FUNCTION APPROXIMATION IN HIGH-DIMENSIONAL SPACES USING LOWER-DIMENSIONAL GAUSSIAN RBF NETWORKS

By

Jones Chui

A Thesis

submitted in partial fulfillment of the requirements

for the Degree of Master of Philosophy

Division of Information Engineering

The Chinese University of Hong Hong

June 1992

UL

360292 thesis
QA
76.87
C473

香港中文大學
大學圖書館系統
★ THE CHINESE UNIVERSITY OF HONG KONG ★
1 - DEC 1993
UNIVERSITY LIBRARY SYSTEM

Acknowledgement

It is a pleasure to express my sincere thanks to my supervisor, Dr M. K. Lai, for his invaluable advice and patient reading of this thesis. Thanks are also given to Mr. F. L. Chung and Mr. Y. C. Chu for many beneficial discussions and suggestions. In addition, I would like to express my deepest gratitude to my family for their understanding and support throughout the past two years.

Abstract

Recently, Gaussian radial basis function (RBF) networks have proven to be useful for approximating continuous functions. For a wide variety of problem domains, the learning speed of RBF networks has been found to be much faster than backprogagation (BP) networks. However, due to the well-known "curse of dimensionality" problem, an excessively large number of RBFs may be required to approximate the function adequately especially in high-dimensional input space which in turn diminish the learning speed advantage of RBF networks. Fortunately, for most real-world problems, only a few relevant input dimensions may be needed to obtain a good approximation. To take advantage of this situation, a new dynamic construction algorithm based on the Sanger's tree structure network is proposed here. This algorithm, which involves a growing as well as a pruning process, builds up the dimensions and the number of Gaussian RBFs incrementally.

The proposed algorithm starts with a RBF network of 1-D Gaussian nodes distributed along each input dimension. During the growing process, new Gaussian nodes which incorporate additional dimensional information are added to the network to reduce the output error. After the growing procedure is completed, the pruning process takes over the control and eliminates the node that has the least contribution to the performance of the network. The entire process halts when the error is reduced to below some desired threshold level. With the above combination of growing and pruning processes, more efficient RBF network structures can be obtained.

Two empirical examples both involving predictions are also given. The first one predicts the dynamics of the chaotic Mackey-Glass equation, while the second one predicts the acoustic waveform of a speech signal. In both cases, the resulting network based on a few relevant dimensions, namely LowD RBFs, are found to be superior in output accuracy and in the reduced number of parameters required than a host of other conventional approaches, including BP, conventional RBFs, and linear prediction technique (LPC).

Contents

Chapter 1

Introduction

When a person hears the voice or catches a glimpse of the face of a familiar person, recognition is instant. Within a fraction of a second after the ear, eye or nose is stimulated, one recognizes the object as desirable or dangerous. The accuracy and speed of such recognition are unmatchable by any of today's man-made machine. Our brain accomplishes this with 10^{11} or 10^{12} interconnected neurons working together. Hence, resemble the structure of human brain system and hereby simulate neuron-like actions become the motivation of many research works for many years.

The beginning of artificial neural networks (ANN) research dates back to 1943 in the pioneer work of McCulloch and Pitts on modeling the simple type neuron activities. During the late 1950s and early 1960s, the invention of Rosenblatt's perceptron and Widrow's ADALINE had made substantial contributions to the development of ANN architectures and implementation concepts. However, the explosion of this field is not ready until the mid 1980s, with the success of various new network models and learning algorithms [12] [16] [28]. Experts from diverse disciplines such as physics, psychology, mathematics, engineering, and computer sciences have been attracted to join the emerging field of ANNs. For the foreseeable future, the bulk of ANN research will play an important role in practical tasks and in the behavioral and brain sciences.

1.1 Fundamentals of Artificial Neural Networks

ANNs are mathematical models of theorized mind and brain activity. ANNs exploit the massively parallel processing and distributed representation properties that are believed to exist in the brain. Typically, ANN processing consists of three elements: (1) a method of processing information, (2) an organized geometry (topology) of interconnected processing units, and (3) a method of encoding (learning) information. To further understand various aspects of ANN system, some of the fundamental issues are explained in detail.

1.1.1 Processing Unit

The primary information processing structure in ANN is the processing unit (PU). A typical PU is shown in figure 1.1. The PU usually has a "state value" or s that is taken to be a linear function of the parameters of the unit itself and of the external input signals,

$$s_j = \sum_{i=1}^{n} w_{ji} x_i + \theta_j \tag{1.1}$$

where s_j is the state value of the jth unit, x_i is the ith external input, w_{ji} is the internal parameter (weight) connected from ith input to the j unit and the term θ associated with input is called threshold of bias. The output of unit j

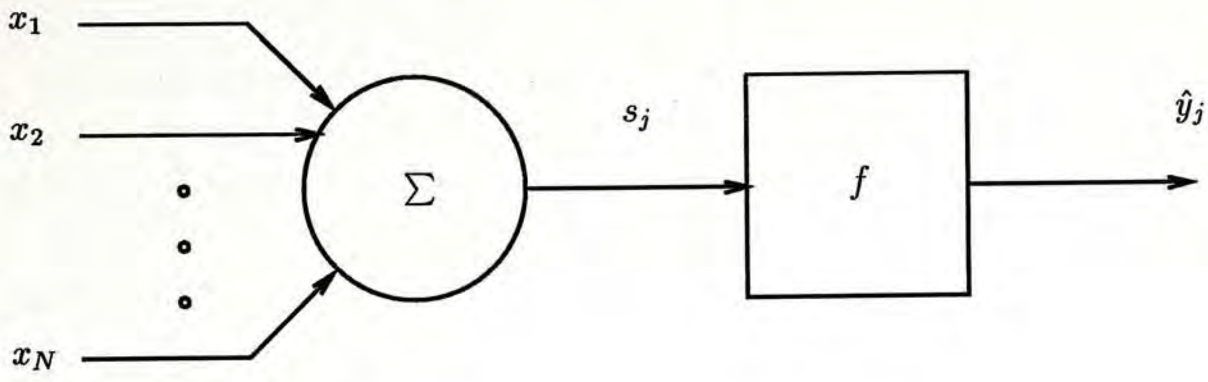

Figure 1.1: Basic processing unit (PU)

($\hat{y}_j$) is typically defined to be a nonlinear function (f) of the state value (s_j). Three common types of nonlinearities as shown in figure 1.2 are: (a) the hard-limiter, (b) the threshold logic, and (c) the sigmoid function. An ANN typically consists of a large number of such PU. Some of them interact directly with the outside environment while others communicate only with their counterparts in the network. Information is distributed over the whole network through the interconnections between them.

1.1.2 Topology

Since computation task and memory storage is shared by a number of PUs operating in parallel, a specific knowledge can be represented by a group (layer) of PUs. An ANN which is hierarchically organized is able to process information in different level of abstractions.

Characteristics of ANN topologies are formed by different connection schemes and layer configurations. Generally, there are three primary PU interconnection

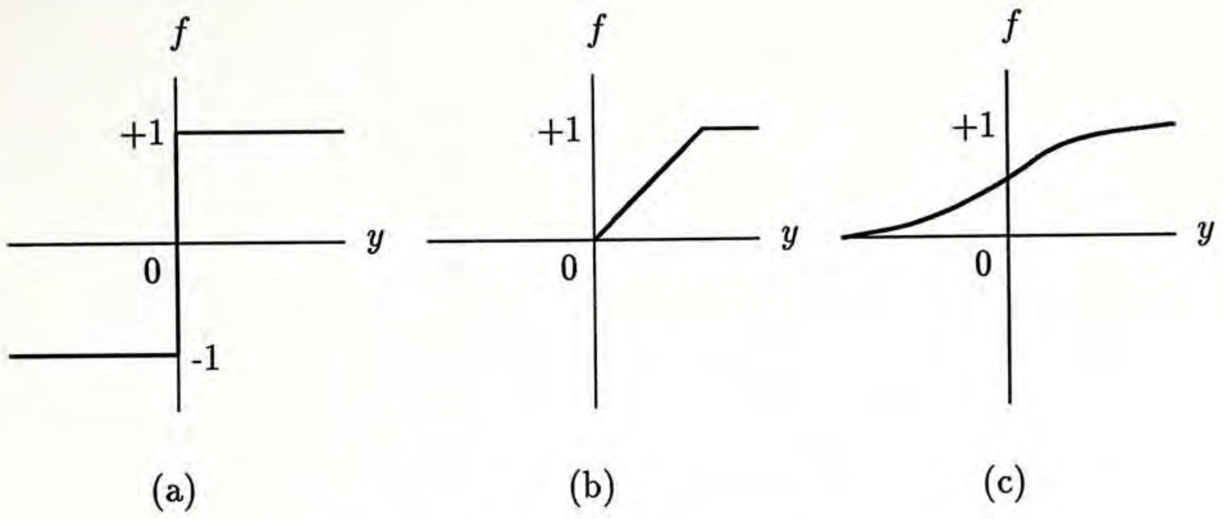

Figure 1.2: Nonlinear activation functions (f)

schemes. They are intra-layer, inter-layer, and recurrent connections. Intra-layer and inter-layer connections are the connections between PUs in the same layer and in different layers respectively. Recurrent connections have the output of PUs looping back to the input of the same PUs.

Layers can also be divided into three types. A layer that receives input signals from the environment is called the input layer, and one that emits signals to the environment is called the output layer. Any layer that lie between the input and the output layers are called hidden layers. Figure 1.3 illustrates one of the common type ANN topology.

1.1.3 Learning Rules

As distinct from the discussion in the previous section, while the network's structure is usually considered fixed, learning rule is defined as the process which modifies the network parameters (e.g. interconnection weights) in order to attain satisfactory system performance in a changing environment. All the learning

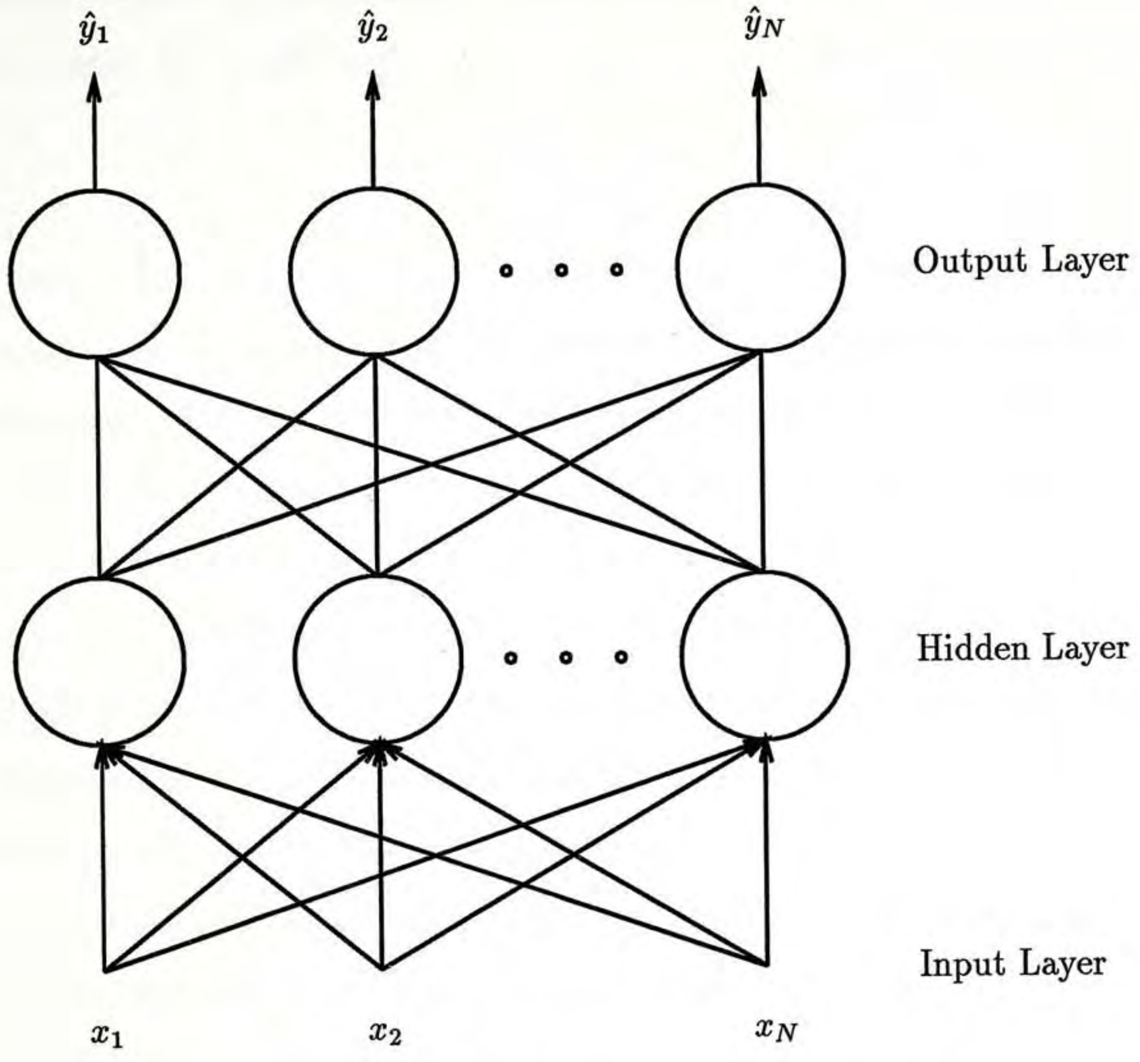

Figure 1.3: Network topology of a three-layer feedforward network

rules can be classified into two categories, supervised learning and unsupervised learning.

In supervised learning, an external teaching signal is required. One of the best example is the least-mean-squared (LMS) algorithm, often called the Widrow-Hoff delta rule. This algorithm minimizes the sum of squares of the linear errors over the set of training samples. The linear error (ϵ) is defined to be the difference between the desired output (y) and the actual output ($\hat{y}$). Having this error signal, the weight in jth PU are then updated according to following rule

$$\Delta w_{ji} = \alpha \epsilon_j x_i \tag{1.2}$$

where $\epsilon_j = (y_j - \hat{y}_j)$, x_i is the ith input, w_{ji} is the weight connection for the jth PU, and α is the learning rate. The choice of α controls stability and speed of convergence. For most practical purposes, α is in the range of $0 < \alpha < 1$.

Quite often, teacher signal is not always available, either because of high cost or lack of knowledge. In this case, an unsupervised learning procedure is necessary. A typical example of this kind of learning is competitive learning [10] [16] [29] which divides a set of multivariate input signals (vectors) into a number of disjoint clusters in such way that the input signals within each cluster are all similar to one another. It is called competitive because there is a set of PUs which compete with one another to become active. There are many variations of the same basic idea, and perhaps the most notable one is Kohonen's topographic maps [15] produced by a modified competitive learning scheme.

1.2 Overview of Various Neural Network Models

Since the first introduction of neural modeling by McCulloch and Pitts in 1943, a large number of ANN models have been developed to tackle different types of practical problems. All these models, based on the fundamental features elaborated above, can be divided into three categories.

In the first category, ANN is functioning as a content-addressable memory (CAM) or associative memory [12] [17]. Such networks store a limited number of pattern samples as interconnection weights which corresponds to a local minimum of an energy function. If a corrupted or distorted sample is presented to the network, it will iterate and hopefully converge to that minimum which retrieve the correct pattern sample in full detail.

Self-organization or the unsupervised learning capability [4] [15] characterizes the nature of ANN in the second category. The network's learning does not require the explicit teaching samples of the function, rather it depends on a task-independent measure of the quality. The network optimizes its internal parameters with respect to that measure and eventually it will reflect the probability distribution of features in the input patterns.

The last category comprises of the feedforward networks or the supervised learning models [1] [28] where the learning can be regarded as performing an input-output mapping from a set of examples. There are many different learning techniques proposed for this type of network. Generally, they can be further divided into two groups: global representation and local representation. For the global representation, learning is based on repeatedly adjusting a small set of global parameters to obtain optimal values. It has the advantage of small memory requirements, but the parameters must often be determined using iterative LMS algorithm which can be slow and are not guaranteed to converge to an optimal solution. The widely used backprogagation (BP) [27] is an example of this kind.

Unlike the compact global models, the local representations often creates a network with a large number of local parameters for table look-up while each parameter only corresponds to a subset of the input region. For example, radial basis function (RBF) [24] networks use many structural locally tuned units so that the optimal solution is more likely to be obtained than BP which uses relatively fewer variables with a larger degree of freedoms.

1.3 Introduction to the Radial Basis Function Networks (RBFs)

In terms of topology, RBF networks can be regarded as a three-layer feedforward network. Input data $\boldsymbol{x}$ are propagated to a single layer of hidden units each of which computes a radially symmetric function of $\boldsymbol{x}$, so that the output (r) of the jth hidden units is given by

$$r_j = g(\|\boldsymbol{x} - \boldsymbol{\mu}_j\|) \tag{1.3}$$

where $\boldsymbol{\mu}_j$ is the center of the RBFs for unit j, and $\| \cdots \|$ denotes a distance measure that is generally taken to be the Euclidean norm. These hidden units encode the inputs by computing how close they are to the centers of the local receptive region. For this, the nonlinear activation function g can be chosen in a variety of ways. For the rest of this thesis we have taken a Gaussian nonlinearity

$$g(x) = e^{-\frac{x^2}{2\sigma^2}} \tag{1.4}$$

The final outputs of the RBFs are the sum of the hidden layer outputs, each weighted by the synaptic strength (w)

$$\hat{y} = \sum_{j=1}^{m} w_j r_j \tag{1.5}$$

The purpose of each output layer weights is to define the contribution of each hidden layer unit to a particular output ($\hat{y}$) of the network.

1.3.1 Historical Development

The notion that feedforward networks compute by using Gaussian hidden units was originally inspired by a review paper on approximation theory, which describes algorithms for multivariable interpolation by Powell [25]. Later on, Lapedes [18] discussed the hypothesis that two layers of sigmoid hidden units produced a "bumps" transfer function in the output space in order to perform approximation. He further pointed out that it might be easier if the weights are synthesised based on "bumps" instead of the original inputs. In other words, it would be advantageous to introduce RBFs to preprocess the inputs. Since then, various forms of RBF networks had been proposed and found successful application in the area of pattern classification and function approximation [3] [22] [23]. In fact, networks which are based on RBFs can outperform BP in various aspects; they are easier to train, are much more predictable, and give intuitively simple solutions. ANN models of this type will be the main theme of the research presented in this thesis.

1.3.2 Some Intrinsic Problems

Although RBF networks are useful for approximating functions in a variety of different domains, they are not free from problems. Due to the localized nature of the RBFs, a very large number of RBFs may be required to approximate an arbitrary function adequately, especially in high-dimensional space. This problem has been known as the "curse of dimensionality" caused by the fact that high-dimensional space is mostly empty. For example, assume that a large number of points is distributed uniformly in the 10-dimensional unit cube. Then the side of a cube containing 5% of the points is $(0.05)^{0.1} = 0.74$. This implies that RBF networks will not be able to pick up small features, unless the number of RBF nodes is gigantic. Besides the heavy size burden imposed by the problem of dimensionality, the calculations required on any of the RBF units increase as the dimension of the input space is high which significantly impair the learning speed advantage of RBF networks.

