APPLICATION OF LEARNING THEORY IN NEURAL MODELING OF DYNAMIC SYSTEMS

By

Kayvan Najarian

B. Sc. (Telecommunications) Sharif University of Technology, Tehran, Iran

M. Sc. (Biomedical Engineering) Amirkabir University of Technology, Tehran, Iran

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

THE UNIVERSITY OF BRITISH COLUMBIA

May 2000

Abstract

Neural networks have been successfully used to model a number of complex nonlinear systems. Although neural networks can create successful models of some nonlinear systems, they are known to overfit the data in some other applications. Therefore, in order to use neural networks reliably, it is necessary to explore the conditions under which neural models perform equally well on the testing and training data sets. This calls for the design of the neural models that create a balance between the testing and training performances.

The newly introduced Probably Approximately Correct (PAC) learning theory addresses the issue of testing-training balance. However, conventional PAC learning only allows static modeling and cannot be applied to dynamic models. In this thesis, PAC learning is extended to more general learning schemes that handle dynamic modeling tasks. The resulting PAC paradigms are then applied to assess the learning properties of several families of dynamic neural networks, including Radial Bases Functions Networks , single-hidden-layer Sigmoid Neural Networks, and Volterra Networks.

Another concern with the use of neural networks for some dynamic modeling tasks is the issue of stochastic stability. Little is known about the stochastic stability of many neural models used in practical applications. The lack of knowledge over the stability of neural models further limits the use of such models. In this thesis, sufficient conditions for stochastic stability of different families of neural networks are presented, which address the above mentioned concern .

Based on the resulting learning frameworks, evolutionary algorithms are then presented that search for a suitable suitable dynamic neural modeling which perform equally well on testing and training data. The evolutionary algorithms are then used for modeling of two applications. The first application deals with next-scan-estimation of a two-dimensional paper basis weight measurement on a paper machine. In the second application, a neural model for a neuromuscular blockade system is developed. The results indicate that accurate and reliable dynamic neural models can be obtained, provided that the learning complexity of such models are controlled during the training procedure.

Table of Contents

List of Tables

List of Figures

Acknowledgement

I would like to gratefully acknowledge the financial support of the research and myself by The Department of Electrical and Computer Engineering as well as The Pulp and Paper Center of The University of British Columbia. I would also like to thank Honeywell-Measurex Devron Inc. for providing me with the paper machine data. I also like to thank the following individuals for their great help and support:

- To my supervisors, Professors Michael S. Davies and Guy A. Dumont, thank you for your guidance, encouragement, and your friendship.
- To my supervisor, Professor Nancy E. Heckman, I deeply appreciate your helping me with the mathematical aspects of the thesis and giving me your invaluable guidelines and encouragement.
- To my wife, my son, my parents and my wife's parents, thank you for all your support, encouragement and patience during the period of completing this thesis.
- To my colleagues at ECE and PPC, Greg E. Stewart, Jan A. Bergstrom, Ahmed A. Ismail, Stevo Mijanovic and Shahram Shirani, thank you for your advice, helpful discussions and friendship.

Chapter 1

Introduction

1.1 Background and Motivation

Developing models from observed data, or function learning, is a fundamental problem in many fields, including statistical data analysis, signal processing, control, forecasting, and artificial intelligence. This problem is closely related to concepts such as function estimation, function approximation, system identification, and regression analysis. Recently, neural networks have become popular tools in nonlinear function estimation due to their ability to "learn" and "generalize" rather complicated functions. Multi-layer Sigmoid Neural Networks (SNN's) are the most frequently used type of neural networks in practical applications. However, neural structures can overfit the data, i.e. they may perform successfully on the training set of data and poorly on the testing one. This is mainly because the fact that in many applications the complexity of the neural models is allowed to grow freely during the training process. Therefore, it is necessary to evaluate and control the complexity of neural models.

Unfortunately, there exists a lack of knowledge over the computational complexity and generalization capabilities of almost all existing neural architectures and their training methods. This lack of knowledge has created a serious concern over using neural networks for sensitive applications where reliability plays a vital role. The need for reliable use of the computational capabilities of neural networks in modeling of complex system calls for a more solid approach towards the concepts of "learning", as well as its application to neural modeling.

Informally speaking, learning theory addresses the ability of a model to deal correctly with the data which were not included in the training set, and the amount of calculation required to perform the approximation process. The above concept, once applied to

neural networks, can evaluate the complexity of neural models and avoid overfitting. In other words, the learning theory can answer some fundamental questions such as: How confident can one be that the performance of a neural approximator developed to model an unknown function is within a pre-specified proximity from the actual unknown function? What is the sufficient number of samples required to train the network (normally referred to as sample complexity of network)? Is a network A inherently "better" at generalization performance than a network type B? Does a network type A require fewer training examples to achieve a given generalization performance than a network of type B? How would a new neuron added to a particular neural structure affect the overall computational learning properties of the network? How are the learning properties of a neural network used for static modeling different from those of the same neural structure used for dynamic modeling? How can one ensure that the data are not overfitted by a neural model? While these questions reflect legitimate concerns and can be found in the literature of many scientific and engineering fields, some will have to be further defined and specified if a precise answer is needed. For example, the word "better" must be defined clearly.

