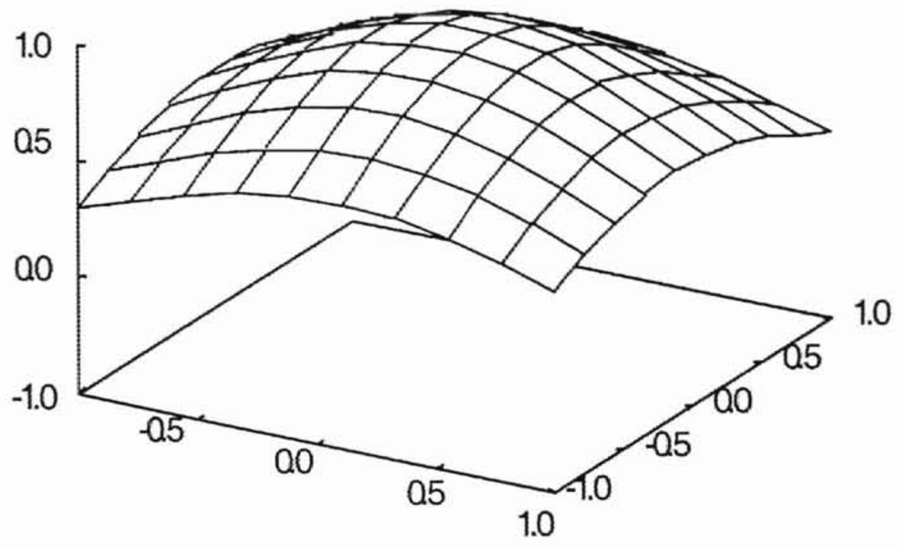


Figure 8. 4 neurons (Centers) in the hidden layer with SF = 1.5
(9 training data and 100 test data)

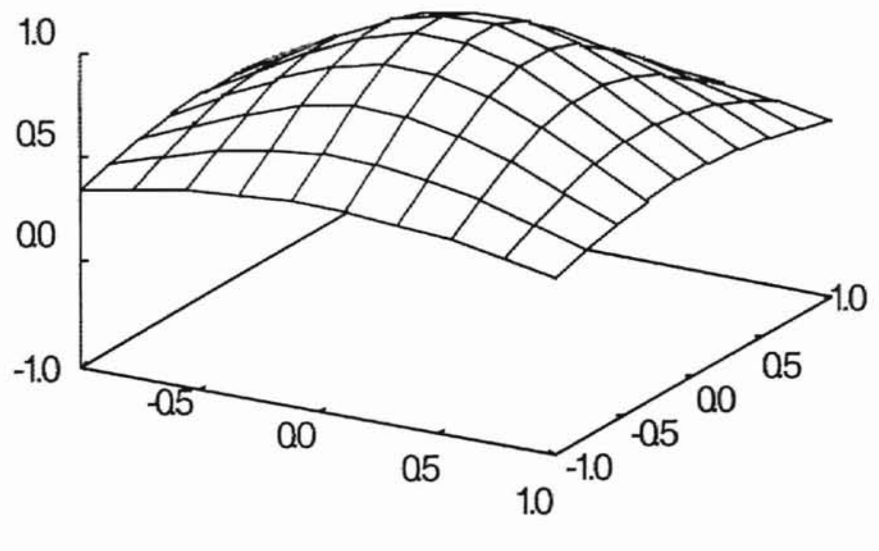


Figure 9. 7 neurons (centers) in the hidden layer with SF =1.0
(25 training data and 100 test data)

RBF network with 9 training data and selecting only 4 centers. Figure 9 shows the figure for the same 100 test data after training the RBF network with 25 data and selecting 7