An efficient way to reduce the computational work of RBF networks is to eliminate the irrelevant variables from the input domain. Note that in some regions of the input space, the desired output function can be approximated using only a few relevant dimensions, which is a very common situation in certain real physical phenomenon. For instance, figure 1.4a shows that the output y is a function of x_1 and x_2. It is clear that y is independent of x_2. Large number of RBF nodes (represented by the small circles) will be saved if we employ only the 1-D RBFs based on relevant variable x_1 as shown in figure 1.4b. However, in many cases, it is impossible a priori for the RBFs to distinguish relevant from irrelevant input dimensions. Technique that can create RBFs to capture low-dimensional features of a single output function is clearly desirable.

1.4 Objective of the Thesis

In this thesis, we address the irrelevant dimension problem by devising a dynamic RBF network construction algorithm based on some heuristic measure

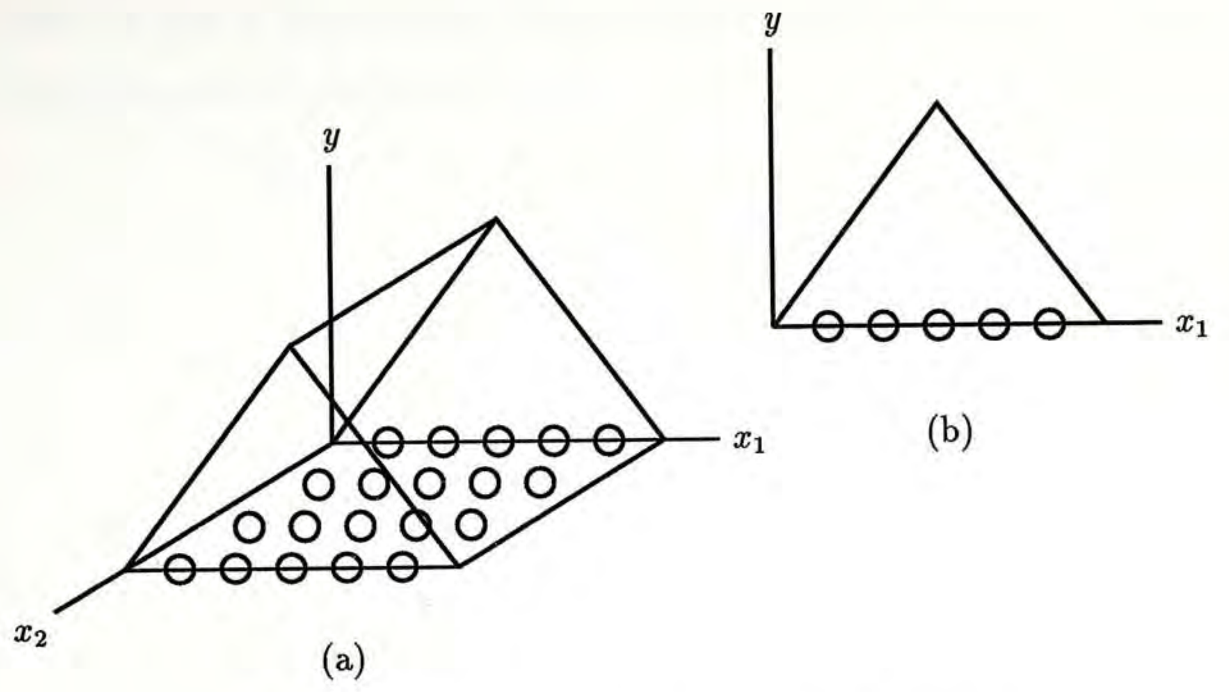

Figure 1.4: Problem of redundant input dimensions

of steepest descent dynamics. Two different approaches guard the evolution of the network growing and pruning. The network's RBF unit is first allowed to grow larger than necessary and then prune it back to yield a smaller and more efficient structure. As the network is grown, some of the RBF units is correlated with more dimensions of the input space. Although the heuristic approaches described here is in no way to guarantee to factor out irrelevant inputs for all the RBF units or even to obtain a optimal network structure, the performance of the suboptimal network using RBFs with fewer input dimensions is surprisingly good as compared to conventional approaches.

This thesis is organized into four chapters. In this introductory chapter, the intrinsic problems in applying RBF networks have been stated. In the next chapter a dynamic construction algorithm based on Gaussian RBFs is presented, along with the detail derivations. In chapter 3, the effectiveness of the resulting

RBF network is demonstrated through various prediction problems. Finally, a brief conclusion is provided in chapter 4.

Chapter 2

Low-dimensional Gaussian RBF networks (LowD RBFs)

In this chapter, a dynamic construction algorithm consisting of network growing and pruning is derived. The algorithm builds a RBF network based on Gaussian nodes of variable input dimension. The structure of the network changes dynamically during learning and is determined by the output function to be approximated.

2.1 Architecture of LowD RBF Networks

2.1.1 Network Structure

The dimensionality of the network elements - RBFs is expandable. Since they will correlate with one input dimension at a time during learning, the synthesis of RBFs in many dimensions may be easier if they are factorizable. It can be easily proven that the only RBF which is factorizable is the Gaussian. A multidimensional Gaussian function can be represented as the product of lower dimensional Gaussians. For example, a n-D Gaussian centered in $\boldsymbol{u}$ can be written as

$$e^{-\|\boldsymbol{x}-\boldsymbol{\mu}\|^2} = e^{-(x_1-\mu_1)^2} \cdot e^{-(x_2-\mu_2)^2} \cdot \ldots \cdot e^{-(x_n-\mu_n)^2} \tag{2.1}$$

where the subscripts $i \ldots n$ denotes the individual dimension for input $\boldsymbol{x} \in R^n$.

With this dimensionality factorization, the network can now start to compute the approximation to the desired output function using a finite set of 1-D Gaussian RBFs. If the function (scalar) y can be determined from only the sum of Gaussian responses of n separable input dimension, then the network output $\hat{y}$ can be expressed in terms of m 1-D Gaussian nodes in each dimension.

$$y \approx \hat{y} = \sum_{i=1}^{n} \sum_{j=1}^{m} w_i^j e^{-\frac{(x_i-\mu_i^j)^2}{2(\sigma_i^j)^2}} \tag{2.2}$$

where w, σ, and μ denotes the height (weight), width, and center of the Gaussian respectively, i indexes the input dimensions and j the 1-D Gaussian nodes. All the parameters of the Gaussian are allowed to train using least mean-squared (LMS) learning algorithm in order to minimize the mean-squared approximation error $\epsilon = \frac{1}{2}(y-\hat{y})^2$. Given sufficient input samples, this algorithm will converge until no further adjustment in w, σ, and μ will improve the approximation.

However, if y does not depend on a linear combination of 1-D Gaussians, then there will be some nonzero residual error $(y - \hat{y})$ which in turn creates considerable average value of weight change variance

$$E[(\Delta w)^2] \tag{2.3}$$

(Refer to section 2.2.1 for more discussions on this heuristic) on certain nodes indicating pressure to incorporate some additional dimension information or "cross-terms". Similar to Sanger's tree-structured network [30] [31] [32], the node with the maximum variance becomes the parent to grow new nodes which include one more input dimension. As shown in figure 2.1, the new 2-D nodes can be generated as the product of the 1-D parent node and the chosen 1-D Gaussians with dimensions different from the parent. In other words, the parent node expands orthogonally to its own dimension axis in a symmetric way. As a result, one of the original 1-D Gaussian will be split into $m(n-1)$ new 2-D nodes and the output $\hat{y}$ is now given by

$$\begin{aligned} \hat{y} &= \sum_{i=1}^{n-1}\sum_{j=1}^{m} w_i^j e^{-\frac{(x_i-\mu_i^j)^2}{2(\sigma_i^j)^2}} + \sum_{j=1}^{m-1} w_k^j e^{-\frac{(x_k-\mu_k^j)^2}{2(\sigma_k^j)^2}} \\ &+ \sum_{l=1,l\neq k}^{n-1}\sum_{j=1}^{m} w_{k,l}^j e^{-\frac{(x_k-\mu_k^j)^2}{2(\sigma_k^j)^2}} e^{-\frac{(x_l-\mu_l^j)^2}{2(\sigma_l^j)^2}} \end{aligned} \tag{2.4}$$

where k denotes the input dimension corresponding to the nodes with the largest variance and l indexes the new 2-D nodes. Note that the structure of expanding the network nodes in terms of symmetric Gaussians is somewhat equivalent to probabilistic neural network (PNN) or "sphere" Gaussian described by Specht [33]. After adding these new 2-D Gaussian nodes, training is then resumed to modify w, σ, and μ to further reduce the error. Such growing and training procedures can be following repeatedly until the output error is reduced below some chosen threshold level.

So far, the growing of new Gaussian units is based on all dimensions that are available except the ones from parent node. Relevant as well as irrelevant input dimensions are included in the expansion. Thus, there is no way to produce a minimal number of Gaussians in the network. However, to filter out the irrelevant inputs can be computationally expensive. The growing process employed here, in fact, sacrifice optimality for simplicity of computation. Unfortunately,

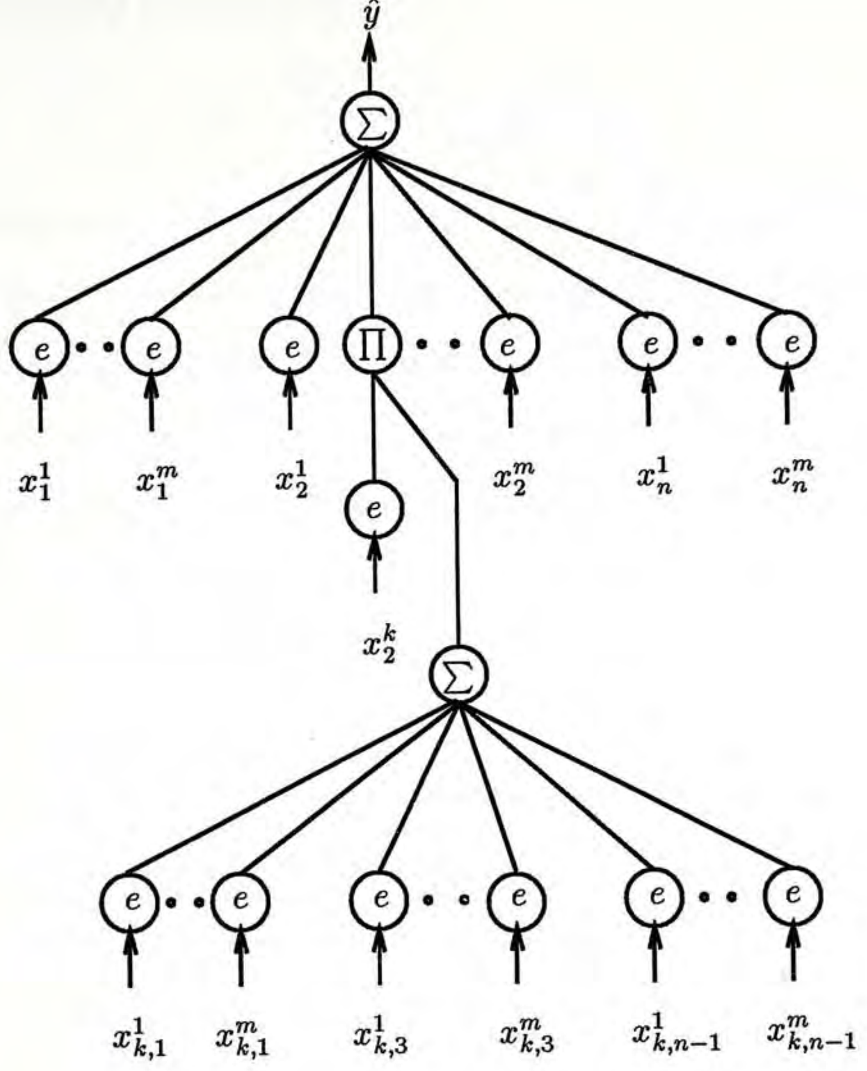

Figure 2.1: Growing structure of the network

the increase in number of nodes is in a maximum factor of $m(n-1)$. There is a great possibility that this may lead to an oversized network which is not only poor in generalization properties, but the speed of convergence is also slow.

To attack this problem, in the way suggested by Karnin [14], a pruning technique is introduced by estimating the slope (sensitivity) of the mean-squared error (ϵ) function with respect to each individual weight (w) in the entired network. Upon the completion of LMS learning, the sensitivity of each node is examined according to the formula

$$s = \frac{E[(\Delta w)^2]}{\alpha} \frac{w(E)}{w(E) - w(S)} \tag{2.5}$$

where the superscripts (S) and (E) denotes the beginning and end weight value right before and after the learning process respectively. (The complete derivation of the above equation will be elaborated in section 2.2.2) Then, the node with the minimum sensitivity is pruned out. By means of this estimation, we can implicitly measure the "redundancy" of each nodes and eliminate the one that has least contribution to the learning of w. More unnecessary nodes will be removed in the subsequent learning process until the network converges to a desired output level.

The description above makes it clear that our approach here is to construct a reasonably large network so that the learning process is successful, then remove some redundant nodes to get a more economical network. Therefore, by incorporating the pruning procedure as the postprocessing step to the growing process, a complete low-dimensional Gaussian RBF network (LowD RBFs) construction algorithm can be formed.

2.1.2 Learning Rules

In order to determine the exact settings of w, σ, and μ in each of the Gaussian nodes, LMS learning algorithm is employed here. Consider only the case of adapting parameter w, the original form of the algorithm can be written as

$$w^{t+1} = w^t - \alpha \frac{\partial \epsilon^t}{\partial w^t} \tag{2.6}$$

or can be expressed in terms of the change of w

$$\Delta w^t = w^{t+1} - w^t = -\alpha \frac{\partial \epsilon^t}{\partial w^t} \tag{2.7}$$

The adaptation cycle index is t. w^{t+1} and w^t indicates the next and present value of w. α is the learning factor. The present error ϵ^t is defined to be the squared difference between the desired function y^t and the approximation obtained from the network $\hat{y}^t$.

$$\epsilon^t = \frac{1}{2}(y^t - \hat{y}^t)^2 \tag{2.8}$$

Applying the steepest descent procedure to error function ϵ, the partial derivatives of ϵ with respect to each of the 1-D Gaussian parameters w, σ, and μ can be obtained as

$$\begin{aligned}
\frac{\partial \epsilon}{\partial w_i^j} &= -(y - \hat{y}) e^{-\frac{(x_i - \mu_i^j)^2}{2(\sigma_i^j)^2}} && (2.9)\\
\frac{\partial \epsilon}{\partial \sigma_i^j} &= -(y - \hat{y}) w_i^j e^{-\frac{(x_i - \mu_i^j)^2}{2(\sigma_i^j)^2}} \left(\frac{(x_i - \mu_i^j)^2}{(\sigma_i^j)^3}\right) && (2.10)\\
\frac{\partial \epsilon}{\partial \mu_i^j} &= -(y - \hat{y}) w_i^j e^{-\frac{(x_i - \mu_i^j)^2}{2(\sigma_i^j)^2}} \left(\frac{x_i - \mu_i^j}{(\sigma_i^j)^2}\right) && (2.11)
\end{aligned}$$

In accordance with equation 2.6 (we now suppress the index t for convenience), the learning rule is then applied to alter parameters of the 1-D nodes with each sample presentation to make an error correction proportional to the partial derivatives.

As the number of nodes grows incrementally, more and more input dimensions are included in the nodes. For input $\boldsymbol{x} \in R^n$, the possible set of Gaussian nodes that can be generated by the construction algorithm is

$$\left\{ w \prod^{i} e^{-\frac{(x_i - \mu_i)^2}{2\sigma^2}} \middle| 1 \le i \le n \right\}$$

Expanding the equation 2.9, 2.10, and 2.11 to adjust the parameters for n-D Gaussians, we have

$$\Delta w^j_{1\cdots n} = -\alpha(y-\hat{y})e^{-\frac{\|\boldsymbol{x}^j-\boldsymbol{\mu}^j\|^2}{2(\sigma^j_{1\cdots n})^2}} \tag{2.12}$$

$$\Delta \sigma^j_{1\cdots n} = -\alpha(y-\hat{y})w^j_{1\cdots n}e^{-\frac{\|\boldsymbol{x}-\boldsymbol{\mu}^j\|^2}{2(\sigma^j_{1\cdots n})^2}}\left(\frac{\|\boldsymbol{x}-\boldsymbol{\mu}^j\|^2}{(\sigma^j_{1\cdots n})^3}\right) \tag{2.13}$$

$$\Delta \mu^j_i = -\alpha(y-\hat{y})w^j_{1\cdots n}e^{-\frac{\|\boldsymbol{x}-\boldsymbol{\mu}^j\|^2}{2(\sigma^j_{1\cdots n})^2}}\left(\frac{x_i-\mu^j_i}{(\sigma^j_{1\cdots n})^2}\right) \tag{2.14}$$

This is equivalent to perform training in the worst case condition - with fully-grown n-D nodes. Since the computational saving of the construction algorithm rests on the assumption that an adequate approximation will not always require all the dimensions of the input data, it is not essential to compute all n Gaussian functions in a node or to learn the parameters corresponding to the redundant input dimensions.

2.2 Construction of LowD RBF Networks

2.2.1 Growing Heuristic

In this section, the development of the heuristics for the network growing and pruning processes is pursued. The ultimate goal is not aimed at producing optimal structure of the network, instead, is to develop a efficient construction algorithm which a competent network can be easily obtained.

In our growing strategy, new Gaussian nodes with correlation of one additional dimension information are added into the network whenever the existing structure is found to be incapable of approximating the desired function. Since optimality is not crucial in the growing algorithm, network parameters like the

香港中文大學圖書館藏書

change of weight (Δw) and the mean-squared error (ϵ) that can be easily acquired along the normal course of LMS learning are employed here to determine where and when to grow new higher-dimensional nodes. This reduces the computational burden imposed by the algorithm.

To understand the choice of growing criteria and their detailed derivation, let us first recall that the weight update equation for 1-D nodes is given by

$$\Delta w_i^j = -\alpha(y - \hat{y})e^{-\frac{(x_i-\mu_i^j)^2}{2(\sigma_i^j)^2}} \tag{2.15}$$

As learning proceeds, the average value of Δw_i^j will tend to zero. If function y depends on a linear combination of the 1-D Gaussian nodes, the error $(y - \hat{y})$ will also approach zero. However, if the nodes do not provide sufficient information to approximate y, then there will be some nonzero error which in turn creates strong tendency to increase or decrease the value of w_i^j. Although the variations of w_i^j with different signs eventually cancel each other, causing the average or expected value $E[\,(\Delta w_i^j)\,]$ tends to zero. Nevertheless, a considerable amount of variance $E[\,(\Delta w_i^j)^2\,]$ will be produced. Such a situation can be well illustrated by a simplified example shown in figure 2.2. Figure 2.2a shows how w_1 of a single 1-D Gaussian nodes based on input x_1 is related to network output $\hat{y}$. In figure 2.2b the desired function y indeed depends on both inputs x_1 and x_2, therefore it is clear that w_1 will fail to converge to a satisfactory solution and the arrow in figure 2.2c indicates the fluctuation of w_1 in the x_1 and x_2 input space.