It is important to notice that experimental investigation or simulation results cannot be guaranteed to provide us with reliable answers to this type of questions; they tell us only about the generalization performance of specific networks applied to specific problems. Therefore, a more theoretical approach towards the concept of learning is required to address the computational performance of neural networks.

In 1984, to formulate the idea of learning, Valiant [1], based on computational learning theory, proposed the Probably Approximately Correct (PAC) learning framework to estimate the generalization capabilities of a given model. This concept created a foundation to assess the generalization properties of different modeling structures including neural networks [12], [13], [14]. However, due to some computational problems, evaluation of the PAC learning property for many types of models (including neural networks) remained as an intractable open problem. Some of these learning problems are still open, while a number of such problems were handled as described below.

A parallel development in the theory of empirical processes was to have a profound

impact on learning theory. Vapnik and Chervonenkis, working on a theory of uniform convergence of relative frequencies to probabilities, introduced the concept of Vapnik-Chervonenkis (VC-) dimension which proved to be a useful tool in empirical estimation of actual probabilities of events [51]. In 1989, the publication of the paper "Learnability and Vapnik-Chervonenkis Dimension" by Blumer et al. [4] represented another milestone in the development of learning theory. This paper made a connection between PAC learning theory and some complexity measures including the VC-dimension. This link made evaluation of the PAC learning property for a family of functions tractable by applying the VC-dimension (or similar complexity measures) of the family, provided that the complexity measure of the family is finite. The resulting theory provided a theoretical foundation for comparing different models from the standpoint of generalization and computational capabilities.

In the 1990s, there was a burst of publications on learning properties of different families of neural networks using the ideas of PAC learning. A number of those papers introduced upper bounds of VC-dimension and similar complexity measures for particular neural architectures in order to show that those families of networks are PAC learnable. Others found the families of neural networks for which complexity dimensions were not bounded, and consequently the families which failed to be PAC learnable ([3], [8], [48], [17], [18], [21], [44], [45], and [36]).

Meanwhile, based on practical results, two families of neural structures have been long known as the most successful architectures for modeling and control applications. The two families are:

- Single-hidden layer Sigmoid Neural Networks (which is hereafter referred to as "SNN's"), and
- Generalized Single-Layer Networks (GSLN's) which includes some popular subfamilies of networks such as: Radial Basis Function Networks (RBFN's), and Volterra networks.

A school of researchers suggests that the general class of GSLN's, using a defined Least Mean Squared (LMS) method for training, is preferred over SNN's due to the relative

amount of computation involved in the training phase and the guaranteed convergence to the global optimum in the parameter space. For example, it is suggested in [46] that in the case of binary-output networks, from the standpoint of PAC learning, GSLN's display a more desirable performance than SNN's, as they can be shown to require fewer examples to learn the unknown functions. However, the size of the parameter space is important and excluding the parameter space from the comparison process makes the results unclear. Some families of GSLN's (including RBFN's) are not truly linear in the parameters in the strict sense and, depending on the set of parameters to be trained, the model may or may not be linearly parameterized. For example, in the case of RBFN's, if the training algorithm is allowed to update the width or the centers of the basis functions, the model is no longer linear in its parameters. As a result, LMS can no longer be used for such cases and no guarantee over the convergence of the parameters to a globally optimal solution can be given. This will be further discussed in Chapter 2.

In applications of neural modeling, even those with satisfactory results, attention has seldom been paid to important issues such as the accuracy of the approximation, confidence of convergence (of the model) to the unknown function, the possibility of overfitting the data, and the sample complexity of the algorithm. Therefore, results are useful for individual cases, but seldom lead to general properties that might guide future work. This suggests that conventional PAC learning might be a useful framework to address such issues in neural modeling. However the learning theory literature indicates that almost all PAC learning results can be applied only to neural modeling of static systems. Since, in many applications, neural networks are used to create dynamic models of unknown systems, it is desirable that learning theory be extended to include dynamic modeling. This would enable a typical task of dynamic neural modeling to be considered as a learning process and to evaluate PAC learning properties of the process. The most commonly used dynamic models in engineering applications are "Finite Impulse Response" (FIR) and "Auto Regressive eXogenous" (ARX) models. In this thesis, new extensions of PAC learning are introduced that include dynamic modeling (both FIR and ARX). The results of the new learning schemes are applied to dynamic modeling with different families of neural networks.