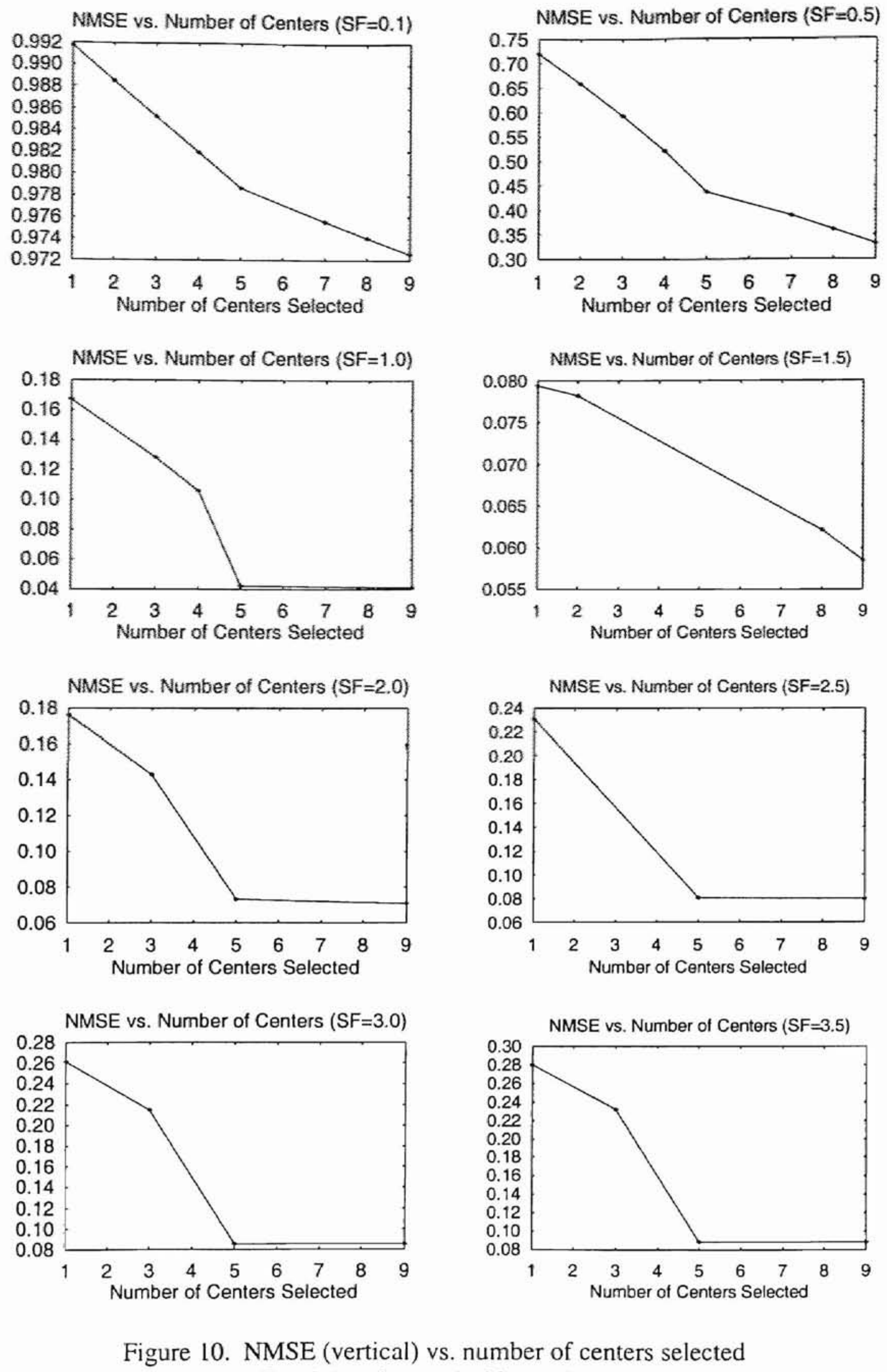
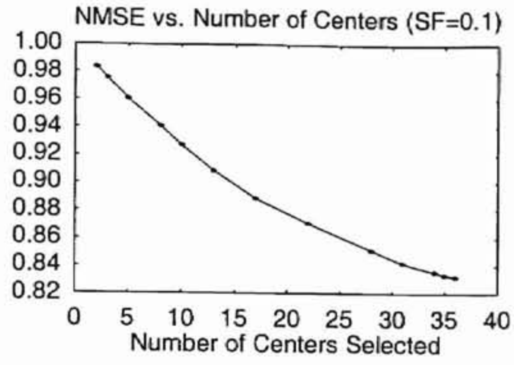
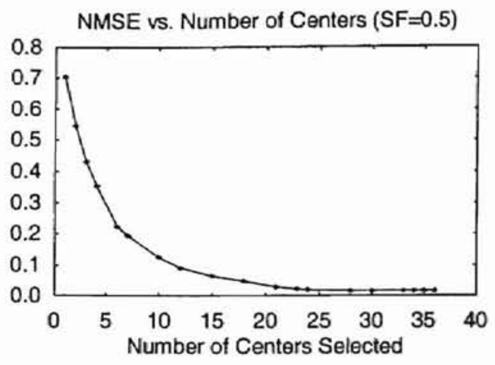
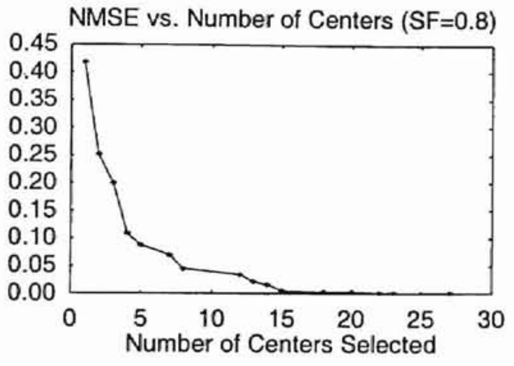
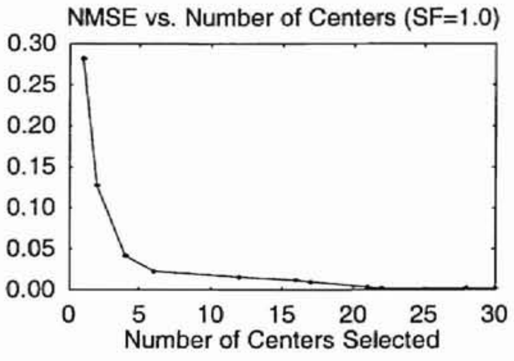


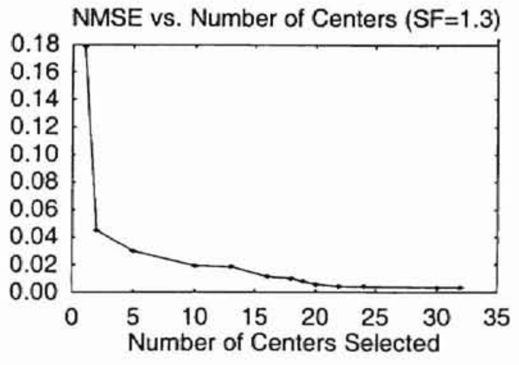
Figure 10. NMSE (vertical) vs. number of centers selected
(9 training data and 100 test data)