Because $E[\,(\Delta w_i^j)^2\,]$ provides deficiency information for each individual 1-D nodes, it becomes the main criterion for determining where to grow new nodes. This leads to the following relation between the weight variance and

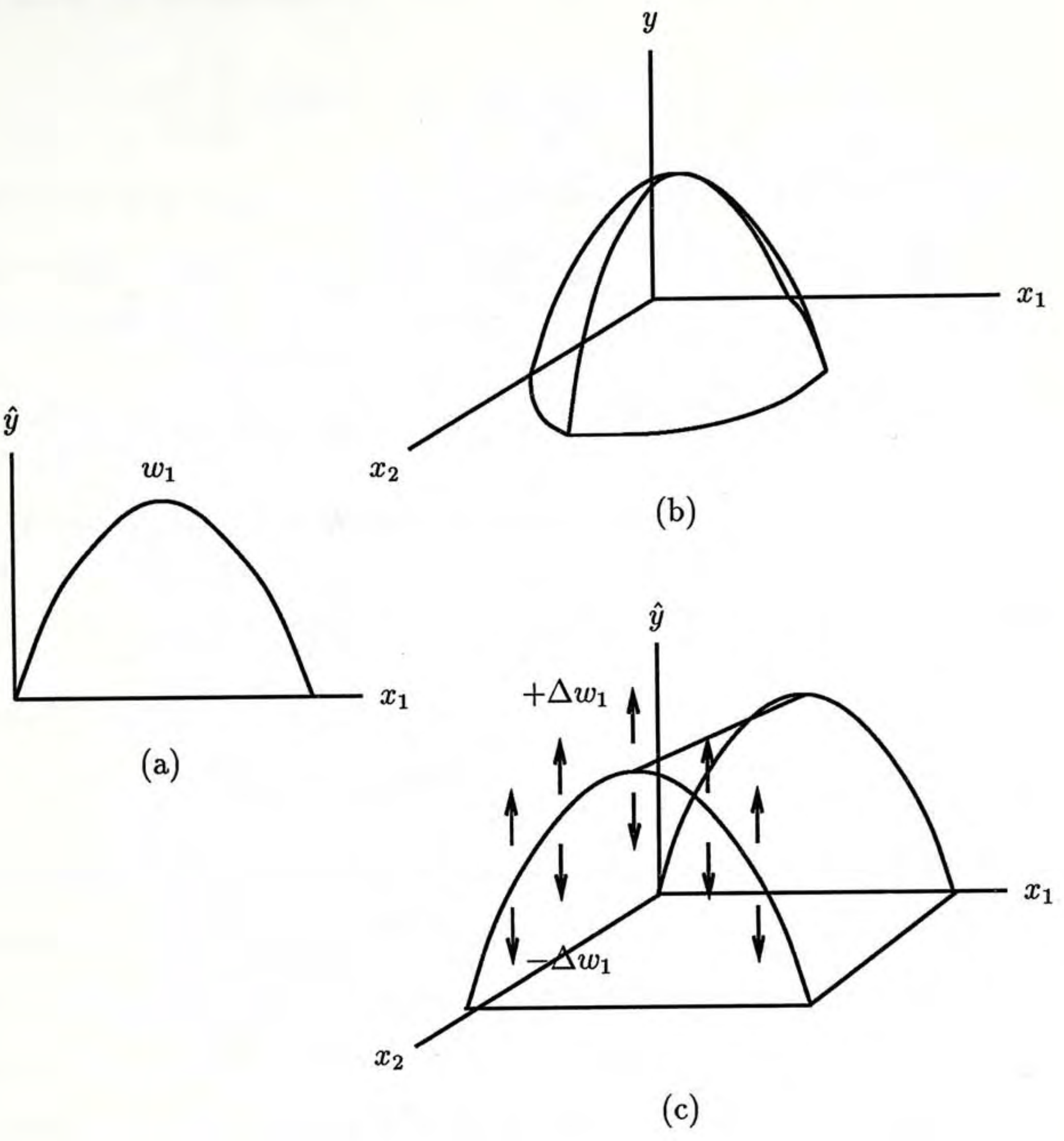

Figure 2.2: Weight change variance problem

mean-squared error.

$$E[\,(\Delta w_i^j)^2\,] = (\alpha^2)E[\,(y-\hat{y})^2(e^{-\frac{(x_i-\mu_i^j)^2}{2(\sigma_i^j)^2}})^2\,] \tag{2.16}$$

Since this is true for all j in i input components, the total variance of all $m\,n$

1-D nodes can be obtained as

$$E[\sum_{i=1}^{n}\sum_{j=1}^{m}(\Delta w_i^j)^2] = (\alpha^2)E[\sum_{i=1}^{n}\sum_{j=1}^{m}(y-\hat{y})^2(e^{-\frac{(x_i-\mu_i^j)^2}{(\sigma_i^j)^2}})] \tag{2.17}$$

For any input $\boldsymbol{x} = [x_1, x_2, \cdots, x_n]$, it is assumed that at least one 1-D Gaussian can respond to one of its components. Thus, the lower bound of the total variance is given by

$$\sum_{i=1}^{n}\sum_{j=1}^{m} E[(\Delta w_i^j)^2] \geq (\alpha^2 n m)E[(y-\hat{y})^2]\, \underset{x_i \in \boldsymbol{x}}{min} \sum_{i=1}^{n}\sum_{j=1}^{m} e^{-\frac{(x_i-\mu_i^j)^2}{(\sigma_i^j)^2}} \tag{2.18}$$

or can be expressed in terms of mean-squared error

$$E[(y-\hat{y})^2] \leq \sum_{i=1}^{n}\sum_{j=1}^{m} E[(\Delta w_i^j)^2]\frac{1}{\delta} \tag{2.19}$$

where

$$(\alpha^2 n m)\, \underset{x_i \in \boldsymbol{x}}{min} \sum_{i=1}^{n}\sum_{j=1}^{m} e^{-\frac{(x_i-\mu_i^j)^2}{(\sigma_i^j)^2}} = \delta > 0 \tag{2.20}$$

From equation 2.19, we see that the error $E[(y-\hat{y})^2]$ will be zero if and only if the total variance $\sum_{i=1}^{n}\sum_{j=1}^{m} E[(\Delta w_i^j)^2] = 0$. Hence, it is advisable to minimize the maximum $E[(\Delta w_i^j)^2]$ for any particular j node in i input such that the error can be reduced in this case.

To decrease the variance of the 1-D node with maximum value, new 1-D Gaussians based on different input components are added into the chosen node as additional second dimensional information. In other words, the original 1-D node of maximum variance are expanded and new 2-D nodes are formed. Again, w_i^j as well as other Gaussian parameters of these 2-D nodes are trained according to the learning equations 2.12, 2.13, and 2.14, so that

$$\Delta w_{k,l}^j = -\alpha(y-\hat{y})e^{-\frac{(x_k-\mu_k^j)^2}{2(\sigma_{k,l}^j)^2}} e^{-\frac{(x_l-\mu_l^j)^2}{2(\sigma_{k,l}^j)^2}} \tag{2.21}$$

$$\Delta\sigma^j_{k,l} = -\alpha(y-\hat{y})w^j_{k,l}e^{-\frac{(x_k-\mu^j_k)^2}{2(\sigma^j_{k,l})^2}}e^{-\frac{(x_l-\mu^j_l)^2}{2(\sigma^j_{k,l})^2}}\left(\frac{(x_k-\mu^j_k)^2+(x_l-\mu^j_l)^2}{(\sigma^j_{k,l})^3}\right) \tag{2.22}$$

$$\Delta\mu^j_i = -\alpha(y-\hat{y})w^j_{k,l}e^{-\frac{(x_k-\mu^j_k)^2}{2(\sigma^j_{k,l})^2}}e^{-\frac{(x_l-\mu^j_l)^2}{2(\sigma^j_{k,l})^2}}\left(\frac{x_i-\mu^j_i}{(\sigma^j_{k,l})^2}\right) \tag{2.23}$$

where k specifies the input of 1-D nodes for which $E[(\Delta w^j_i)^2]$ is largest. From the above equations, it is clear that the parameters of these 2-D Gaussian nodes are being trained to correct the insufficiency of previous 1-D ones based on the context specified by additional dimensions of the inputs.

Upon the completion of every learning period, the accumulated output error (ϵ) of the network is recorded and checked. If it does not indicate a significant improvement over the previous one, new higher-dimensional Gaussians nodes are made to grow in place of the original node with largest variance such that a better approximation of y can be achieved. In what follows, we will describe the detailed implementation of the LowD RBF network growing process step by step:

Step–1 Set the INITIAL MSE THRESHOLD - stopping criteria for the growing algorithm. Set the CONVERGENCE TOLERANCE - determine whether the algorithm may continue learning or add new higher-dimensional nodes.

Step–2 Select m 1-D Gaussian nodes allocated on each input dimensions. Initialize the parameters of these nodes to be:

- w - set to small random values and save them in temporary array.

- σ - equal σ_{max} which has the value large enough to cover the domain of the input space.
- μ - distribute all m nodes evenly over the input range of each dimensions.

Step–3 Set $i = 1$ (index for learning period) and $j = 1$ (index for growing period).

Step–4 Invoke one periodic learning procedure at the ith learning period, where one learning period implies the presentation of all s training samples to the network. The learning procedure is as follows:

- Get the next training sample.
- Perform steepest descent on all Gaussian parameters w, σ, and μ.
- Accumulate $(\Delta w)^2$ and save them in temporary array.
- Compute the network performance at the ith learning period by

$$p_i = \sum^{s} \epsilon_s(i)$$

where $\epsilon_s(i)$ represents the error function of s samples at ith learning period.

Step–5 If the performance improvement due to the parameters learning is saturated, (i.e. $p_i \leq$ INITIAL MSE THRESHOLD) then terminate the growing algorithm.

Step–6 If it is not saturated, check the percentage of improvement compared with the previous learning period by

$$m_i = |\frac{p_i - p_{i-1}}{p_i}| \times 100\%$$

If $m_i >$ CONVERGENCE TOLERANCE, set $i = i + 1$, go to Step-4 and continue another learning period.

Step–7 If $m_i \leq$ CONVERGENCE TOLERANCE, apply the Gaussian nodes growing procedures listed below:

- Find the node with maximum $E[(\Delta w)^2]$ as the parent.
- Generate m new 1-D Gaussians on each input dimensions except the ones take over by the parent node.
- Initialize all parameters in newly born Gaussian nodes as
  - w - set to small random values and save them in temporary array.
  - σ - equal $\sigma(j)$ which shrinks continuously using function

    $$\sigma(j) = max[\sigma_{max}e^{(-\frac{j}{DECAY})}, \sigma_{min}]$$

    at jth growing period.
  - μ - distribute all m nodes evenly over the input range of the chosen input dimensions.
- Synthesize the new higher-dimensional nodes as a product of all these new 1-D Gaussians and the parent node.
- Remove the original parent node from the network.

set $i = i + 1$ and $j = j + 1$, go back to Step-4 and continue the next learning period.

Typically, p_i is an accuracy measurement of output of the network at the ith learning period. p_i larger than the INITIAL MSE THRESHOLD are either immediately corrected by the allocation of new nodes or continuously repaired using steepest descent. The choice of these two actions to be taken are distinguishable by m_i. A very small value of m_i (or one less than the CONVERGENCE TOLERANCE) means that the trajectory of error transient comes into a flat region of the error surface. Since there is a great possibility that the network may become stable or enter the slow convergence state, new nodes should be added as a perturbation for the learning of the network. The width $\sigma(j)$ of these newly added nodes is the scale of resolution that the network is able to capture the detail of the approximated function at the j growing period. The learning starts with largest scale $\sigma(j) = \sigma_{max}$, which creates a coarse representation of the function. Then refines the representation by allocating nodes with smaller and smaller $\sigma(j)$. Finally, it reaches σ_{min} which is the smallest length scale of interest.

Note that the defective node selection heuristic described here only depends on the measurement of the maximum variance of Δw. In fact, there are two different kinds of parameter change $\Delta\sigma$ and $\Delta\mu$ which are also available as part of the LMS learning. However including all of them into our growing rule may not be the best idea. Since the axis of optimization for $\Delta\sigma$ and $\Delta\mu$ are dependent on Δw, using only Δw is adequate to reflect the inadequacy of each Gaussian nodes. Moreover, memory storage requirements of our growing algorithm can be lessened.

2.2.2 Pruning Heuristic

Up to now, our study merely concentrated on network growing. It is time to switch our attention to the final stage of the LowD RBF network development-the pruning process. As pointed out in section 2.1.1, in order to increase the probability for the network to approach a satisfactory solution, the growing of new Gaussian nodes would include predetermined combination of all input dimensions so that the maximum number of nodes $m(n-1)$ is added at each growing period. This is undesirable at first glance, because a considerable number of nodes which contain irrelevant input information would be unavoidably generated as a by-product of such a process. However, the technique for factoring out irrelevant components of the inputs can be computationally expensive and requires a large body of *a priori* information about the underlying structure of the problem. Yet many other simpler methods have been proposed to accomplish this reduction. One of them is to let the network grow larger than necessary and the unneeded nodes are then removed afterwards. This is exactly the approach that is pursed here.

Unfortunately, there is no general way to determine which nodes can be removed while the network performance would not be significantly impaired due to the removals. One possibility which is used in [14] is to eliminate the nodes according to their "contribution" to the LMS learning. It is suggested that the "contribution" of each node is explicitly measured by calculating the slope (or sensitivity) of the output error function $\epsilon = \frac{1}{2}(y-\hat{y})^2$ with respect to the training parameter. Here, our approach for pruning is to estimate the sensitivity of ϵ with respect to the Gaussian parameter at the end of each learning period. Then the node with the lowest sensitivity value is pruned. More redundant nodes would be removed in the subsequent periods so that the speed of convergence of the whole network is improved.

For the sake of simplicity, the estimation of sensitivity (s) is only in terms of the Gaussian weight (w). The significance of this choice will be apparent in the later part of our discussion. Starting the formulation with 1-D nodes, assuming that ϵ is a function of w_i^j only, s_i^j with respect to w_i^j can be defined as

$$s_i^j = -\frac{\epsilon(E) - \epsilon(S)}{w_i^j(E) - w_i^j(S)} w_i^j(E) \tag{2.24}$$

where E and S denote the specific values at the beginning and end of the LMS learning respectively. Note that during the development of completed network structure, both the initialization of new Gaussian parameters and the evaluation of sensitivity are performed periodically. This makes the duration of training process vary from node to node. To account for this situation, we will approximate the sensitivity by the average slope of ϵ

$$s_i^j \approx -\frac{\epsilon(E) - \epsilon(S)}{E[w_i^j] - S[w_i^j]} \frac{w_i^j(E)}{w_i^j(E) - w_i^j(S)} \tag{2.25}$$

where $E[w_i^j] - S[w_i^j]$ denotes the difference in learning periods according to individual w_i^j.

In a normal LMS search procedure, the actual error difference $\epsilon(E) - \epsilon(S)$ is a function of all learning parameters. Apparently, this is in contrast to our assumption in equations 2.24 and 2.25. To elaborate, consider the variation of ϵ in the domain of $\boldsymbol{w}$ while assuming all other parameters $\boldsymbol{\sigma}$ and $\boldsymbol{\mu}$ remains constant. Substituting equation 2.2 into 2.8 and expanding yields

$$\epsilon = \frac{1}{2}(y^2 - 2yw_i^j e^{-\frac{(x_i-\mu_i^j)^2}{2(\sigma_i^j)^2}} + (w_i^j)^2 e^{-\frac{(x_i-\mu_i^j)^2}{(\sigma_i^j)^2}}) \tag{2.26}$$

where ϵ is a quadratic function of w_i^j so that a convex error surface is formed. Figure 2.3 shows a example of a typical surface contour for a network with w_1 and w_2 only. The position of a point and the contour lines represents the values of two weights and the error respectively. The actual trajectory of the error difference descending from point S to E is indicated by the line with arrow. From this figure, it is obvious that we can simplify the calculation of s_1 by using only the partial influence of the error due to the changes in w_1 which corresponds to the path from $S[w_1]$ to $E[w_1]$. Hence our assumption is clarified.

Regarding equation 2.25, the calculation of s_i^j can now be evaluated precisely by expressing the error difference as

$$\epsilon(E) - \epsilon(S) = \int_{E[w_i^j]}^{S[w_i^j]} \frac{\partial \epsilon}{\partial w_i^j} dw \tag{2.27}$$

The integral is along the error path projected onto the w_i^j axis from $S[w_i^j]$ to

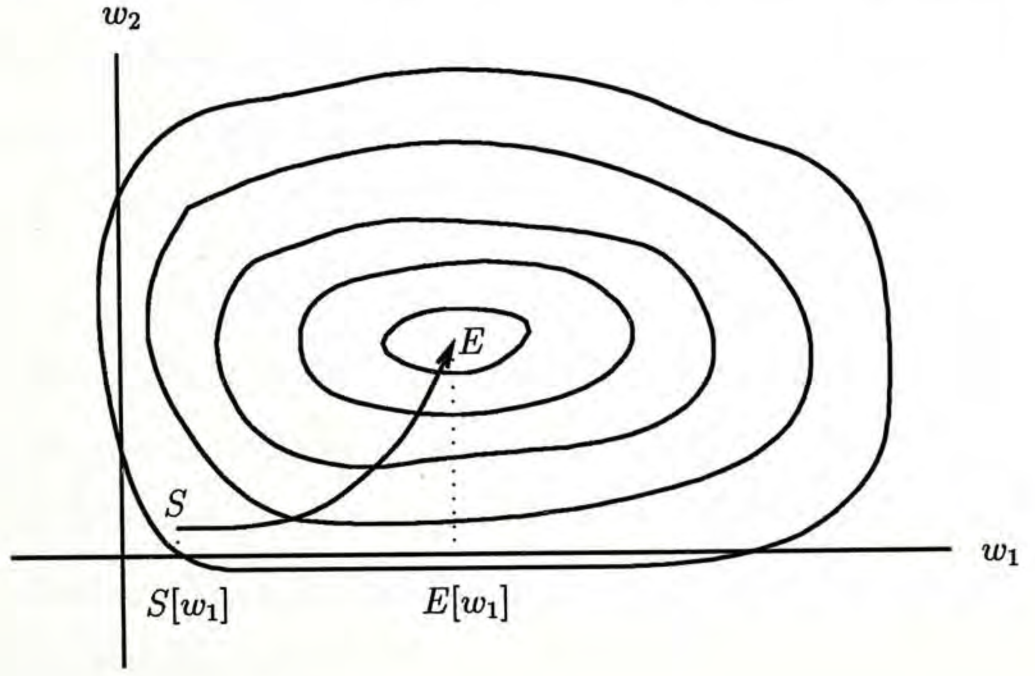

Figure 2.3: Learning trajectory of the error path

$E[w_i^j]$. Since the LMS learning calculation is performed at discrete times. The above expression can be further simplified by replacing the integration by a summation.

$$\epsilon(E) - \epsilon(S) = \sum_{n=S[w_i^j]}^{E[w_i^j]} \frac{\partial \epsilon}{\partial w_i^j}(n)\, \Delta w(n) \tag{2.28}$$

For the 1-D nodes, w_i^j are updated according to 2.15, hence the estimated sensitivity to the removal of w_i^j can be rewritten as

$$s_i^j = \frac{\sum_{n=S[w_i^j]}^{E[w_i^j]} (\Delta w_i^j(n))^2}{\alpha(E[w_i^j] - S[w_i^j])} \frac{w_i^j(E)}{w_i^j(E) - w_i^j(S)} \tag{2.29}$$

or can be expressed in terms of expected value

$$s_i^j = \frac{E[(\Delta w_i^j)^2]}{\alpha} \frac{w_i^j(E)}{w_i^j(E) - w_i^j(S)} \tag{2.30}$$

Periodically, the output error ϵ is checked to see if it shows signs of slow improvement over previous ones. A redundant node with the smallest sensitivity value with be eliminated in the hope that it can speed up the convergence of the network. This removal procedure are continued until the error is reduced to below some desired level. Details of the pruning algorithm are given below:

Step–1 Set the FINAL MSE THRESHOLD to a smaller value (e.g. 50% of the INITIAL MSE THRESHOLD in the growing process) - stopping criteria for the pruning algorithm. Set the CONVERGENCE TOLERANCE equal to the previous value in the growing procedure - determine whether the algorithm may continue learning or eliminate the redundant nodes.