In the case of an ARX modeling task, it is also essential to deal with other issues such as stability, and the statistical dependency of the resulting processes. As will be shown later in the thesis, without addressing such issues, it is not possible to even define a meaningful learning paradigm that includes an ARX modeling task. Moreover, evaluating the stability of a system is an important part of a typical dynamic modeling task. Dynamic models for which stability have not been guaranteed can not be used in many industrial applications.

To show how the learning results can be applied in real world, two nonlinear systems will be modeled using neural models. The first system is the two-dimensional scanning sensor for monitoring the basis weight of the paper being produced in a typical paper machine. The basis weight sensors take the measurements as they move laterally along the moving sheet of paper being produced. These sensors are occasionally taken off-line for maintenance purposes or even failure. While the sensors are off-line, the paper machine is still running. As a result, for some time there are no direct readings of the basis weights for the paper being produced. Neural networks are applied to use the information before the sensors go off-line to estimate the basis weights while direct measurement is not possible.

The second system to be modeled is a neuromuscular blockade system. In many surgical procedures, two types of drugs are used for anesthesia. The first drug deals with unconsciousness of the patient and controls the patient's nervous system so as to make sure that the patient does not feel the pain. However, an unconscious patient may make involuntary muscle movements. The second type controls the motion of muscles during the surgery. Such drugs block the neuromuscular activities of the muscles by generating a temporary paralysis in muscles, and so make sure that no unwanted motion occurs during the surgery. In this research, neural networks are used to model the nonlinear relation between the quantity of the injected drug and the rate of paralysis (muscle relaxation).

1.2 Contributions

The main contributions of this thesis are as follows.

- The definition of PAC learning is extended to new learning schemes which include FIR and ARX dynamic modeling procedures. The resulting new learning schemes are then applied to different families of SNN's and GSLN's, and the learning properties of such neural models are assessed. The sample complexity (the size of a training data set which guarantees reliable modeling) of neural dynamic modeling with different families of SNN's and GSLN's is bounded. The specific bounds on the sample complexity, besides being useful for a systematic modeling task, give the form of the functional dependency between the accuracy, the statistical confidence, the characteristics of neural models and the size of the training data. These dependencies are then used to define meaningful complexity measures based on learning properties of a neural model. These dependencies together with the resulting complexity measures are even more practically useful than the bounds themselves, as described later. Also, using the bounds, the learning properties of dynamic neural modeling with different families of neural networks are quantitatively compared. This comparison helps the user prefer one type of neural models over the others.

- A set of sufficient conditions for stochastic stability of the sigmoid neural ARX models is presented. These conditions allow quantitative evaluation of stability for some important families of dynamic neural models.

- Based on the functional dependencies obtained in the sample complexity bounds, new cost functions that create a balance between the empirical error and the learning complexity of different families of neural networks (both for FIR and ARX scenarios) are presented. These learning-based cost functions are shown to avoid overfitting of the data.

- Using the above mentioned cost functions, new algorithms based on Evolutionary Programming (EP) are introduced that can be used to identify neural models of complex nonlinear systems using the learning and computational properties of the models. These algorithms minimize the learning-based cost functions and are used in modeling tasks where the available training data sets are small. The proposed algorithms are meant to avoid overfitting the data and to provide models that perform equally well against the training and testing data sets.

- The EP-based algorithms for training of neural FIR models are used to predict the future basis weight scans of the paper in a typical paper machine.
- The EP-based algorithms for training of neural ARX models are used to successfully estimate the nonlinear behaviour of a neuromuscular blockade system.

1.3 Thesis Outline

The outline of the thesis is as follows.

- Chapter 2: An extension of PAC learning which includes nonlinear FIR modeling is presented. Then, using the new learning scheme, the learning properties of two families of SNN's, two families of Radial Basis Functions (RBFN's) and a family of Volterra networks are evaluated. These results include bounds on the sample complexity of FIR neural modeling with each of the neural structures. The learning properties of different families are also compared with each other.
- Chapter 3: In Chapter 3 the focus is given to the FIR modeling tasks, where a limited number of training data points are available. The learning results of Chapter 2 are used to generate EP-based algorithms that generate neural models of such systems. In order to test the performance of the proposed algorithm, a number of numeric simulations are given in this chapter.
- Chapter 4: One of the proposed evolutionary algorithms developed in Chapter 3 is applied to predict the basis weight data on the scanning system of a paper machine.
- Chapter 5: This Chapter considers neural ARX modeling using SNN's. The stochastic stability, geometric ergodicity and geometric α-mixing (strong mixing) properties of the SNN's are addressed, and a set of sufficient conditions for such properties is presented. Then, the conventional PAC leaning scheme is extended to learning with geometrically α-mixing data. This leads to a framework to assess the learning properties of dynamic SNN's. Based on the learning results, two evolutionary neural ARX modeling algorithms are presented and tested against a set of numeric examples.