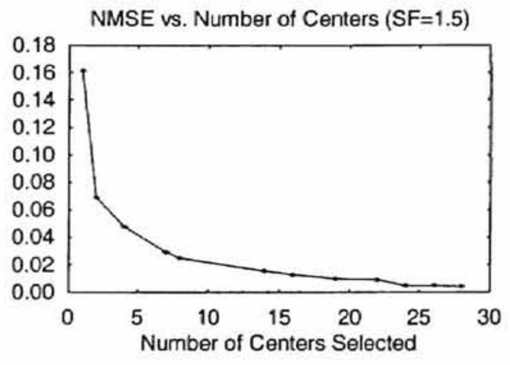

NMSE vs. Number of Centers (SF=0.1)
Number of Centers Selected

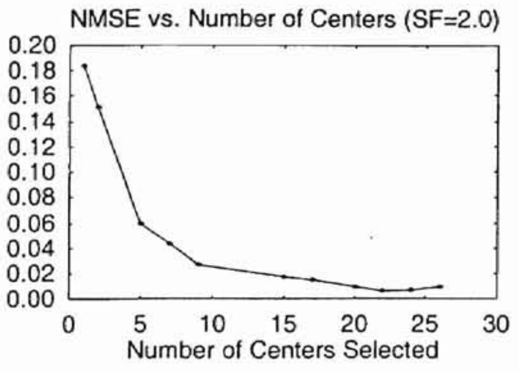

NMSE vs. Number of Centers (SF=0.5)
Number of Centers Selected

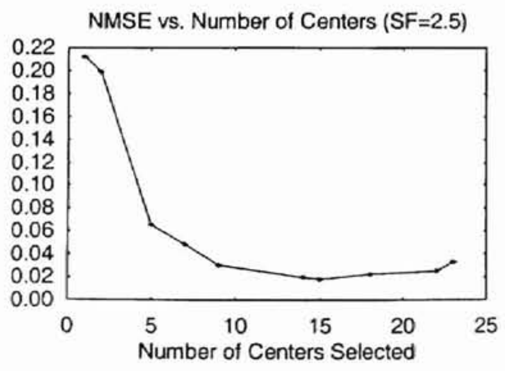

NMSE vs. Number of Centers (SF=0.8)
Number of Centers Selected


NMSE vs. Number of Centers (SF=1.0)
Number of Centers Selected


NMSE vs. Number of Centers (SF=1.3)
Number of Centers Selected


NMSE vs. Number of Centers (SF=1.5)
Number of Centers Selected


NMSE vs. Number of Centers (SF=2.0)
Number of Centers Selected


NMSE vs. Number of Centers (SF=2.5)
Number of Centers Selected

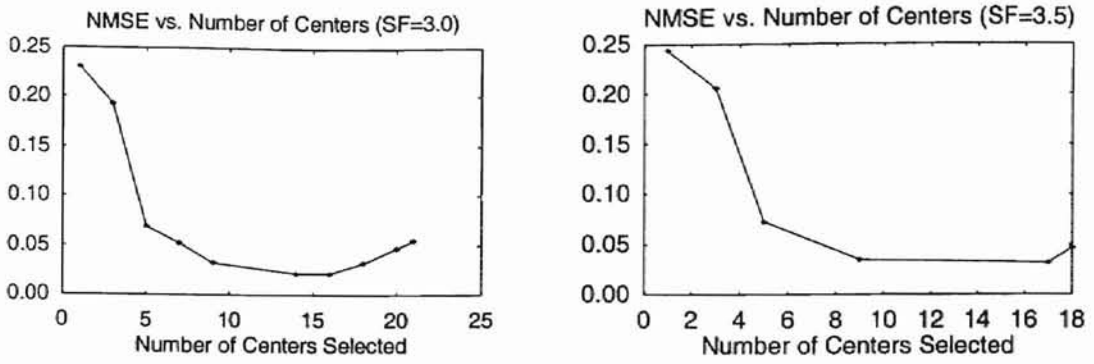


Figure 11. NMSE (vertical) vs. number of centers selected (36 training data and 100 test data)

centers. Comparing these three figures, it is easy to see that the shapes of the simulations (Figure 8 and 9) by the RBF network are very similar to the shape of the original function (Figure 7). The RBF networks were trained only with a small number of data for each simulation and the simulation result is good.

In order to show how the smoothing factor affects the NMSE and the number of the centers selected, a number of graphs were drawn (Figure 10 and 11). Since the stopping criteria for training a RBF network is controlled by $1 - \sum_{j=1}^{M} Err_j < \rho$ (4.8) for a given error tolerance ρ, it is hard to increase the number of centers selected only by one at each training. In the simulation program, we use ρ to control how the number of centers will be selected.

In Figure 10, the RBF network was trained with 9 equidistributed data. The horizontal axis is the number of centers selected. The vertical axis is the NMSE on the 100 equidistributed test data. When a small number as the smoothing factor (Figure 10, SF =

0.1 or SF = 0.5) is chosen, the NMSE for the test data is every large even though all 9 training data are selected as centers. When the smoothing factor is gradually increased, the approximation result is pretty good even though only 2 or 3 centers are selected (Figure 10, SF = 1.0 or 1.5). But when the value of the smoothing factor is increased further, the NMSE increases also (Figure 10, SF = 2.5, 3.0 and 3.5). Therefore for this example, the best value for the smoothing factor is 1.5 to train the network (Figure 8). For 36 training data, the results are similar (Figure 11).

5.3 Simulation II

Another two-dimensional function

$$F_2(x_1, x_2) = sin(x_1 x_2)\, exp(-|x_1 x_2|) \tag{5.2}$$

was used to do a simulation. This function is also simulated with noise, i.e.,

$$F(x_1, x_2) = sin(x_1\, x_2)\, exp(-|x_1 x_2|) + 0.05\mu \tag{5.3}$$

where μ is a random number $\mu \in [-1,1]$.

In this simulation, 9 equidistributed training data (same as in simulation I) are used to train a RBF network. Figure 13 shows the test results for both regular and noise situations while smoothing factor = 1.0. Then 16 equidistributed training data were used to train a RBF network while choosing the smoothing factor σ = 1.5 (If the same method in section 5.2 were used, a good value of the smoothing factor σ is around 2.5). 4 and 8 neurons (centers) at the hidden layer were chosen in each simulation. Figure 14 and 15 show both simulation results.