Step–2 Continue the one periodic learning procedure at the *i*th learning period.

Step–3 Check if $p_i \leq$ FINAL MSE THRESHOLD. If yes, exit the pruning algorithm with success.

Step–4 If no, check the marginal improvement compared with the previous learning period. If $m_i \leq$ CONVERGENCE TOLERANCE, Retrieve all necessary data from the temporary array and compute the sensitivity of each nodes according to

$$s = \frac{E[(\Delta w)^2]}{\alpha} \frac{w(E)}{w(E) - w(S)}$$

then remove the one with minimum value.

Step–5 Set $i = i+1$, go back to Step-4 and continue another learning period.

2.2.3 Summary

We have presented the idea of automatically constructing appropriate structure of LowD RBF networks that successfully approximates a given function which might have a few irrelevant input dimensions. The construction algorithm mainly composes of two processes - network growing and pruning. The growing process builds up the network elements - (Gaussian nodes) incrementally until the learning is successful. However, in order to avoid using *a priori* knowledge of the data, all input dimensions information are included in the node adding rule. Thus, large amount of unnecessary elements may involve in the resultant network structure. The pruning procedure is then used to trim the oversized network by removing unneeded nodes. This exactly complements the previous process. In fact, by simply cascading the pruning procedure to the growing process, a complete algorithm is formed. Figure 2.4 shows the flowchart

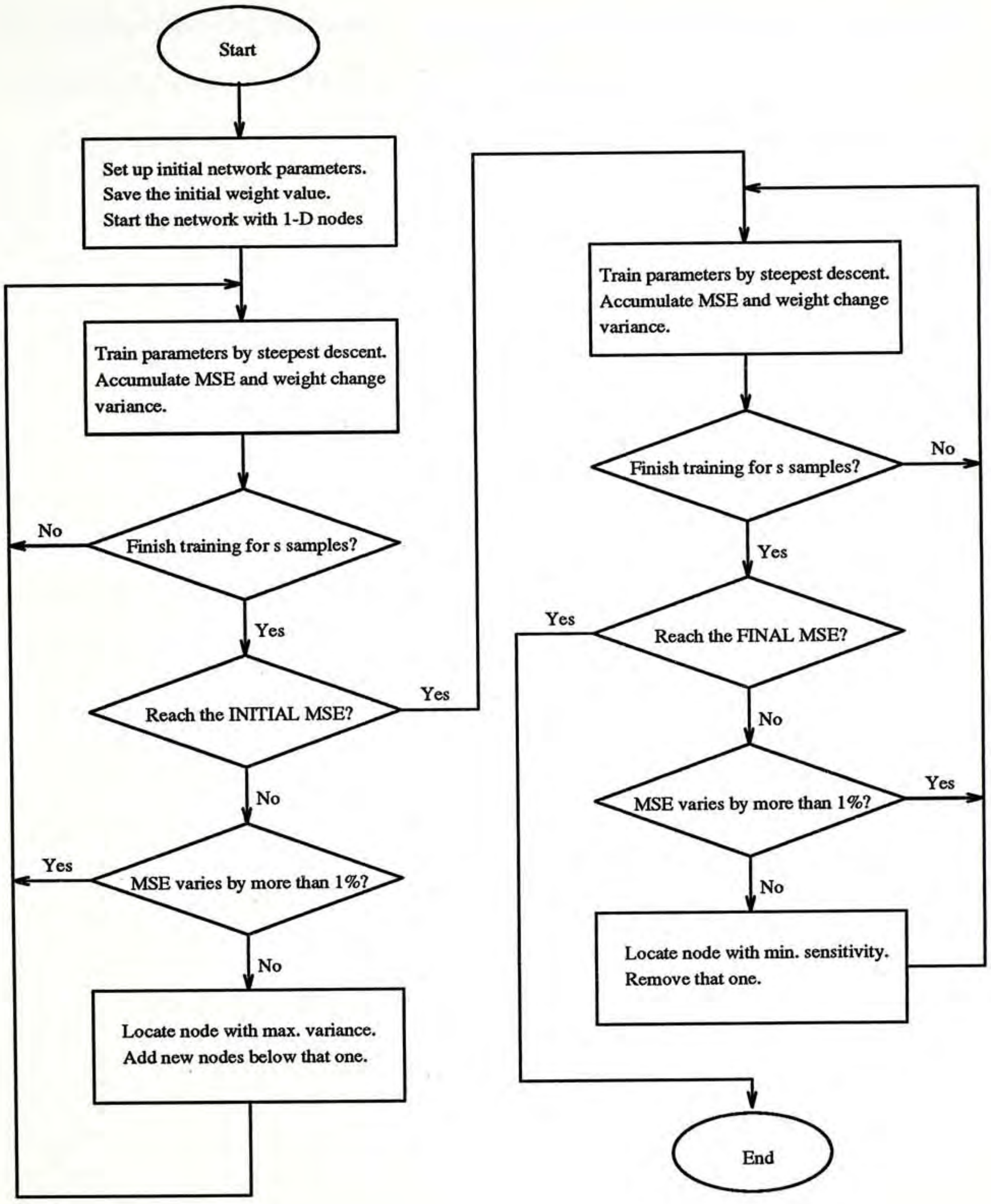
Start
Set up initial network parameters.
Save the initial weight value.
Start the network with 1-D nodes
Train parameters by steepest descent.
Accumulate MSE and weight change variance.
Finish training for s samples?
No
Yes
Reach the INITIAL MSE?
Yes
No
MSE varies by more than 1%?
Yes
No
Locate node with max. variance.
Add new nodes below that one.
Train parameters by steepest descent.
Accumulate MSE and weight change variance.
Finish training for s samples?
No
Yes
Reach the FINAL MSE?
Yes
No
MSE varies by more than 1%?
Yes
No
Locate node with min. sensitivity.
Remove that one.
End

Figure 2.4: Flowchart of the construction algorithm

of this combined algorithm. In the next chapter we shall verify the usefulness of this algorithm by examining whether the algorithm helps save computation and memory space while achieving high accuracy.

Chapter 3

Application examples

In this chapter we present the results of some numerical experiments used to test the performance of our low-dimensional Gaussian RBF networks. We also perform extensive analysis on the associated network construction algorithm. Although the simulations being studied are indeed quite simple, they still effectively demonstrate the operating principles of the new algorithm. The first set of experiments uses synthetic data. It involves prediction of the Mackey-Glass differential delay equation. Several conventional ANN models will be used for performance comparison. The second experiments employs the LowD RBFs as a nonlinear predictor of speech data. The purpose is to check whether the LowD RBFs is able to exploit nonlinear as well as linear correlations in real data.

3.1 Chaotic Time Series Prediction

As a simple test case, LowD RBFs is used to predict the chaotic time series generated by integrating the Mackey-Glass delay differential equation [20]

$$\frac{dx(t)}{dt} = 0.2\frac{x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t) \tag{3.1}$$

where τ is a delay parameter which specifies the width of an initial function. This time series has an infinite-dimensional phase space. However, it does have low-dimensional attractors whose dimension increases with τ [8]. This series has become something of a standard benchmark for prediction algorithms [5], [9], [18], and [22].

First, we consider the case where $\tau = 17$, for which the series has an attractor with a fractal dimension of about 2.1 [8]. To generate the values of x at discrete time steps, the above equation is integrated using a fourth-order Runge-Kutta method with the initial function set to a constant value of 0.8. Following the previous references, the networks is trained to predict $x(t + 85)$ given $x(t)$, $x(t-6)$, $x(t-12)$, and $x(t-18)$ as inputs. (We shall refer to this as an "85-step prediction".)

To set up the LowD RBFs construction algorithm, we have to decide how many single dimension Gaussian nodes to be allocated for each input components and their initial width σ_{max}. We have no specific criterion for deciding what the optimal values should be. In the following experiments the number of nodes is chosen to be 4 and the width to be half of the maximum input range divided by that node number $\sigma_{max} = \frac{1.4}{8}$. This choice is arbitrary and definitely not optimal with regard to prediction accuracy. Other parameters used by the algorithm to predict Mackey-Glass equation with $\tau = 17$ are listed as follows:

- $\alpha = 0.025$
- DECAY= 30
- CONVERGENCE TOLERANCE= 0.01
- INITIAL MSE THRESHOLD= 0.0002
- FINAL MSE THRESHOLD= 0.000085

Since the above DECAY and the two MSE THRESHOLD parameters have great influence on the characteristics and accuracies of the final LowD RBF networks, special attention are paid to the sensitivity analysis of these parameters in the subsequent sections.

During the dynamic construction process, the sample inputs to the network are randomly taken from 500 training set at each learning period. Gaussian nodes are either added or removed from the network whenever the CONVERGENCE TOLERANCE is found to be less than 0.01. The final network structure is grown to have the following number of nodes with maximum dimension of 3.

- Number of 1-D nodes = 4
- Number of 2-D nodes = 124
- Number of 3-D nodes = 115

Figure 3.1 shows the variation of the average MSE as the number of nodes is changed. The complete process requires 425 learning periods which takes approximately 50 minutes of CPU time on a DEC 5000 workstation. To indicate how accurate the result could become, the normalized root mean-squared (NRMS) error is employed here as the figure of merit.

$$NRMS\ Error = \sqrt{\frac{\sum^{s}(y-\hat{y})^2}{\sum^{s}(y-\bar{y})^2}} \tag{3.2}$$

where $\bar{y}$ is the mean target value over s testing samples. In equation 3.2, the NRMS is defined to be the root mean-squared error, divided by the standard deviation of the data sample. It is necessary to remove the scale dependence of the sample and the standard deviation provides such a scale to use. Thus, the NRMS is insensitive to the dynamic range of the time series.

Now to measure the performance of the network for the 85-step prediction, the previous 500 training samples is used as a test set and input to the network again, the NRMS error is found to be 0.051. Figure 3.2 shows the predictions and the true time series are virtually indistinguishable. Figure 3.3 shows the NRMS error as a function of 85-step predictions.

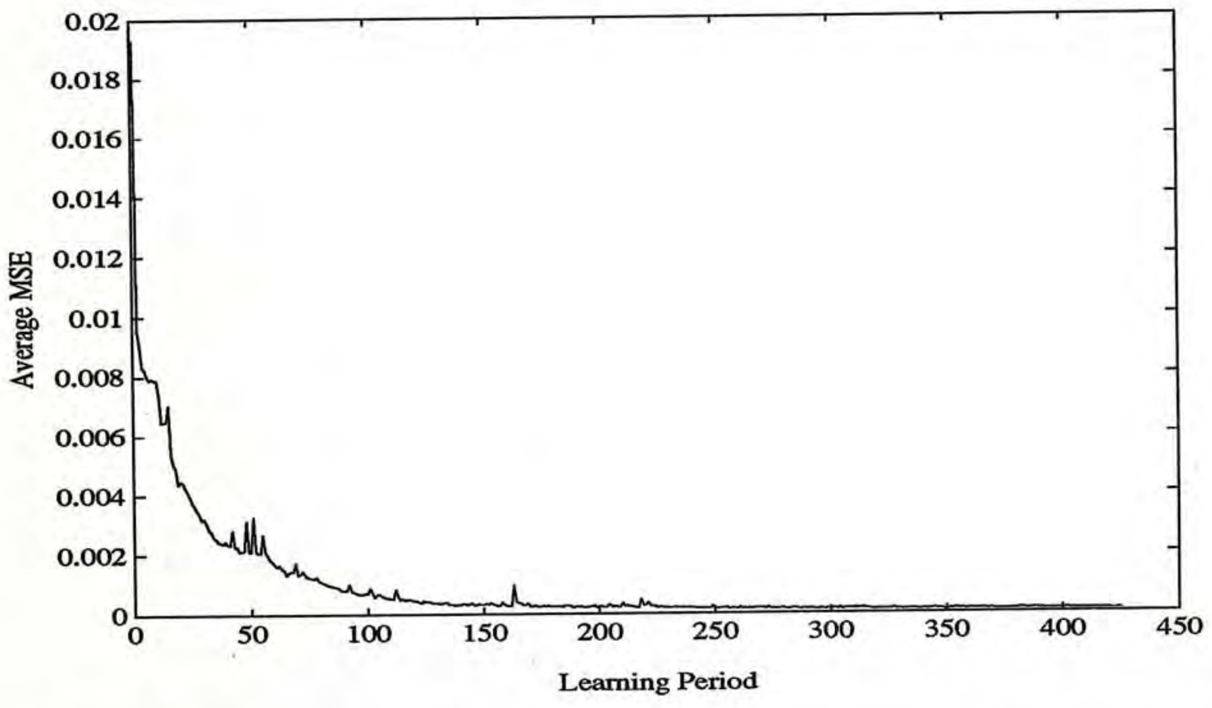

Figure 3.1: Error transitions of the network during construction process.

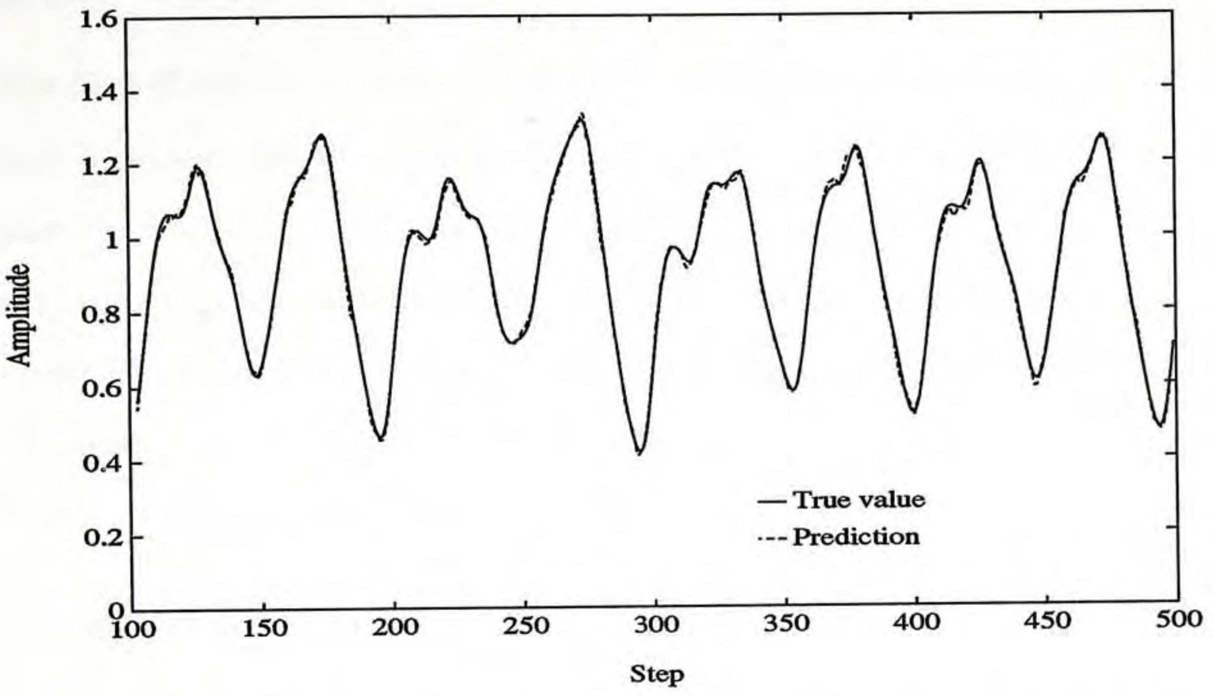

Figure 3.2: The 85-step prediction output of LowD RBFs and the true values.

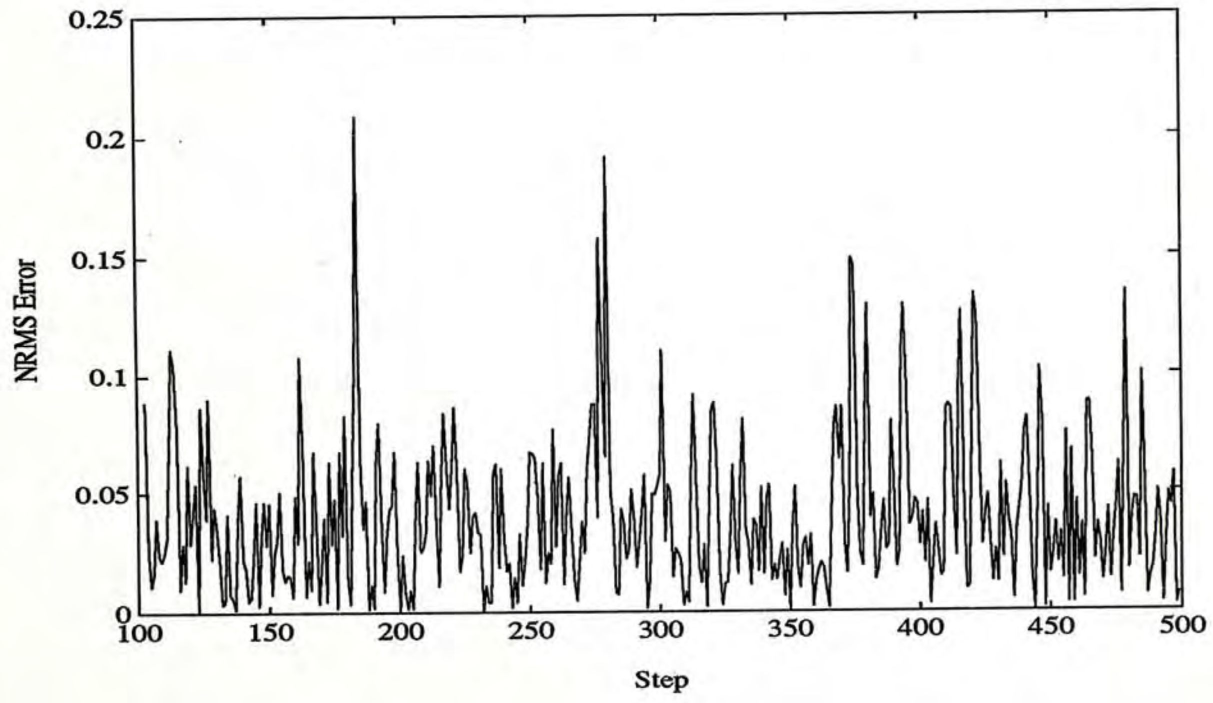

Figure 3.3: The NRMS error on each time step for the 85-step predictions.

3.1.1 Performance Comparison

The idea of training a network based on low dimensional feature is not new. In fact, a network algorithm described as "Gaussian bars" by Hartman et al. [11] have similarities to the LowD RBFs proposed here. In the network with only a single layer of 1-D nodes, the structure is exactly equivalent to a single layer of Gaussian bar networks. However, the two networks differ in the way that the Gaussian nodes are combined for the multilayer architecture: the LowD RBFs are composed layers of 1-D nodes via multiplication in order to produce higher dimensional nodes, while in a multilayer Gaussian bars, the networks are more similar to the structure of BP in which the outputs of one Gaussian bars unit can regard as direct inputs to the other units. We now compare the LowD RBFs with the Gaussian bars as well as other network models. Note that the output error and parameters saving of the LowD RBFs are actually comparable to or even exceeded the results in [11], which summarizes again in table 3.1.

Network type	Total parameters	Normalized RMS error
LowD RBFs	768	0.06
	1083	0.05
Gaussian Bars	750	0.22
	1461	0.19
	4500	0.06
BP	171	0.54
	601	0.53
RBFs	601	0.30
	1801	0.03
	3001	0.02

Table 3.1: Performance summary for 85-step prediction problem.

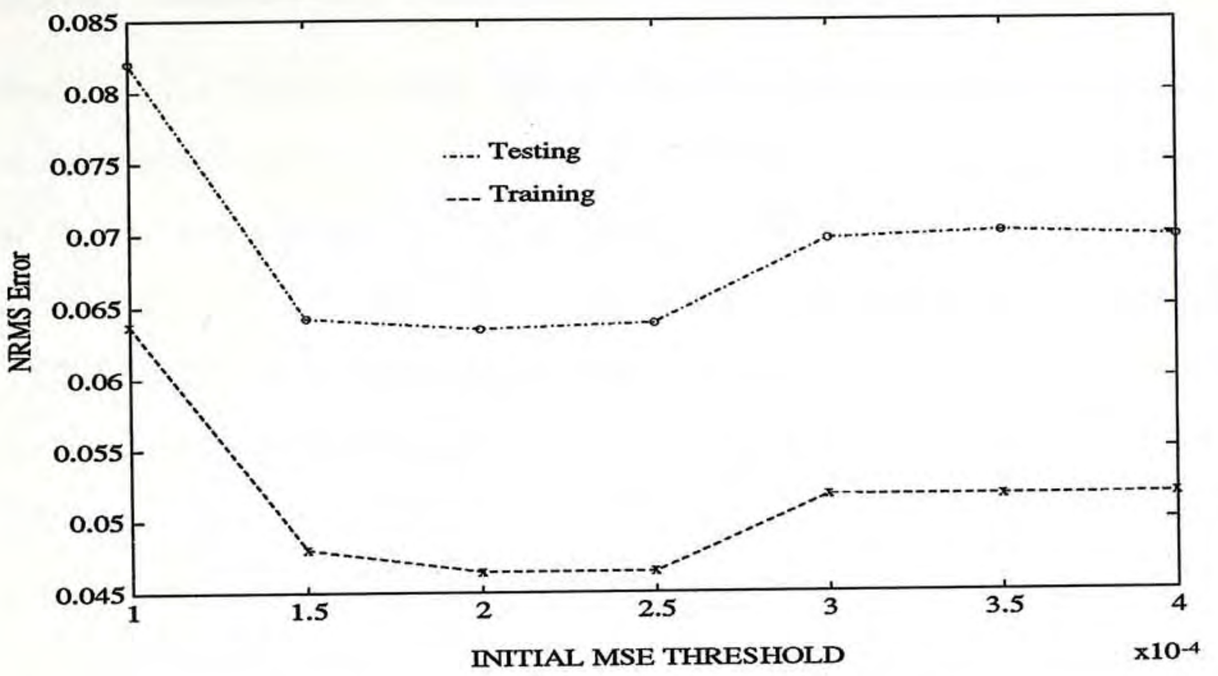

Figure 3.4: The total network parameters as a function of the INITIAL MSE THRESHOLD

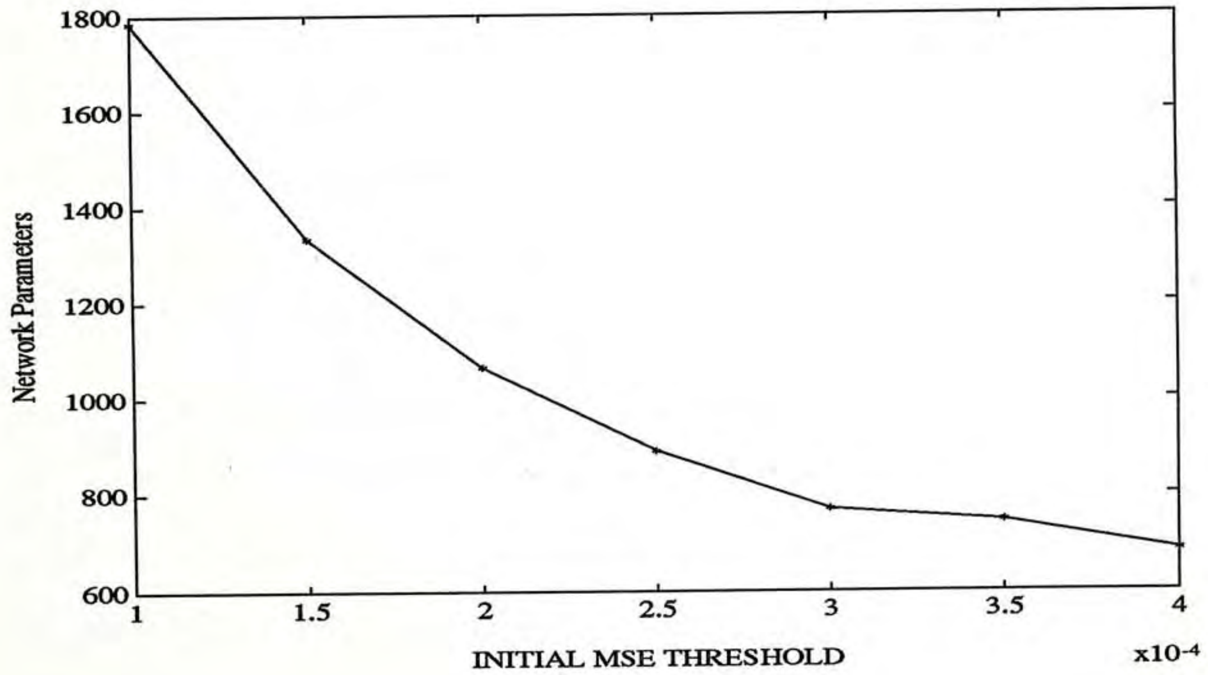

Figure 3.5: The prediction error as a function of the INITIAL MSE THRESHOLD

3.1.2 Sensitivity Analysis of MSE THRESHOLDs

Basically, the INITIAL MSE THRESHOLD controls the amount of Gaussian nodes growth in the dynamic construction process. To study the effectiveness of the the above parameter for the 85-step prediction task, we vary the INITIAL MSE THRESHOLD from 0.0001 to 0.0004, while fix the FINAL MSE THRESHOLD to a small constant value of 0.00007 and let the networks grow for maximum 450 learning periods. In addition to the original training set, a 500 testing samples of the Mackey-Glass equation at $t = 2000$ are employed here to reflect the generalization ability. Figure 3.4 shows the reduction of total parameters as the value of INITIAL MSE THRESHOLD increases. Figure 3.5 depicts the trend of NRMS error as the function of INITIAL MSE THRESHOLD. It is clear that as the MSE increases, the error both on training and test sets decrease as the network has enough degrees of freedom to fit the structure of all the LowD RBF nodes. However, after a critical value of MSE has been reached, the error starts to increase. From this point, the network has suffered from the overfitting problem which will be discussed in detail shortly afterwards

3.1.3 Effects of Increased Embedding Dimension

To study a higher-dimensional case, we now consider the Mackey-Glass equation with τ=30. At this value of τ, the series has an attractor with a fractal dimension of 3.6 [8]. We reconstructed the network inputs using values at $x(t)$, $x(t-6)$, $x(t-12)$, $x(t-18)$, $x(t-24)$, and $x(t-30)$. Because the dimensionality of the attractor is larger than that of the $\tau = 17$ case, the number of training samples is increased to 1000 in order to set up the LowD RBFs, for which the task is to predict $x(t+6)$. (Again, we shall refer to this as a "6-step prediction".) If the network can perform this sufficiently well, then it can be taken to iterate on its own outputs and make predictions from the "seed" data samples. For example, if one wants to predict $x(t+12)$, the network is first used to compute the $x(t+6)$ value based on the initial "seed" data, then feeds the $x(t+6)$ back into the input to predict $x(t+12)$ using the predicted $x(t+6)$ value instead of the actual $x(t+6)$ from the time series. This procedure corresponds to iterate the network to perform prediction at multiples of $x = 6$ ("Iterated prediction"). Since iterative method of prediction will cause small errors to accumulate, as expected, the error will increase farther as the network iterated into the long-term prediction.

To tackle the above higher-dimensional problem, we start the LowD RBF using the following setting of parameters.

- Number of nodes on each dimension=2
- $\sigma_{max} = \frac{1.4}{4}$
- $\alpha = 0.025$
- DECAY= 30
- CONVERGENCE TOLERANCE= 0.01
- INITIAL MSE THRESHOLD= 0.00005
- FINAL MSE THRESHOLD= 0.00003

The network is trained for 322 learning periods. (It requires almost the same CPU time as in 85-step prediction.) The final structure has the following number of nodes with a maximum dimension of 3

- Number of 1-D nodes = 3

- Number of 2-D nodes = 67
- Number of 3-D nodes = 135

which is able to achieve an NRMS error rate as low as 0.024 for the 6-step prediction. Figure 3.6 shows the trend of average MSE as the number of nodes is varied. Figure 3.7 shows that the 6-step prediction matches the actual time series very well. In figure 3.9, it shows that the network is capable of producing a valid iterated prediction up to 550 steps (with NRMS < 1). The NRMS errors for the 6-step and iterated prediction tasks are plotted in figure 3.8 and 3.10 respectively.

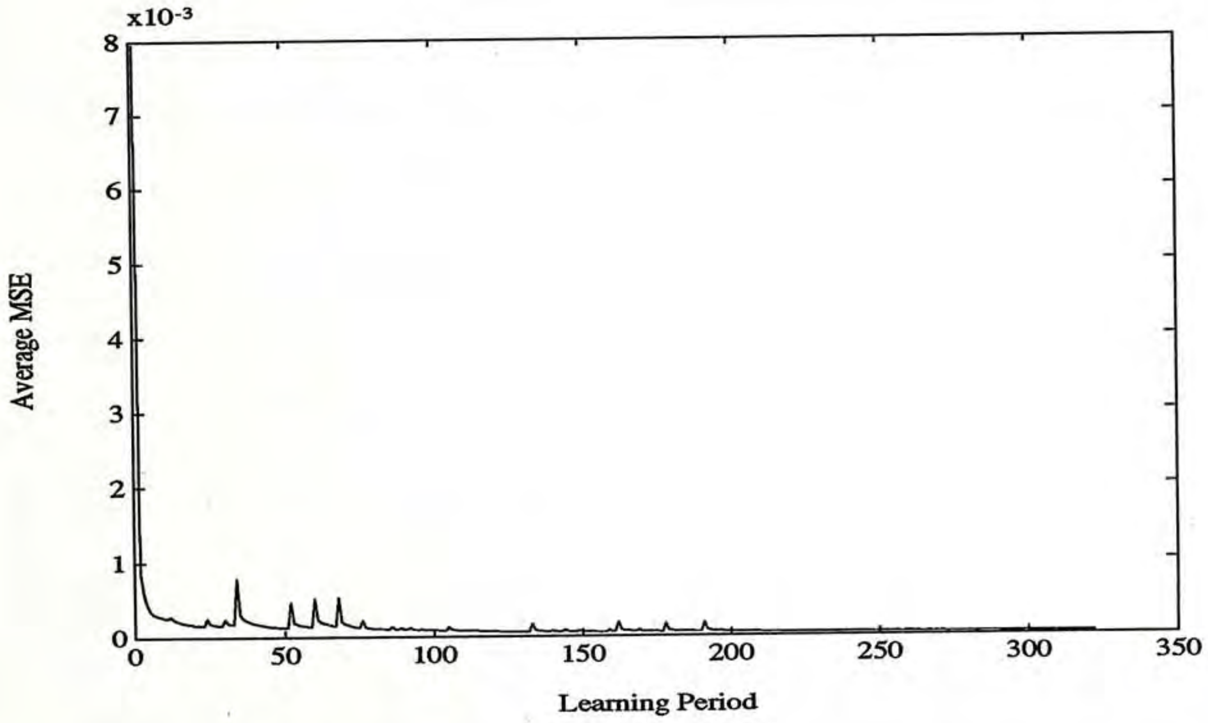

Figure 3.6: Error transitions of the network during construction process.

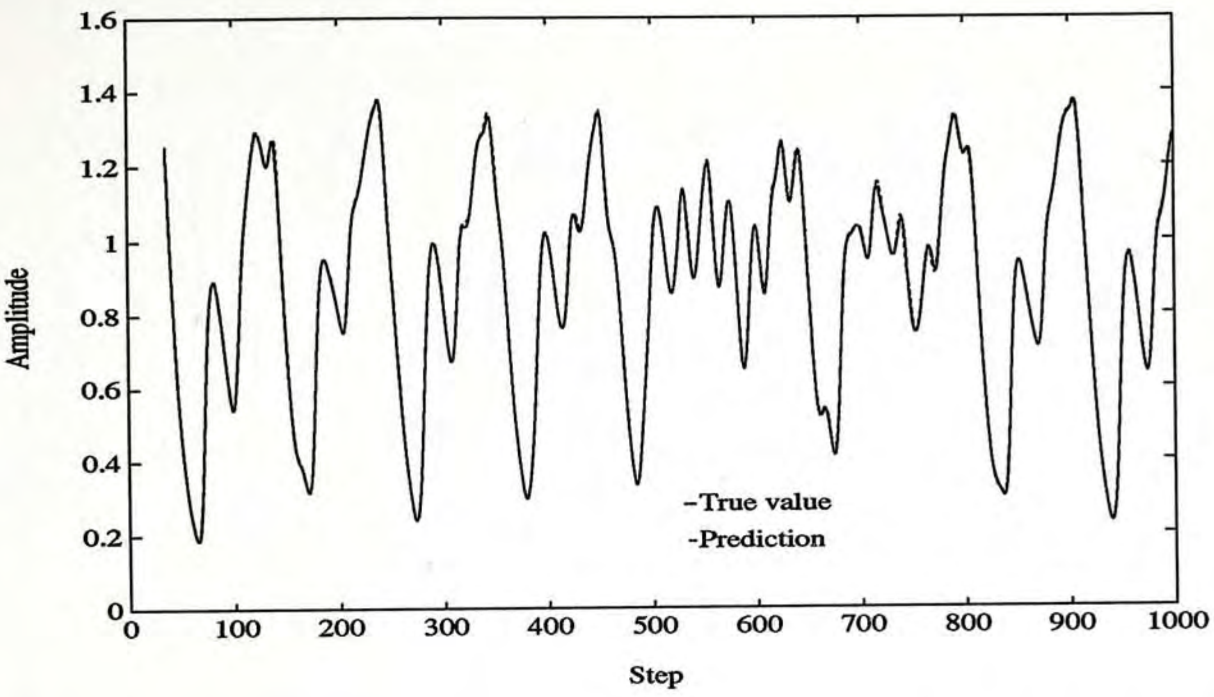

Figure 3.7: The 6-step prediction output of LowD RBFs and the true values.

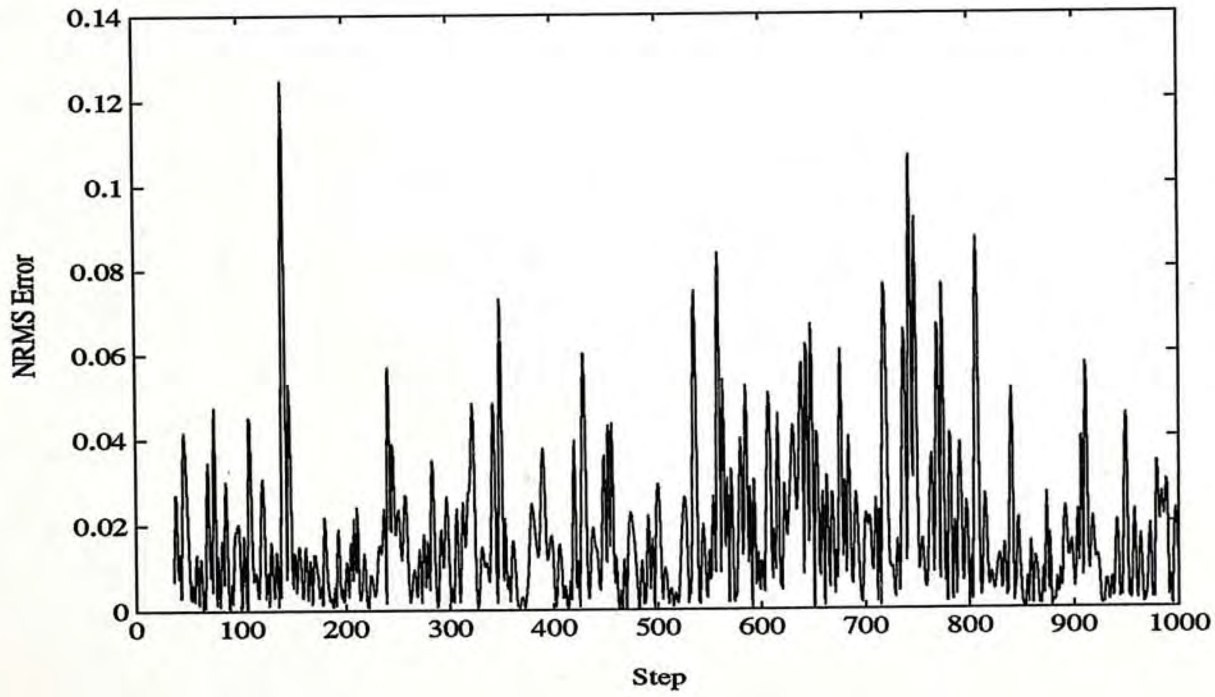

Figure 3.8: The NRMS error on each time step for the 6-step predictions

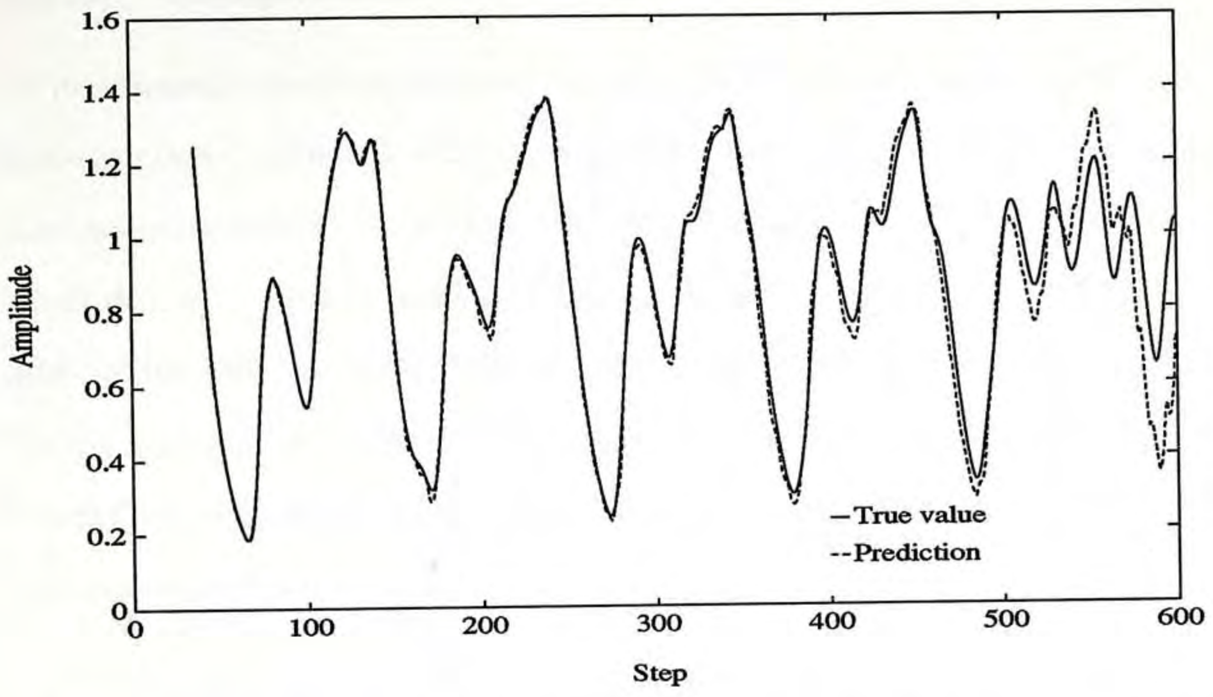

Figure 3.9: The iterated prediction output of LowD RBFs and the true values.