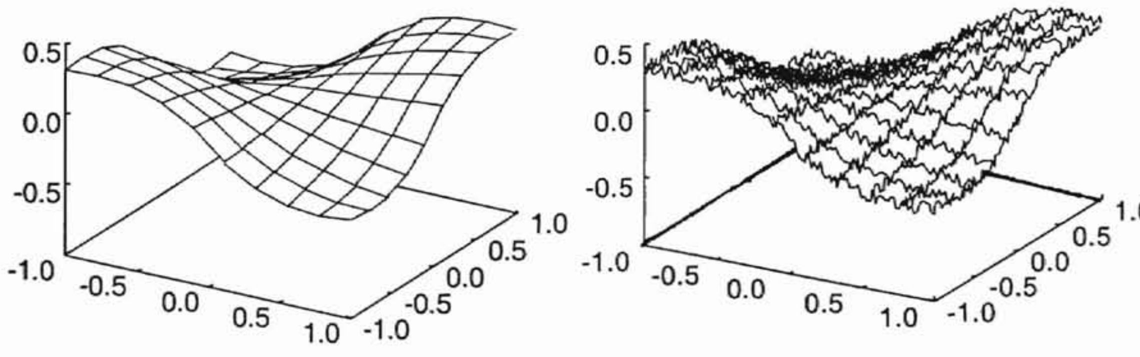


Figure 12. Shape of $F(x_1, x_2) = sin(x_1x_2)\ exp(-|x_1x_2|)$
(right with noise)

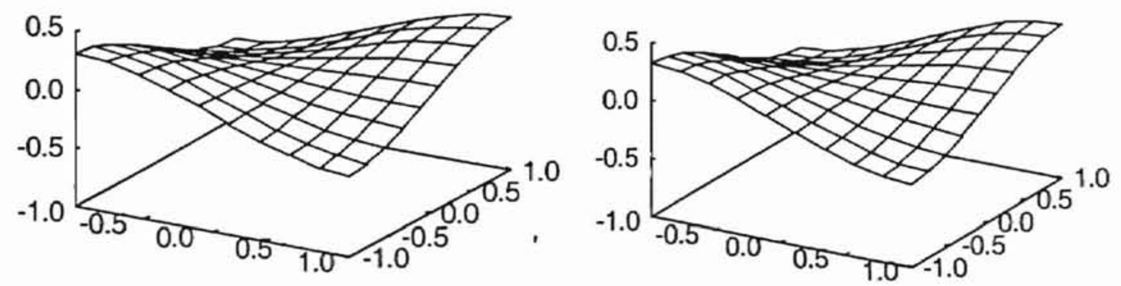


Figure 13. 4 Neurons (Centers) in the hidden layer
(9 training data and 100 test data, right with noise)

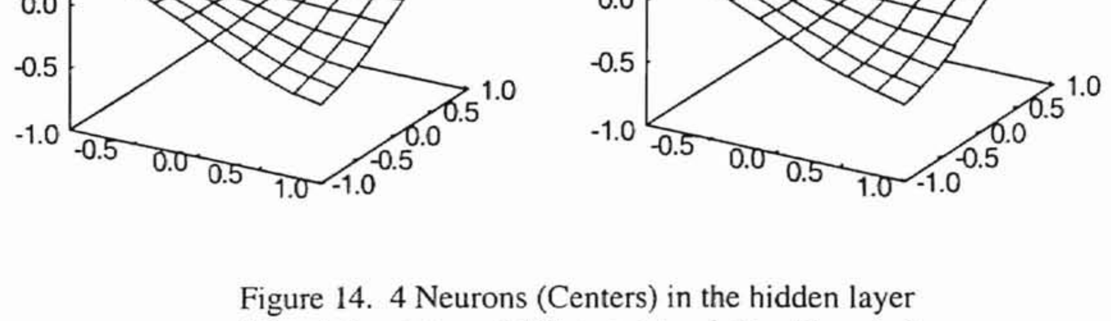


Figure 14. 4 Neurons (Centers) in the hidden layer
(16 training data and 100 test data, right with noise)

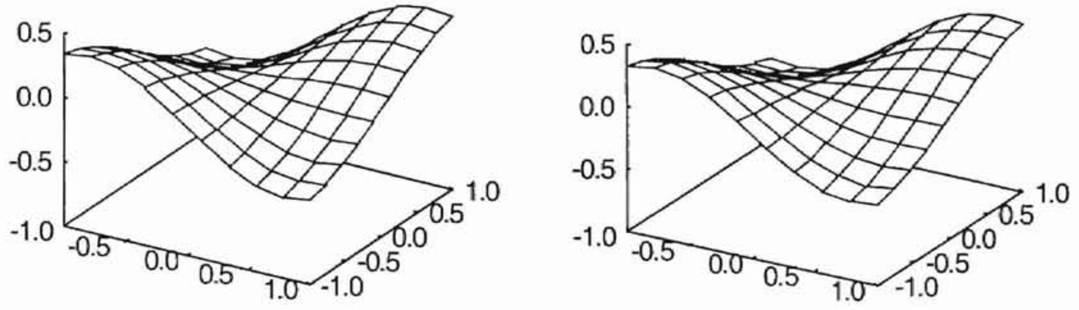


Figure 15. 8 Neurons (Centers) in the hidden layer
(16 training data and 100 test data, right with noise)

5.4 Simulations with MLP Network

In this section, MLP networks were used to do the same two simulations above. the Back-Propagation (BP) algorithm was widely used in the 1980's. It is based on gradient descent: weights are modified in a direction that corresponds to the negative gradient of an error measure. For weights on connections that directly connect to network outputs, this is straightforward and very similar to the Adaline. The gradient of these backward-propagated error measures can then be used to determine the desired weight modifications for connections that lead into these hidden neurons. There are a lot of books [Mehrotra et al. 97] [Haykin 94] on neural networks giving the details of BP.

BP is very slow. The computational requirements for training may be large even for networks of reasonable size. There are several better methods in the literature to accelerate the learning process by reducing the number of iterations required for convergence. Two such methods, Newton's and conjugate gradient methods [Mehrotra et al. 97] [Haykin 94] are most popular.

Newton's method for MLP [Haykin 94] [Mehrotra et al. 97] relies on analytically determining the minimum of a paraboloid that passes through two know points. This method often gives rapid convergence to the minimum, especially if the error surface between the newly generated point and one of its generating points has a roughly paraboloidal shape.