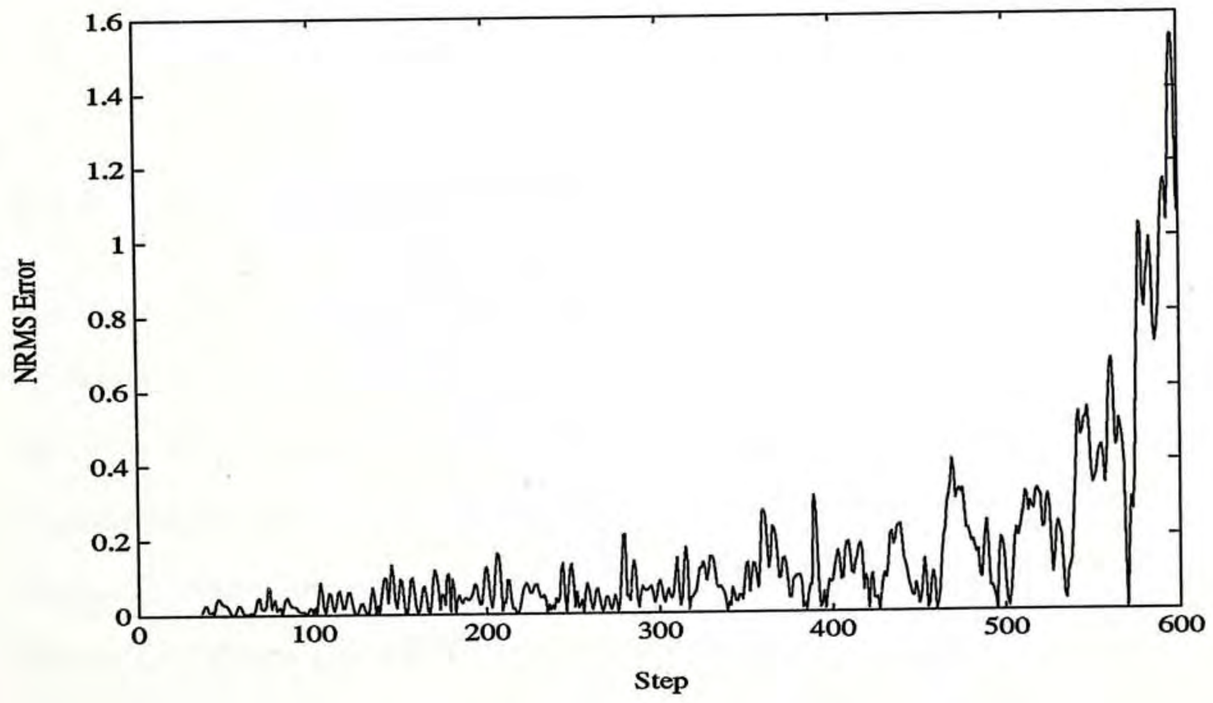

Figure 3.10: The NRMS error on each time step for the iterated predictions.

3.1.4 Comparison with Tree-Structured Network

As mentioned in previous section, the most closely related algorithm is Sanger's tree-structured networks which use Fourier basis functions to approximate a continuous function. Note that the Sanger's algorithm uses on-line learning which does not store the past input samples. This is in contrast to the training method employed in LowD RBFs networks. Therefore, it is difficult to compare the performance of these two algorithm directly. However, for reference only, some of the experiment results from 6-step prediction problem described above are summerized in table 3.2

Network type	Total parameters	NRMS error	Iteration steps
LowD RBFs	952	0.024	550
Tree-structured nets.	12720	0.025	600

Table 3.2: Performance summary for the 6-step prediction problem.

3.1.5 Overfitting Problem

It is important to note that at the beginning of the dynamic construction process, the width of the initial Gaussian nodes are defined by a large value of σ, so that they are able to create a coarse representation of the function. As learning progresses, the representation can be refined by reducing the effective radius of the newly born nodes exponentially at a rate controlled by the DECAY constant. Figure 3.12 shows the NRMS error versus different values of the DECAY. (To test the generalization performance of the network, additional 1000 test samples are taken from the output of the Mackey-Glass equation at t=4000). In this fig-

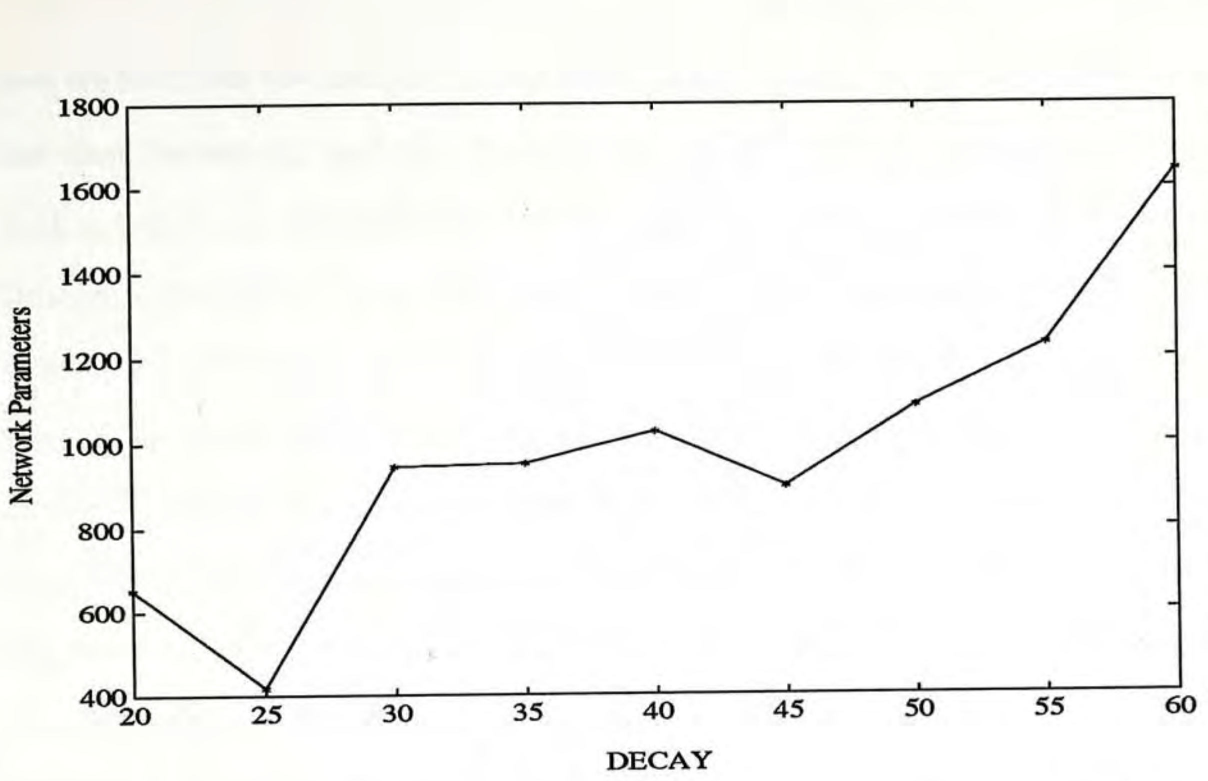

Figure 3.11: The total network parameters as a function of the DECAY parameter

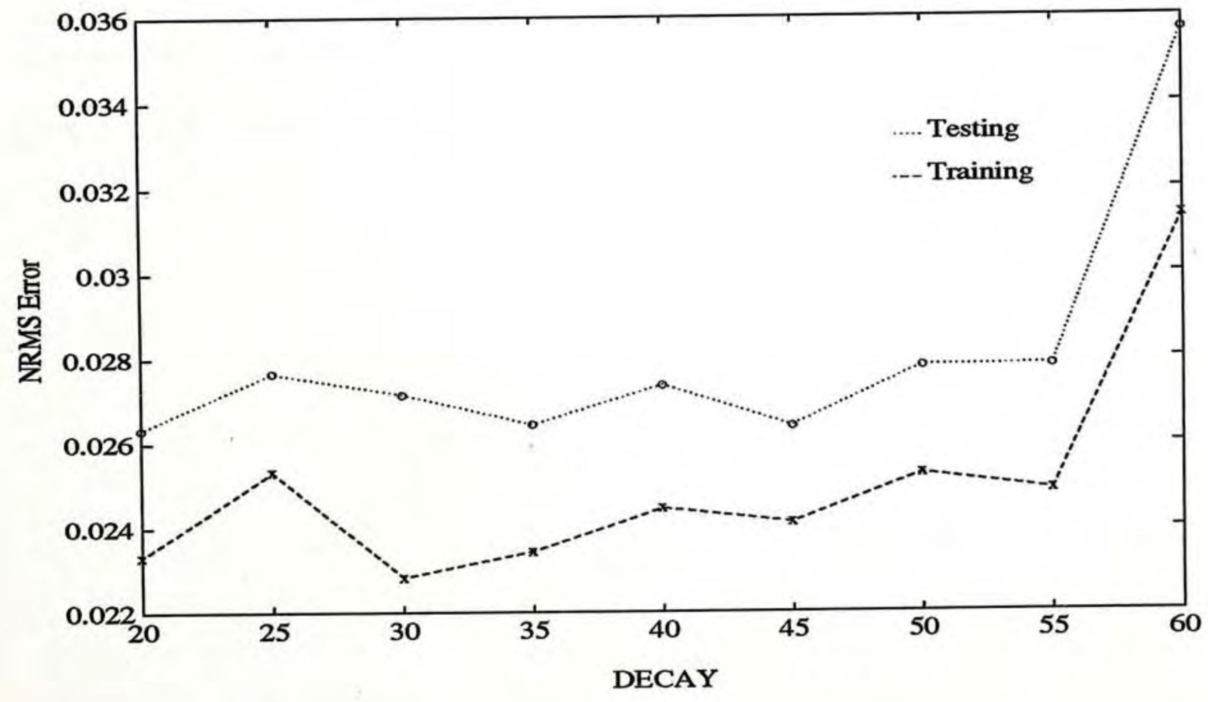

Figure 3.12: The prediction error as a function of the DECAY parameter

ure, we find that the results on both training and testing set are basically followed the slow increasing tendency in error rate as the DECAY increases. However, such a trend can not be applied to the large value of DECAY. It is because at this value of DECAY (e.g. DECAY= 60) the effect of radius decrement releases, new nodes with large width are simply added to the network. The result is that these newly added nodes are significantly overlapped which pushes many of the original nodes away from the input region, so as to reduced the overall error in the fastest possible way, rather than adjust them locally. In this case, the network is depleted of effective nodes to reduce the error in subsequent learning period. Consequently, large number of nodes are added to the network to compensate for this "loss", which in turn generate an enormous size of the network structure with little improvement on the error rate. This is further supported by the results as plotted in figure 3.11. Finally, we have to point out that the overfitting problem also occurs when too many Gaussian nodes (regardless of the width) are pumped into the input region of interest. This gives a good explanation of the situation in figure 3.5

3.2 Nonlinear prediction of speech signal

Speech coding has a long history. Approaches to coding can be divided into two groups. One is called waveform coding, which deals with the acoustic waveform itself. PCM, DPCM, delta modulation [13], and transform coding fall in this group. Another good representation of speech is to model the process by which a human speaker produces speech. For example, the vocal tract can be modeled as an all-pole filter with the poles corresponding to the vocal tract's resonance frequencies. This is the well-known linear predictive coding (LPC) technique [21]. The set of LPC coefficients plus the excitation (periodic signal or white noise) forms a more compact representation of the speech signals than the acoustic waveform. Speech coding based on this type of models is called parameter coding.

A major limitation of LPC is that it only models the poles of the vocal tract. For instance, nasal sounds which require at least a pole-zero model could not be modeled adequately. Since a zero has to be approximated by several poles, this often results in requiring a higher order linear model than ought to be necessary. Therefore, several researchers [6] [19] [34] [7] have investigated the possibility of using nonlinear, especially ANN based, predictor to further improve speech modeling.

As similar to the previous work on modeling of chaotic processes, speech production can be viewed as a flow on a low-dimensional manifold. Results in [2] and [34] reveal that most of the speech lies on an attractor with dimension approximately ranging from 3.3 to 3.4. Hence, it is possible to produce good forecasts of speech using only a subset of its past values. This situation makes the LowD RBF networks a good candidate as an nonlinear predictor (NLP) for the speech signal.

In this experiment, the LowD RBFs as well as the LPC model are examined in the task of speech prediction. Here, the LPC algorithm is implemented using the relation

$$\hat{x}(t) = \sum_{i=1}^{n} a_i x(t-i) \tag{3.3}$$

where n defines the order of prediction model or usually refers it as predictor order, $x(t)$ is the $(n+1)$th sample of the signal, a_i is a set of LPC coefficients which are computed every 10ms on a frame length of 25.6ms. Note that an error signal $e(t)$ (residual) would be generated by the equation 3.3 is defined as

$$e(t) = x(t) - \hat{x}(t) \tag{3.4}$$

Since the LPC parameters only extract the poles of the vocal tract, if the actual speech signal was well modeled by the linear predictor, then $e(t)$ is a good approximation to the external excitation.

The speech data (corresponding to the isolated word /one/) are extracted from a male speaker sampled at 10KHz with 12-bit resolution in a duration of 0.55s. An 1000 samples of the speech from segment 0.2s to 0.3s (as shown in figure 3.13) are employed as the training set to illustrate the performance of the LowD RBFs in the present of noise and periodic signal. Given n (predictor order) normalized samples as inputs, the task is to predict the value of $(n+1)$th sample. For the case of $n = 12$, the network is trained using the following parameters:

- Number of nodes on each dimension=2
- $\sigma_{max} = \frac{1.0}{4}$

- $\alpha = 0.025$
- DECAY= 20
- CONVERGENCE TOLERANCE= 0.01
- INITIAL MSE THRESHOLD= 0.00025
- FINAL MSE THRESHOLD= 0.000115

The complete training requires 370 learning periods, which produces a network of 259 nodes with a maximum dimension of 3.

- Number of 1-D nodes = 14
- Number of 2-D nodes = 189
- Number of 3-D nodes = 56

Figure 3.15 shows the prediction output of the LowD RBF networks. Before examining the nonlinear prediction results, we have to define a new figure of merit commonly used in the speech community [26] - prediction gain

$$\text{Prediction Gain} = -20log_{10}(\text{NRMS})\ \text{dB} \tag{3.5}$$

In the above test, the LowD RBFs can achieve a gain of 16.5dB, while only about 11.95dB is obtained using LPC. The results is further illustrated by the NLP and LPC residuals shown in figure 3.16 and 3.14 respectively.

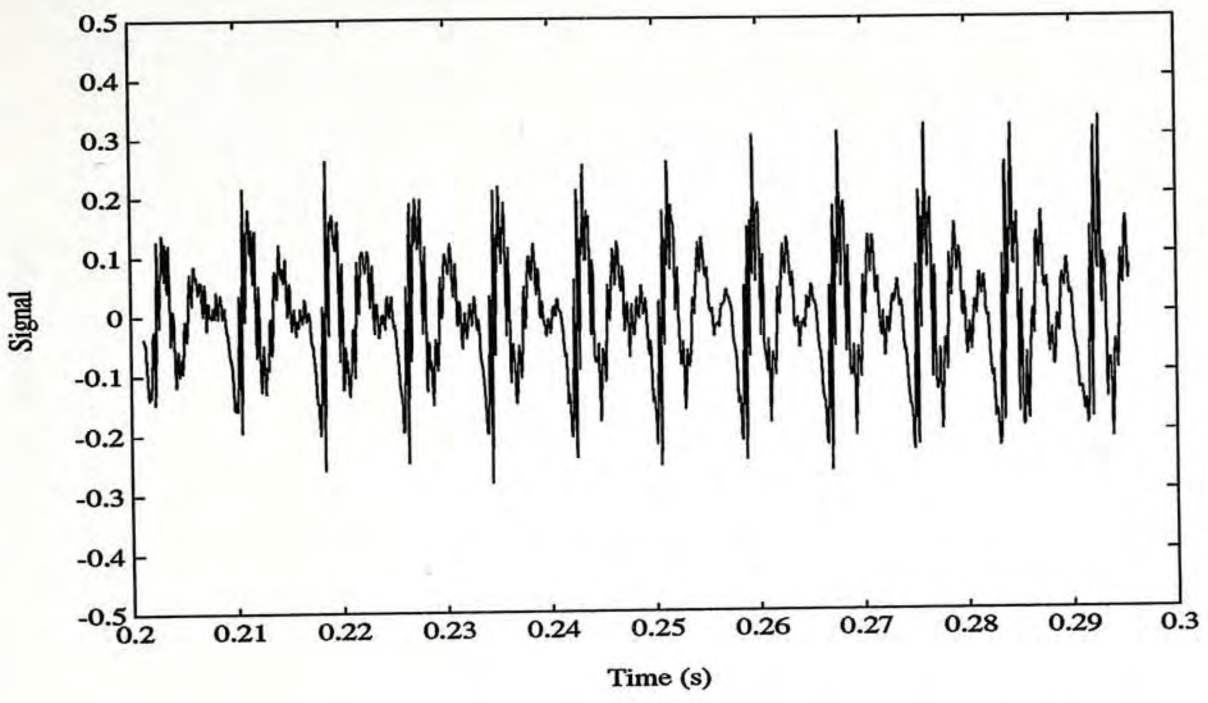

Figure 3.13: The original speech signal in the segment from 0.2s to 0.3s

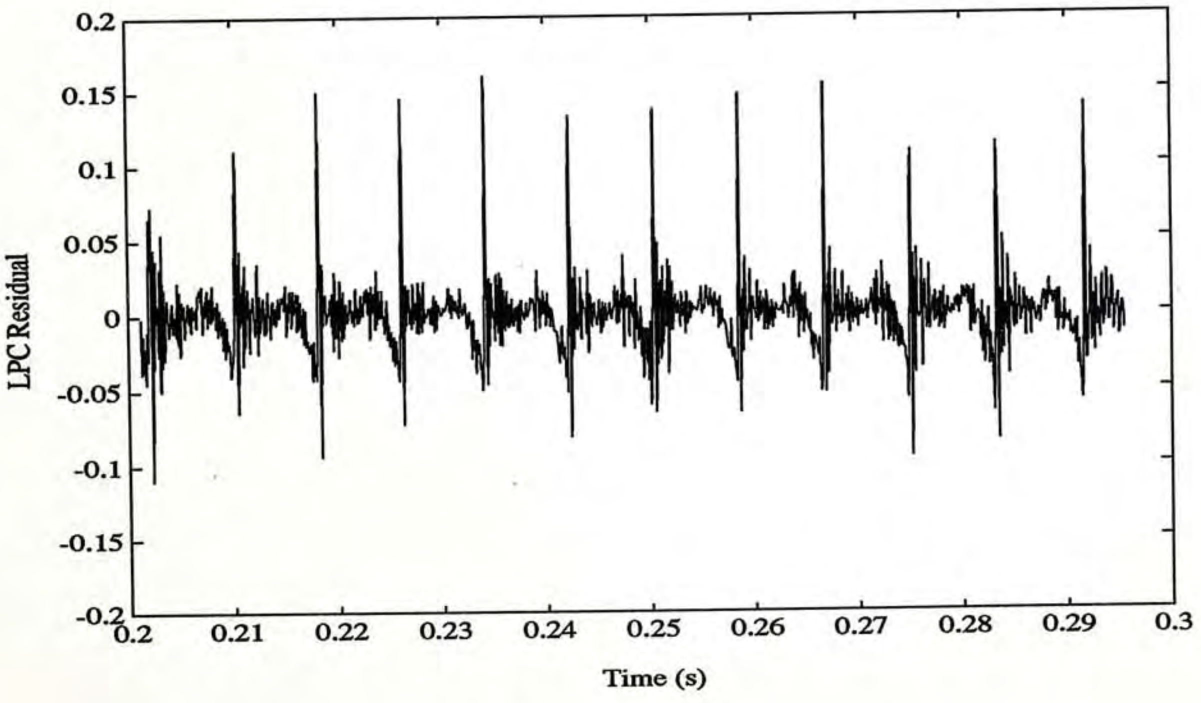

Figure 3.14: The residual after the linear prediction.

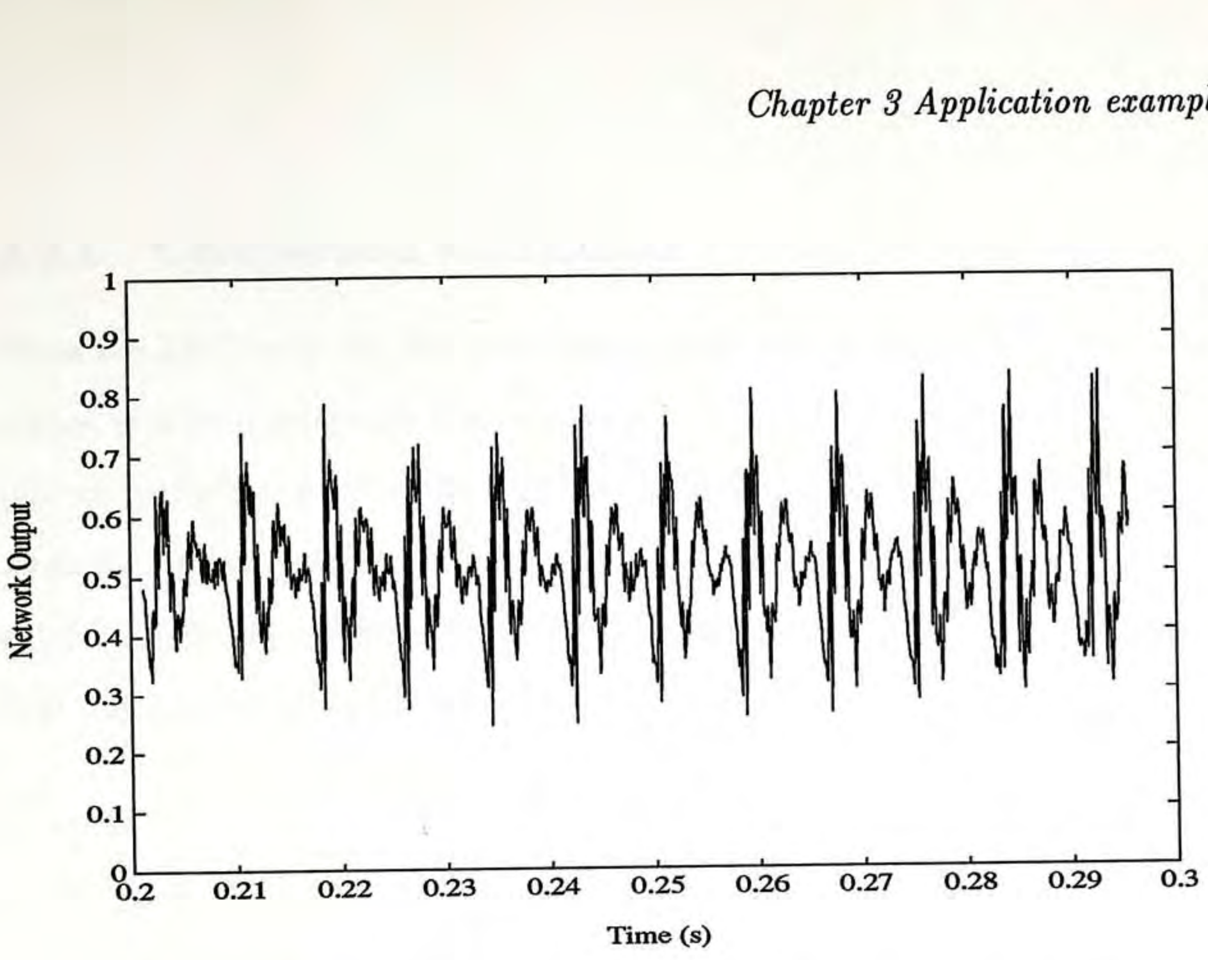

Figure 3.15: The prediction output of the LowD RBFs network.

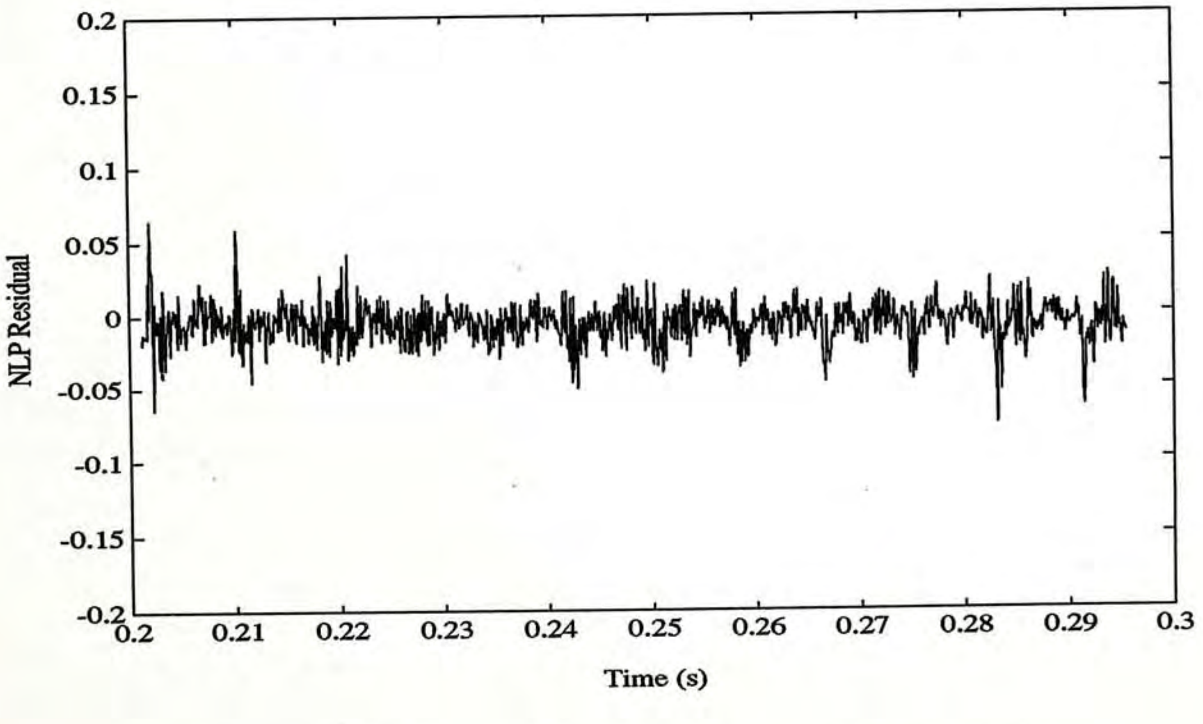

Figure 3.16: The residual after the nonlinear prediction

3.2.1 Comparison with Linear Predictive Coding (LPC)

Since the LPC is by far the most widely used speech prediction technique, it serves as a benchmark for the comparison of various NLP models [6] [19]. A plot of prediction gain versus predictor order for LowD RBFs as well as LPC models is shown in figure 3.17. From this figure, it is clear that LowD RBFs is superior in performance in terms of prediction gain by about 4dB compared to that obtained of using LPC model.

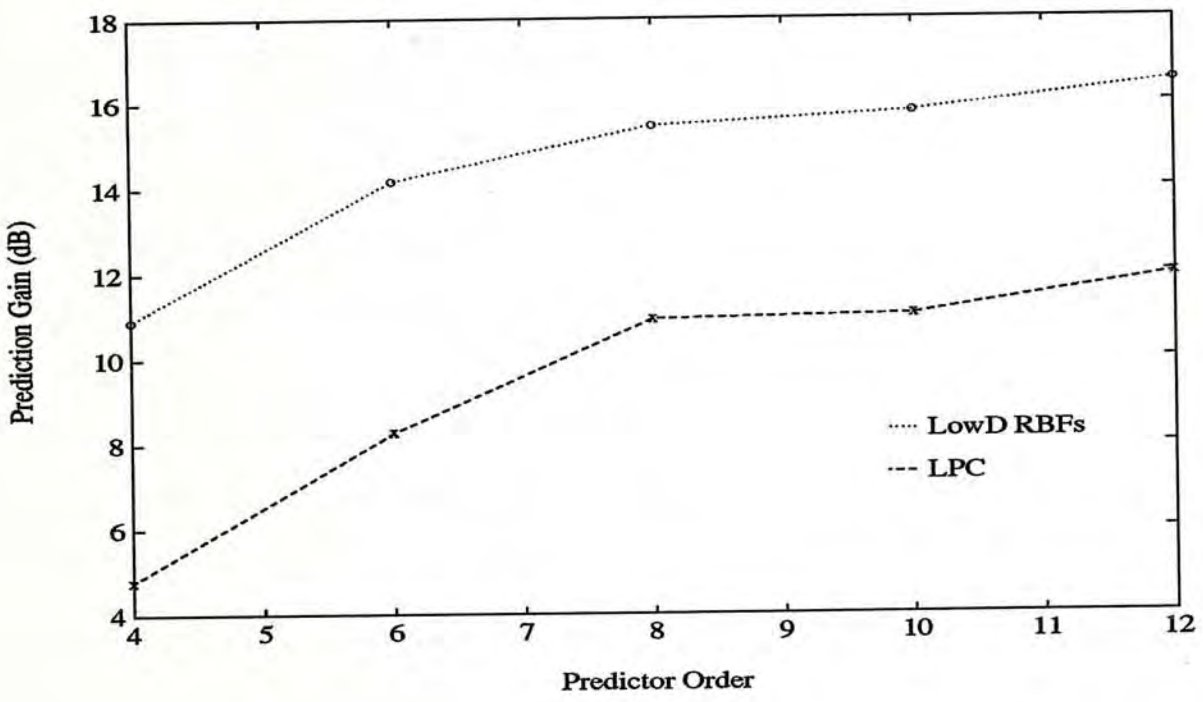

Figure 3.17: The prediction gain as a function of predictor order for LowD RBFs and LPC techniques.

3.2.2 Performance Test in Noisy Conditions

To study the performance of the LowD RBFs in a noisy condition. A different segment of speech from 0.1s to 0.2s (as shown in figure 3.18) are extracted as a new training set. Based on the same construction parameters used previously, except the following,

- **INITIAL MSE THRESHOLD=** 0.000075
- **FINAL MSE THRESHOLD=** 0.000062

the resulting LowD RBF network generates the following number of nodes with a maximum dimension of 2.

- Number of 1-D nodes = 13
- Number of 2-D nodes = 173

Figure 3.20 shows the prediction output of this network. In this test, the prediction gain of the network is only 5.13dB compared to 6.06dB for LPC. The results indicate that LowD RBFs do not perform as good as LPC in which a high noise level is presented. To explain this situation, let us first recall that both growing and pruning heuristics are based on the measurement of weight change variance. Unfortunately, this variance is indeed an unreliable selection index in the presence of noise. It is due to the fact that the Gaussian nodes with the largest noise level will always have the highest variance, regardless of their usefulness to the function approximation. For the sake of completeness, the residuals of NLP and LPC techniques are also shown in figures 3.19 and 3.21 respectively.

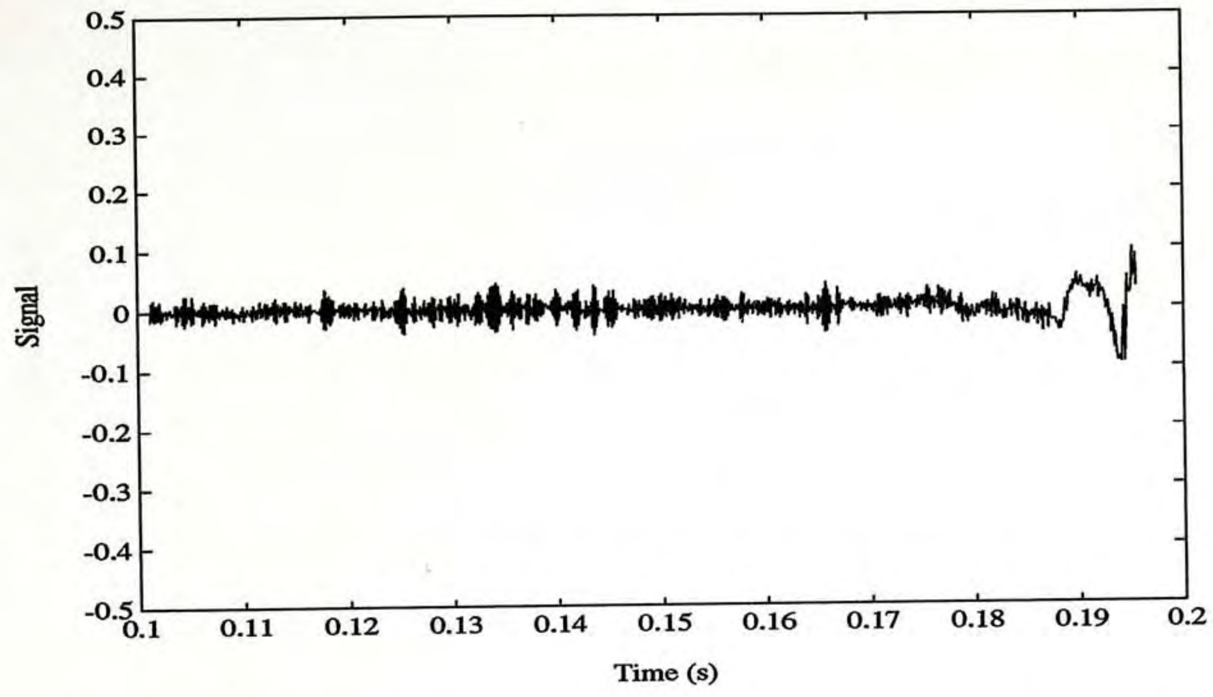

Figure 3.18: The original speech signal in the segment from 0.1s to 0.2s.

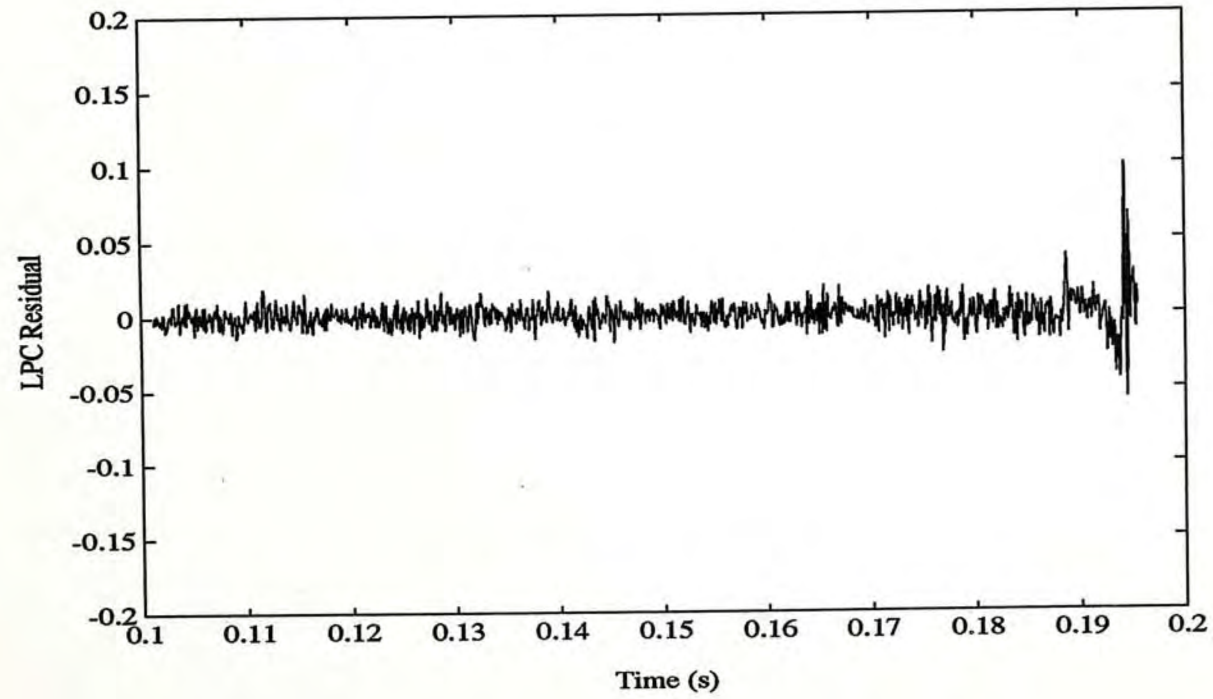

Figure 3.19: The residual after the linear prediction.