The conjugate gradient method [Shewchuk 94] is an iterative optimization procedure that conducts a special kind of gradient descent in d-dimensional space. A line search is carried out ot find a local minimum along each search direction, and the $(i + 1)$th direction is chosen to be "conjugate" to the first i directions. In the application of the conjugate gradient method to feed-forward neural network training, a series of direction vectors is computed along which modifications are successively made to the weight vector.

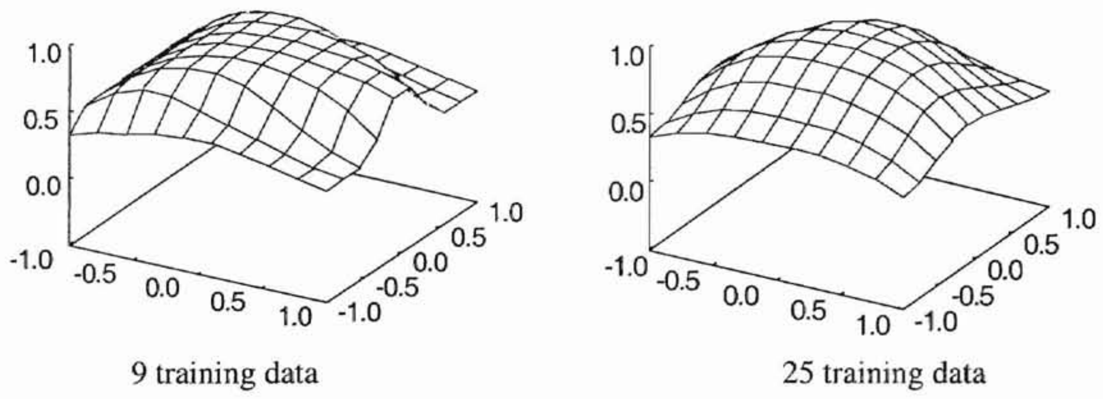


Figure 16. MLP network to simulate $F_1(x_1, x_2)$
(4 neurons at hidden layer and 100 test data)

At the beginning, a MLP network is used to simulate the function $F_1(x_1, x_2)$ (5.1) or $F_2(x_1, x_2)$ (5.2) respectively, and it is trained by back-propagation. The simulation results are not very close to the original functions. Since back-propagation is basically a hill-climbing technique, it runs the risk of being trapped in a local minimum [Haykin 94]. It is undesirable to have the learning process terminate at a local minimum, especially if it is located far above a global minimum. The theoretical work of back-propagation learning to explain the local-minimum is a difficult task to accomplish so far [Haykin 94].

Then the simulation of $F_1(x_1, x_2)$ (5.1) is repeated with a MLP network trained by Newton's method. Figure 16 (left) gives the results from MLP network by using the same training and test data in Figure 8 (9 training data and 100 test data). With a very small number of training data, MLP can learn the this function, but it is not accurate. With the same MLP configuration, the result is much better (Figure 16 right) by increasing the number of training data to 25. If the number of neurons is increased to 8 in the hidden layer, Figure 17 shows smoother graphs.

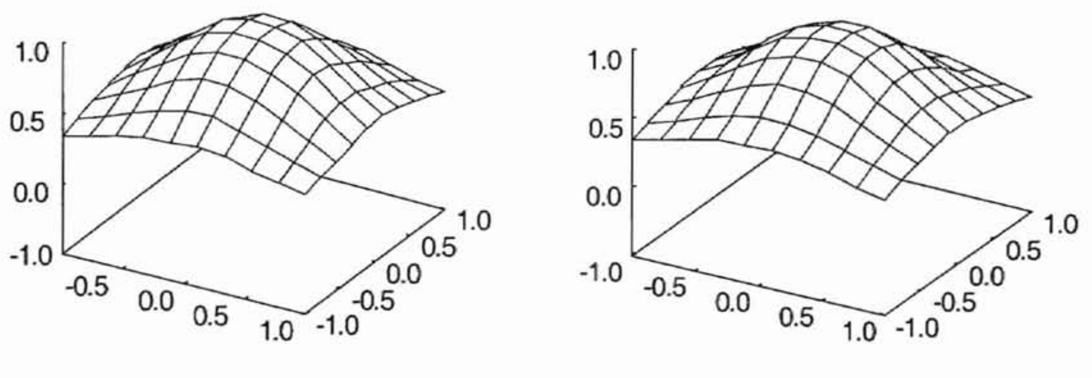


Figure 17. MLP network to simulate $F_1(x_1, x_2)$
(4 neurons and 25 training data, right with noise)

The second function $F_2(x_1, x_2)$ (5.2) is also simulated with an MLP network trained by Newton's method. The same training and test data are used in Figure 13, i.e., equidistributed 9 training data and 100 test data. With 9 training data, Figure 18 (left) shows that the MLP network could learn $F_2(x_1, x_2)$, but it is not very accurate. The right side of Figure 18 is function $F_2(x_1, x_2)$ with noise. When both number of neurons and test data sets increase, the simulation result is much better (Figure 19).

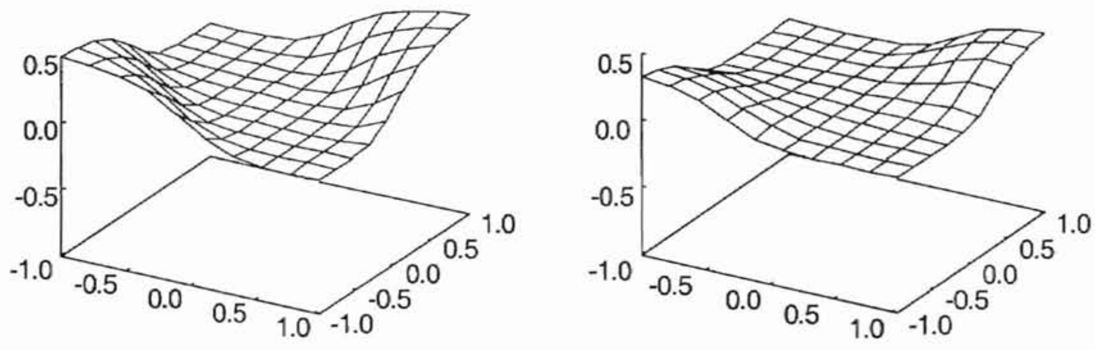


Figure 18. MLP network to simulate $F_2(x_1, x_2)$
(4 neurons and 9 training data, right with noise)