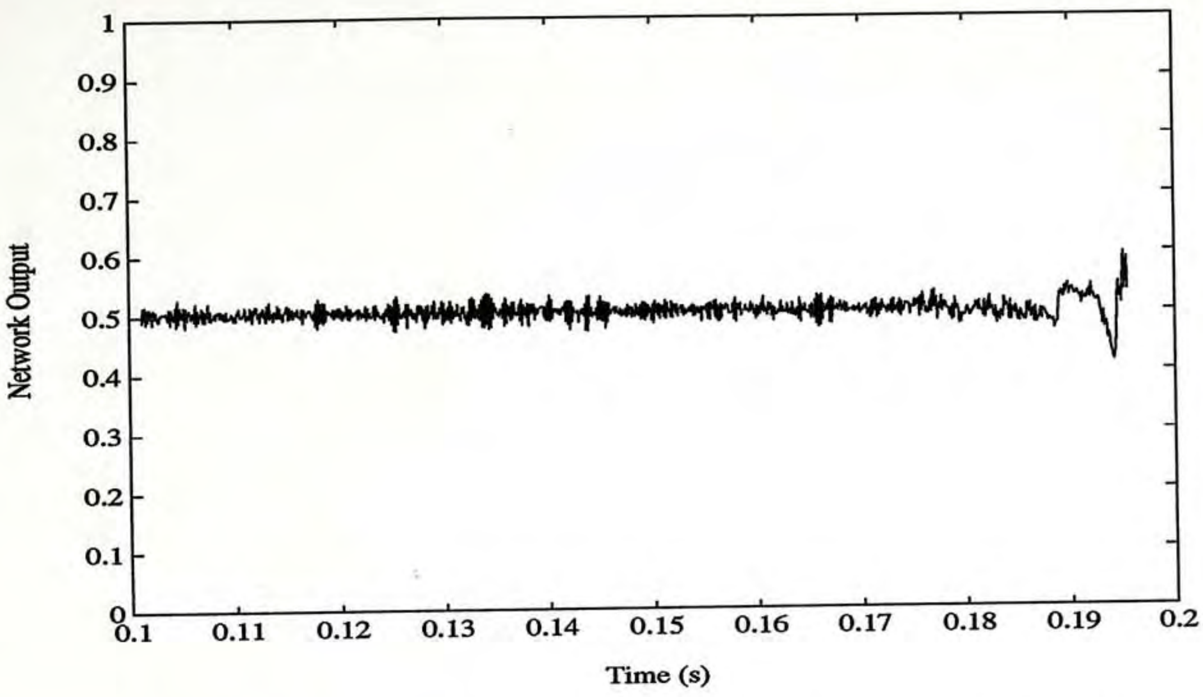

Figure 3.20: The prediction output of the LowD RBFs network.

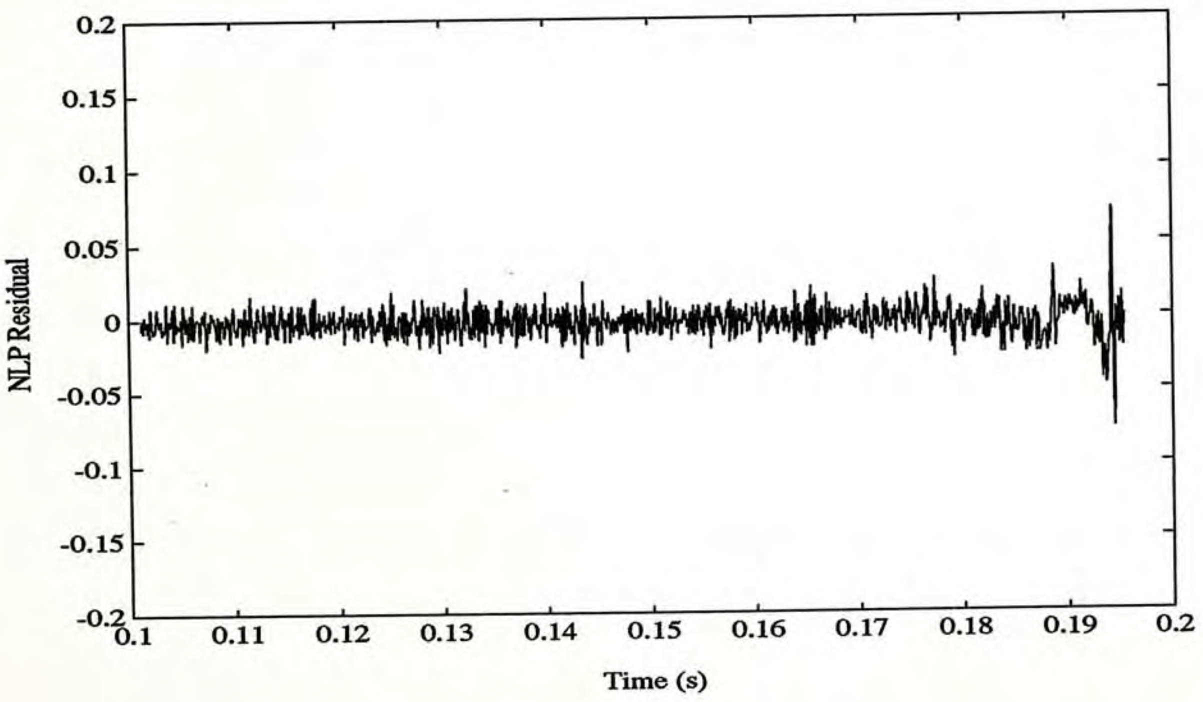

Figure 3.21: The residual after the nonlinear prediction.

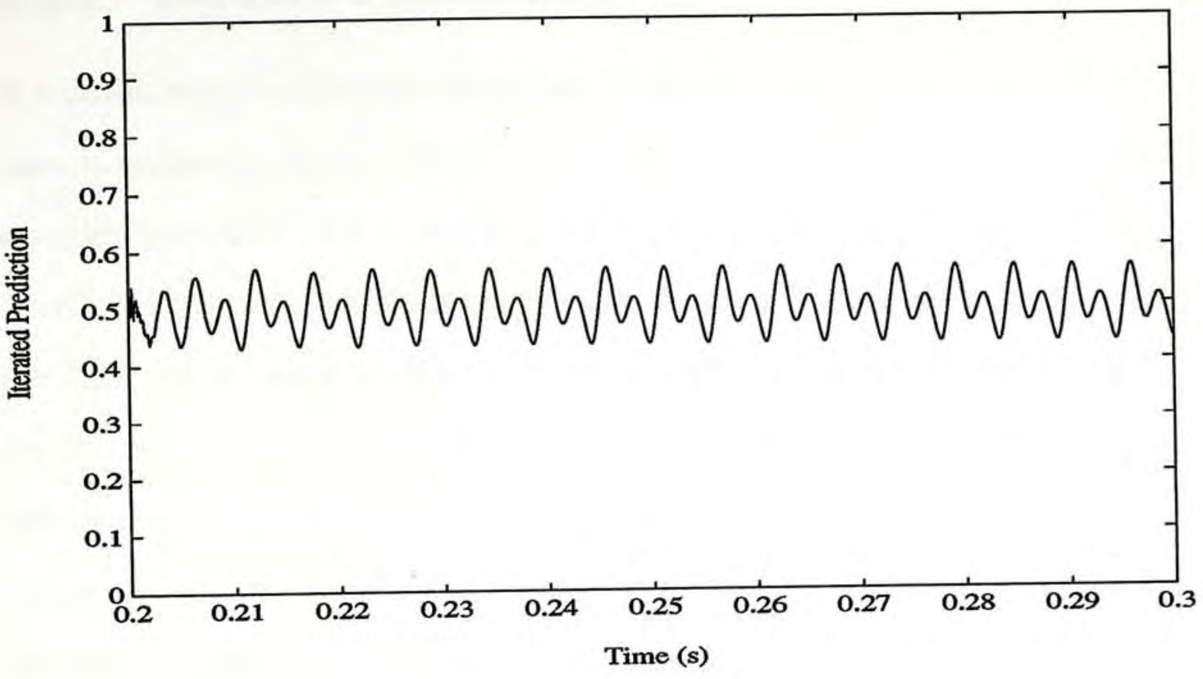

Figure 3.22: The iterated prediction by LowD RBFs in the segment from 0.2s to 0.3s.

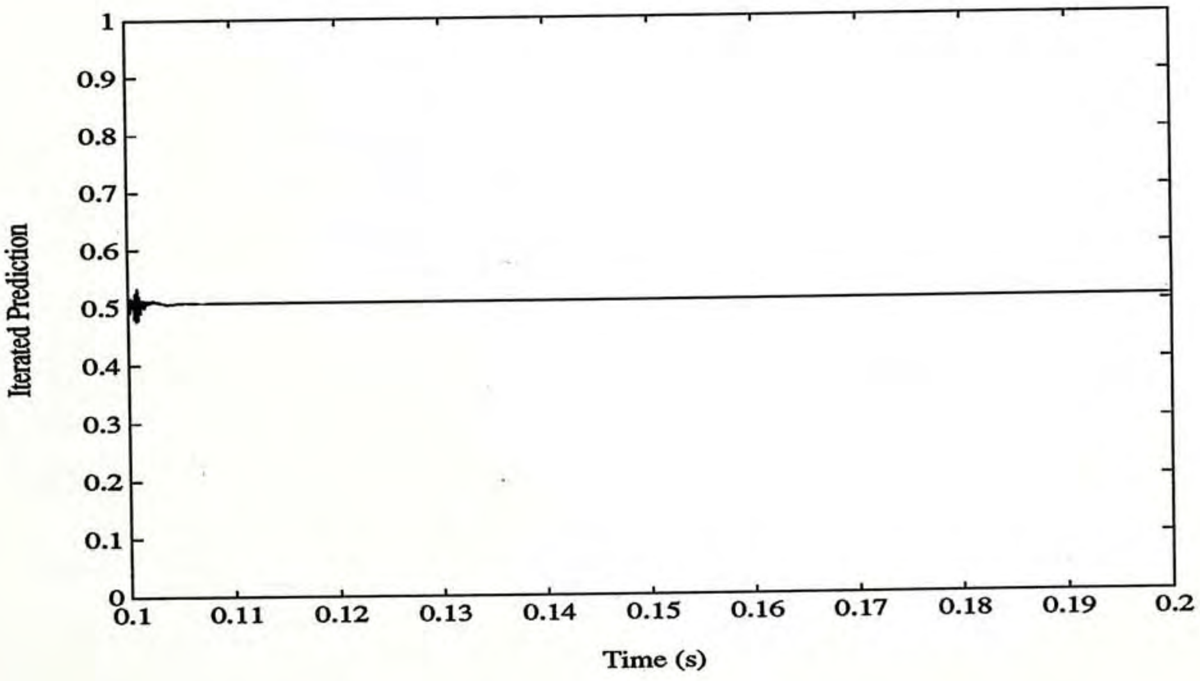

Figure 3.23: The iterated prediction by LowD RBFs in the segment from 0.1s to 0.2s.

3.2.3 Iterated Prediction of Speech

It is interesting to take the above LowD RBF networks trained by two different speech segments, and put them into the iterated predictions. Using the first 12 samples from each of the segments as a "seed", and iterate the corresponding LowD RBFs produced the waveforms in figures 3.22 and 3.23. Note that in figure 3.22, the network is able to produce a quasi-periodic speech-like waveform. On the other hands, the iterated network output in figure 3.23 exhibits a heavily damped structure of the original noisy signal. In both cases, the iterated waveforms are not the same as the actual waveforms. Actually, the two LowD RBFs produce the low-passed version of the original waveforms. Thus, it is possible that the LowD RBFs is able to encapsulate a few important relevant dimensions of underlying structure of the speech production system.

Chapter 4

Conclusion

4.1 Discussions

In this thesis, we presented an alternative architecture of the Gaussian RBF networks, which we called LowD RBFs for approximating continuous functions in high-dimensional input spaces, based on the assumption that most of the input dimensions are redundant. A new dynamic construction algorithm composed of growing and pruning processes was developed for building LowD RBFs of variable sizes and dimensions, determined by the inherent dimension of the function to be approximated. The approach used in this algorithm is to grow the number of LowD RBF nodes larger than is necessary and then prune it back to yield a smaller and more efficient structure. Since both growing and pruning criterions use terms of the steepest descent dynamics that are readily available during the normal course of LMS training, only negligible extra computational overhead is needed to implement the algorithm.

Empirical results in the previous chapter showed that the performance of the LowD RBFs with respect to accuracy and efficiency was comparable to or even surpassed most of the conventional approaches. This was especially true if the training set contains redundant input dimensions. Since the behavior of the construction algorithm is problem dependent, we compared the LowD RBFs to a host of other models on two quite different function approximation problems: For the Mackey-Glass equation, we found that the LowD RBFs learn much faster than BP, while retaining the same precision as conventional RBFs. In the nonlinear prediction of speech signals, the LowD RBFs were able to produce an additional 4dB of prediction gain than that obtained using the LPC linear predictor.

4.2 Limitations and Suggestions for Further Research

We have discussed the advantages of the construction algorithm and demonstrated the outstanding performance of the resulting LowD RBF networks. However, one should not overlook the mechanism used to generate the LowD RBFs. It is important to realize that both the growing and pruning procedures are only heuristics. There is no way to guarantee that they can produce a optimal network structure. The behavior of these heuristics may perform badly under certain conditions such as in the presence of high noise level. Further research is necessary to improve the robustness of the growing and pruning methods.

Bibliography

[1] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A Learning Algorithm for Boltzmann Machines. *Cognitive Science*, 9:147–169, 1985.

[2] M. D. Alder, R. Togneri, and Y. Attikiouzel. Dimension of the Speech Space. *IEE Proceedings I (Communications, Speech and Vision)*, 138(3):207–214, June 1991.

[3] D. S. Broomhead and D. Lowe. Multivariable Functional Interpolation and Adaptive Networks. *Complex Systems*, 2:321–355, 1988.

[4] G. A. Carpenter and S. Grossberg. A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition machine. *Computer Vision, Graphics, and Image Processing*, 37:54–115, 1983.

[5] M. C. Casdagli. Nonlinear Prediction of Chaotic Time Series. *Physica D*, 35:335–356, 1989.

[6] R. M. Dillon and C. N. Manikopoulos. Neural Net Nonlinear Prediction for Speech Data. *Electronics Letters*, 27(10):824–826, May 9 1991.

[7] F. Fallside. Analysis of Linear Predictive Data as Speech and of ARMA Process by a Class of Single-Layer Connectionist Models. In F. Fogelman

Soulié and J. Hérault, editors, *Neurocomputing: NATO ASI Series Vol F68*, pages 265–283. Springer-Verlag, Berlin, 1990.

[8] J. D. Farmer. Chaotic Attractors of an Infinite Dimensional System. *Physica D*, 4:366, 1982.

[9] J. D. Farmer and J. J. Sidorowich. Predicting Chaotic Time Series. *Phys. Rev. Lett.*, 59(8):845–848, August 1987.

[10] S. Grossberg. *Neural Networks and Natural Intelligence.* MIT Press, Cambridge MA, 1988.

[11] E. J. Hartman and J. D. Keeler. Predicting the Future: Advantages of Semilocal Units. *Neural Computation*, 3:566–578, 1991.

[12] J. J. Hopfield. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proc. Natl. Acad. Sci.*, 79:2554–2558, April 1982.

[13] N. S. Jayant. Digital Coding of Speech Waveforms: PCM, DPCM, and DM Quantizers. *Proceedings of the IEEE*, 62:611–632, May 1974.

[14] E. D. Karnin. A Simple Procedure for Pruning Back-Propagation Trained Neural Networks. *IEEE Transactions on Neural Networks*, 1(2):239–242, June 1990.

[15] T. Kohonen. Self-Organizing Formation of Topologically Correct Feature Maps. *Biological Cybernetics*, 43:59–69, 1982.

[16] T. Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, New York NY, 2nd edition, 1988.

香港中文大學圖書館藏書

[17] B. Kosko. Adaptive Bidirectional Associative Memories. *Applied Optics*, 26:4947–4960, December 1 1987.

[18] A. Lapedes and R. Farber. How Neural Nets Work. In D. Z. Anderson, editor, *Neural Information Processing Systems*, pages 442–456. Am. Inst. of Physics, New York NY, 1988.

[19] D. Lowe and A. Webb. Adaptive Networks, Dynamical Systems, and the Predictive Analysis of Time Series. In *Proceedings of First IEE International Conference on Artifical Neural Networks*, pages 95–99, London UK, October 1989.

[20] M. C. Mackey and L. Glass. Oscillation and Chaos in Physiological Control Systems. *Science*, 197:287, 1977.

[21] J. Makhoul. Linear Prediction: A Tutorial Review. *Proceedings of the IEEE*, 63:561–580, 1975.

[22] J. Moody. Fast Learning in Multi-Resolution Hierarchies. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, pages 29–39. Morgan Kaufmann, San Mateo CA, 1989.

[23] M. Niranjian and F. Fallside. Neural Networks and Radial Basis Functions in Classifying Static Speech Patterns. CUED/F-INFENG/TR 22, Cambridge University Engineering Dept. Technical Report, 1988.

[24] T. Poggio and F. Girosi. Networks for Approximation and Learning. *Proceedings of the IEEE*, 78(9):1481–1497, September 1990.

[25] M. J. D. Powell. Radial Basis Functions for Multivariable Interpolation: A Review. In J. C. Mason and M. G. Cox, editors, *Algorithms for Approximation*, pages 143–167. Clarendon Press, Oxford, 1987.

[26] L. R. Rabiner and R. W. Schafer. *Digital Processing of Speech Signals.* Prentice-Hall, Englewood Cliffs NJ, 1978.

[27] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Internal Representations by Error Propagation. In D.E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing*, chapter 8. MIT Press, Cambridge MA, 1986.

[28] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing*, volume I & II. MIT Press, Cambridge MA, 1986.

[29] D. E. Rumelhart and D. Zipser. Feature Discovery by Competitive Learning. *Cognitive Science*, 9:96–112, 1985.

[30] T. D. Sanger. A Tree-Structured Adaptive Network for Function Approximation in High-Dimensional Spaces. *IEEE Transactions on Neural Networks*, 2(2):285–293, March 1991.

[31] T. D. Sanger. A Tree-Structured Algorithm for Reducing Computation in Networks with Separable Basis Functions. *Neural Computation*, 3:67–78, 1991.

[32] T. D. Sanger. Using LMS Trees for Image Processing. In *Proceedings of International Joint Conference on Neural Networks*, pages 405–410, Seattle WA, July 1991.

[33] D. F. Specht. Probabilistic Neural Networks. *Neural Networks*, 3:109–118, 1990.

[34] B. Townshend. Nonlinear Prediction of Speech Signals. In M. Casdagli and S. Eubank, editors, *Nonlinear Modeling and Forecasting, SFI Studies in the Sciences of Complexity*, chapter Proc. XIII, page 21. Addison-Wesley, Redwood City CA, 1991.

CUHK Libraries
000360292