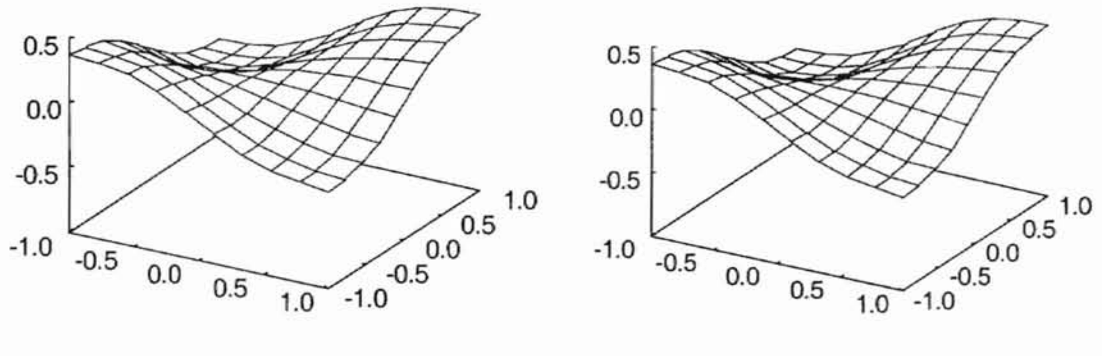


Figure 19. MLP network to simulate $F_2(x_1, x_2)$
(8 neurons and 36 training data, right with noise)

5.5 Discussion

From the above simulations, we can draw the following conclusions:

- The RBF network can approximate the function in the two-dimensional space very well with a relatively small number of neurons in the hidden layer and with a small number of training data (e.g., 2 or 3 centers with 9 training data).
- There always exists one smoothing factor that could achieve a good approximation with fewer centers selected. When this smoothing factor is used, a small number of centers were selected and the network achieved a very small NMSE on the test data (in Figure 11, a best choice of smoothing factor is 1.3 when fewer than six neurons in the hidden layer are used). When a small number of centers is used in the simulation for certain error tolerance, the computation time for training the network will be reduced dramatically.
- When more centers are chosen, the NMSE on the training data will become smaller. But this property is not always guaranteed on test data sets (Figure 11, SF = 2.5, 3.0 and 3.5).
- OLS is a very good method to handle noise. Comparing Figures 13, 14 and 15 with Figure 12, the simulation shows that RBF network can learn the noise well with only small number (4 or 8) of centers selected.
- The configuration of an RBF network is decided dynamically during the OLS training procedure. In other words, the number of neurons in the hidden layer is not fixed before the training procedure; it will be decided when the OLS training procedure is finished. The number of neurons (centers) in the hidden layer is controlled by the error tolerance ρ in (4.8). The configuration of an RBF network is decided

automatically after it has been trained by OLS when a given error tolerance ρ is satisfied.

- The configuration of a MLP network, on the other hand, is given before training. Therefore, the configuration is static during the training. Through the simulation above, MLP with the same configuration of RBF network (i.e., same number of neuron in the only hidden layer), needs more test data to learn the functions.

CHAPTER VI CONCLUSION

In this thesis, OLS for RBF networks is used to solve function approximation problems. Neurons in the hidden layer are selected one-by-one from training data. When adding a hidden neuron (center), the new information contributed is due to that part of its corresponding vector which is orthogonal to the space spanned by the vectors corresponding to the previously added hidden neurons (centers). At each training step, a hidden neuron was selected through optimization such that it, together with previously obtained hidden neurons, would minimize the residual error. The Gram-Schmidt orthogonalization method (or the Householder Transformation method) is used at each step to form an orthogonal basis for the space spanned by the vectors corresponding to the hidden neurons (centers). The orthogonal basis is used both in hidden neuron (center) selection and in weight calculation. Using this technique, it is possible to find the number of hidden neurons required to provide a good representation and hence avoid using an unnecessarily large network.

Through these simulations, it is shown that OLS provides an effective means for developing RBF networks. This training method can be easily used to determine the number of hidden neurons required and the proper weights. The examples show that the OLS for RBF networks provides good modelling capabilities, as well as MLP networks.

They also show that the OLS for RBF networks needs less training data sets than MLP networks, to achieve the same simulation results.

GLOSSARY

Activation Function: A mathematical function that a neuron uses to produce an output. Also called the response function or kernel function.

Artificial Neural Networks: Computers whose architecture is modelled after the brain. They contain idealized neurons called nodes which are connected together in some network.

Artificial Neuron: The primary object in an artificial neural network of human creation. It is a processing element of a network.

Associative Memory: This type of memory is not stored on any individual neuron but is a property of the whole network. This is very different from conventional computer memory where a given element of memory is assigned a unique address which is needed to recall that memory element.

Autoassociative: Making a correspondence of one pattern or object with itself.

Axon: The part of a biological neural cell that contains the dendrites, connecting this neural cell to other cells. The incoming stimulation of a neural cell is transported from the cell's core through the axon to the outgoing connections.

Back-propagation: A learning algorithm used by neural nets with supervised learning. The errors at the output layer are propagated back to the layer before in learning. if the previous layer is not the input layer, then the errors at this hidden layer are propagated back to the layer before.

Bias: A value added to the activation of a neuron. Its purpose is to generate different inputs for different input patterns given to the net.

Conjugate Gradient Method: An optical method to smooth the decent to an error minimum which incorporates memory of past search directions into the formation of each sequential weight update cycle for a neural network.

Dendrite: The connections between biological neural cells. Electrical stimulation is transported from cell to cell using these connections.

Feed-forward: A specific connection structure of a neural net, where neurons of one neuron layer may only have connections to neurons of later layers. An example of such a net type is the Multi-Layer Perceptron Networks.

Forward-propagation: The output values of a neural net's neurons are only propagated through the net in one direction, from the input layer to the output layer.

Gaussian Function: A very famous function, used in FBF networks as kernel function: $\phi(r) = \exp(-r^2 / \sigma^2)$.

Kernel Function: See activation function.

Hidden Layer: An array of neurons positioned in between the input and output layers.

Input Layer: An array of neurons to which an external input or signal is presented.

L_2 function: A function that is square-integrable on the given domain.

Layer: An array of neurons is positioned together in a network.

Learning Algorithm: A mathematical algorithm that a neural net uses to solve specific problems. The process of finding an appropriate set of connection weights to achieve the goal of the network operation.

Multi-Layer Perceptron (MLP): A feed-forward neural net, built of an input layer, at least one hidden layer and one output layer.

NetTalk: A perceptron-type network capable of reading aloud English text with the aid of a voice synthesizer.

Neural Network: A collection of processing elements arranged in layers, and a collection of connection edges between pairs of neurons. Input is received at one layer, and output is produced at the same or at a different layer.

Neuron: An element of a neural net's neuron layer.

Noise: Distortion of an input.

OLS: For a statistician, "Ordinary Least Squares" (as opposed to weighted or generalized least squares), which is what the neural networks literature often calls "LMS" (Least Mean Squares). For a neural networker, "OLS" means "Orthogonal Least Squares", which is an algorithm for feed-forward regression proposed by Chen et al. [Chen et al. 91] for training RBF networks.

Output: A value or a set of values (pattern), generated by the neurons of a neural net's output layer. Used to calculate the current error value of the net.

Output Layer: The last layer of a neural net, which produces the output value of the net.

Pattern Recognition: Ability to recognize a given sub-pattern within a much larger pattern. Alternatively, a machine capable of pattern recognition can be trained to extract certain features from a set of input patterns.

Perceptron: An artificial neural network capable of simple pattern recognition and classification tasks. It is composed of at least three layers where signals only pass forward from nodes in the input layer to nodes in the hidden layers and finally out to the output layer. There are no connections within a layer.

Propagation: The passing of values and errors through the different layers of a neural net during its learning process.

Response Function: See activation function.

Self-organization: A learning algorithm used by the Kohonen Feature Map neural net. During its learning process, the neurons on the net's feature map are organizing themselves depending on given input values. This will result in a clustered neuron structure, where neurons with similar properties (values) are arranged in related areas on the map.

Sigmoid Activation: A specific type of activation function $\varphi(x) = 1 / (1 + \exp(-x))$.

Supervised Learning: A specific type of learning algorithm. The output of the net is compared with a target output. Depending on the difference between these patterns, the net error is computed.

Training: The process of helping in learning by a neural network, by providing desired values corresponding to inputs. In other words, to select a particular structure for a neural network.

Threshold: A specific value that must be exceeded by a neuron's activation function, before this neuron generates an output.

Unsupervised Learning: A specific type of a learning algorithm, especially for self-organizing neural nets like the Kohonen Feature Map. Unlike supervised learning, no target outputs exist.

Weight: A connect between two neurons with a value that is dynamically changed during a neural net's training process.

REFERENCES

[Ackley et al 85] D. H. Ackley, G. E. Hinton and T. J. Sejnowski, "A learning algorithm for Boltzmann Machines", Cognitive Science, 9, 147-169, 1985.

[Anderson and Rosenfield 88] J. A. Anderson and E. Rosenfield, eds., *Neurocomputing: Foundations of Research*, MIT Press, Cambridge, MA, 1988.

[Aleksander and Mortan 90] I. Aleksander and H. Morton, *An Introduction to Neural Computing*, Chapman & Hall, London, UK, 1990.

[Bashkirov et al. 64] O. Bashkirov, E. M. Braverman and I. B. Muchnik, "Potential function algorithms for pattern recognition learning machines." *Automation and Remote Control* 25, 629-631, 1964.

[Baum and Haussler 89] E. B. Baum and D. Haussler, "What size net gives valid generalization?" *Neural Computation* 1, 151-160, 1989.

[Beale and Jackson 1990] R. Beale and T. Jackson, *Neural Computing: An Introduction.* Adam Hilger, Bristol, UK, 1990.

[Broomhead and Lowe 88] D. S. Broomhead and D. Lowe, "Multivariable functional interpolation and adaptive networks," *Complex Systems* 2, 321-323, 1988.

[Charkravarthy and Ghosh 96] Srinivasa Charkravarthy and Joydeep Ghosh, "Scale-based clustering using the radial basis function network", *IEEE Transactions on Neural Networks*, 7(5), 1250-1261, 1996.

[Chen et al. 91] S. Chen, C. F. N. Cowan and P. M. Grant , "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Transactions on Neural Networks*, 2(2), 302-309, 1991.

[Chng et al. 94] E. S. Chng, S. Chen and B. Mulgrew, "Efficient computational schemes for the orthogonal least squares algorithm," *IEEE Transactions on Signal Processing*, 43(1), 373-376, 1994.

[Collins and Scofield 88] E. S. Collins and C. L. Scofield, "An application of a multiple neural networks learning system to emulation of mortgage underwriting judgements," *IEEE International Conference on Neural Networks*, San Diego, CA, II, 459-466, 1988.

[Cybenko 89] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematical Control Signals System* 2, 303-314, 1989.

[Duda and Hart 73] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. John Wiley and Sons, New York, NY, 1973.

[Dubes and Jain 79] R. Dubes and A. K. Jain, "Validity studies in clustering methodologies," *Pattern Recognition*, 11, 235-254,1979.

[Duchon 76] J. Duchon, "Splines minimizing rotation-invariant semi-norms in Sobolev space," In *Constructive Theory of Functions of Several Variables*, Schempp and Zeller (eds), Lecture Notes in Mathematics, volume 571, Springer-Verlag, New York, NY, 85-100, 1976

[Funahashi 89] K. Funahashi, "On the approximate realization of continuous mapping by neural networks," *Neural Networks* 2, 183-192, 1989.

[Gill et al. 91] Philip Gill, Walter Murray and Margaret Wright, *Numerical Linear Algebra and Optimization*, Volume 1, Addison-Wesley Publishing Company, Redwood City, CA, 1991.

[Harston 90] C. T. Harston, "Business with neural networks", in A. J. Maren, C. T. Harston, and R. M. Rap, eds., *Handbook of Neural Computing Applications*, Academic Press, San Diego, CA, 391-400, 1990.

[Haykin 94] S. Haykin, *Neural Networks: A Comprehensive Foundation.* Macmillan College Publishing Company Inc., New York, NY, 1994.

[Hebb 49] D. O. Hebb, *The Organization of Behavior.* Wiley, New York, NY, 1949.

[Hecht-Nielsen 90] R. Hecht-Nielsen, *Neurocomputing*, Addison-Wesley, Reading, MA, 1990.

[Hopfield 82] J. J. Hopfield, "Neural network and physical system with emergent collective computational abilities," *Proceeding of the National Academy of Science of the USA* 79, 2554-2558, 1982.

[Irie and Miyake 88] B. Irie and S. Miyake, "Capacity of three-layered perceptrons," *Proceeding of the IEEE International Conference on Neural Networks* 1, 641-648, 1988.

[Jacob 95] Bill Jacob, *Linear Functions and Matrix Theory*, Springer-Verlag, New York, NY, 1995.

[Le Cun et al. 90] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel, "Handwritten digit recognition with a Backpropagation network," in D. S. Touretsky, ed., *Advances in Neural Information Processing System 2*, San Mateo, CA, Morgan Kaufman, 396-404, 1990.

[Light 92] W. A. Light, "Some aspects of radial basis function approximation," In *Approximation Theory, Spline Functions and Applications* (S. P. Singh, ed.), NATO ASI Series, Vol. 256, 163-190, Boston, MA, Kluwen Academic Publisher, 1992.

[Lippmann 87] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE Transactions on Acoustics, Speech, and Signal Processing* 4, 4-22, 1987.

[Lippmann 89] R. P. Lippmann, "Review of neural networks for speech recognition," *Neural Computation* 1, 1-38, 1989.

[Lowe 89] D. Lowe, "Adaptive radial basis function nonlinearities, and the problem of generalization," *1st IEE International Conference on Artificial Neural Networks*, 171-175, London, UK, 1989.

[McCulloch and Pitts 1943] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics* 5, 115-133, 1943.

[Mehrotra et al. 97] K. Mehrotra, C. J. Mohan and S. Ranka, *Elements of Artificial Neural Networks*, MIT Press, Cambridge, MA, 1997.

[Miccelli 86] C. A. Micchelli, "Interpolation of scattered data: distance matrices and conditionally positive definite functions," *Construction Approximation* 2, 11-22, 1986.

[Miller et al. 90] W. T. Miller, R. S. Sutton and P. J. Werbos, eds., *Neural Networks for Control,* MIT Press, Cambridge, MA, 1990.

[Minsky and Papert 69] M. L. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, MA, 1969.

[Moody and Darken 89] J. E. Moody and C. J. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Computation* 1, 281-294, 1989.

[NewWave 98] NewWave Intelligent Business Systems, Inc., http://sunflower.singnet.com.sg/midaz/Intronn.htm, accessed June 1998.

[Poggio and Girosi 90] T. Poggio and F. Girosi, "Networks for approximation and learning," *Proceedings of the IEEE* 78, 1481-1497, 1990.

[Powell 85] M. J. D. Powell, "Radial basis functions for multivariable interpolation: A review," *Institute of Mathematics and its Application Conference on "Algorithms for the Approximation of Functions and Datas,"* Shrivenham, 1985.

[Rumelhart et al. 86] Rumelhart, D. E, G. E. Hinton and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition,* Vol. 1, eds. D. E. Rumelhart and J. L. McClelland, MIT Press, Cambridge, MA, 1986.

[Sejnowski and Rosenberg 86] T. J. Sejnowski and C. R. Rosenberg, Nettalk: "A Parallel Network That Learns to Read Aloud," The Johns Hopkins University Electrical Engineering and Computer Science Technical Report JHU/EECS-86/01, 1986, Reprinted in [Anderson and Resenfield 88], 663-672.

[Shewchuk 94] J. R. Shewchuk, "An introduction to the conjugate gradient method without agonizing pain," Technical Report CMU-CS-94-125, Carnegie-Mellon University, 1994.

[Widrow and Stearns 85] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1985.

VITA

Li Zhang

Candidate for the Degree of

Master of Science

Thesis: ORTHOGONAL LEAST SQUARES ALGORITHM FOR RADIAL BASIS FUNCTION NETWORKS AND ITS COMPARISON WITH MULTI-LAYER PERCEPTRON NETWORKS

Major Field: Computer Science

Biographical:

Personal Date: Born in Xi'an, Shaanxi, P. R. China, son of Derong Zhang and Xiangui Yang. Married to Dongxiao Zhu.

Education: Received Bachelor of Science degree in Computational Mathematics from Peking University, Beijing, P. R. China, in July 1986; received Master of Science degree in Computational Mathematics from Peking University, Beijing, P. R. China, in July 1991; received Master of Arts degree in Mathematics from the University of Oklahoma, Norman, Oklahoma, May 1994. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in September 1998.

Experience: Employed by Shaanxi Teachers' University, Xi'an, P. R. China as an instructor from July 1986 to August 1988; employed by the Department of Mathematics, Peking University, P. R. China, as a graduate teaching assistant from September 1988 to December 1991; employed by the Department of Mathematics, the University of Oklahoma, as a graduate teaching assistant from January 1992 to May 1994.