- Chapter 6: One of the EP-based algorithms is used for neural ARX modeling of the neuromuscular blockade system.
- Chapter 7: The conclusions of the research along with suggested future work are given.

It is necessary to mention the following remarks regarding the thesis.

1. Throughout the thesis, some existing lemmas or theorems have been mentioned and reviewed. In the case of such theorems and lemmas, the name of the person who presented and proved the theorem and the corresponding reference are mentioned at the beginning of the statement. All other lemmas and theorems (i.e. the statements that do not start with a name) are the ones proposed and proved by us in this thesis.
2. If a technique used for the proof of a theorem is inspired or motivated by a certain methodology introduced in other works, the name of the person and the reference to the work is mentioned before the theorem.
3. Due to the complex notation used in some chapters of the thesis, it was necessary to modify a very small portion of the notation used in chapter 5. More specifically, the notation is kept the same throughout chapters 2, 3 and 4, while starting from chapter 5, a few letters and symbols used in the previous chapters have been redefined to refer to new concepts. Also, notice that the notation used in chapters 5 and 6 is exactly the same.
4. The main objective of the applications given in the thesis is to show that the introduced neural algorithms can be used to model industrial and biological systems. However, in none of the applications are neural networks claimed to be the best type of structures for modeling of the given systems. In other words, the applications given here are only used to illustrate the performance of learning-based neural modeling.

Chapter 2

A Learning Framework For FIR Modeling

2.1 Introduction

In a modeling procedure an unknown function "f" is to be estimated to the prespecified accuracy "ϵ" and statistical confidence "$(1-\delta)$". In order to perform the estimation, using a set of input-output training data generated by the function f, an approximator function "h" is found to model f. The modeling of an unknown system f with a feedforward neural network h can be considered as a typical example of this procedure. Probably Approximately Correct (PAC) learning theory, proposed by Valiant [1], relates the accuracy and confidence of the modeling task. PAC learning and other similar learning schemes allow quantitative evaluation of the learning properties of modeling procedures in which the data are independently and identically distributed (i.i.d.) in accordance with a probability measure P. The available results in PAC learning theory are applicable only to static modeling tasks as they make use of Hoeffding's inequality [52] which is applicable only to i.i.d. data [20], [19]. However, in many real modeling procedures, the assumption of data being i.i.d. is violated. As indicated in [39], one important group of applications to which the results of learning theory with independent data are not directly applicable is "Nonlinear Finite Impulse Response" (NFIR) modeling, where the output depends on the present as well as the past inputs. As a result, in an NFIR model, the inputs at times "t" and "$t+1$" are correlated and consequently dependent [31], [32]. The importance of FIR models comes from the fact that dynamic systems can often be efficiently approximated by appropriate FIR models. The main contribution of the present chapter is the extension of the PAC learning theory to modeling of NFIR procedures.

This chapter also establishes the learning properties of a general family of neural models. These learning properties are in the form of inequalities that relate the accuracy and statistical confidence of the models to the number of training data used for modeling.

Among different neural FIR models, feedforward neural networks and radial basis functions have been applied in many modeling applications. However, despite the popularity of feedforward neural networks, the training methods available for such neural models are relatively complicated and the convergence to the optimal set of weights (parameters) is not guaranteed. In the case of RBFN's, provided that only the weights of the basis functions are to be trained, the optimization procedure becomes a linearly parameterized one. As a result, for such RBFN's simple minimization techniques can be used in training the model. Moreover, such minimization methods guarantee convergence to the optimal set of parameters. As a result, radial basis functions networks have recently been used in different modeling applications [38], [25], [47]. It can be observed that restricting the optimization procedure to the weights of the basis functions reduces the computational capabilities of RBFN's but is necessary for linear dependency. Including other parameters of RBFN's (such as the ones that change the basis functions) optimization process makes such models suffer from the same problems as mentioned for SNN's.

In this chapter, the learning properties of RBFN's, SNN's, Volterra networks, and simple linear models are assessed and upper bounds for the sample complexity (the minimum size of the training data) for NFIR modeling using such neural structures are presented. The chapter is organized as follows: Section 2.2 gives the basic definitions of the PAC learning theory. The idea of PAC learning with i.i.d. data is extended to PAC learning with m-dependent data in Section 2.3. Sections 2.4, 2.5, 2.6 give specific results on the learning properties of FIR modeling using general families of RBFN's, SNN's, Volterra Networks, and linear models respectively. The main results of these sections (on learning properties of different families of neural networks) are given in Theorems (2.4.1.1), (2.4.2.1), (2.5.1.1), (2.5.2.1), (2.6.1), and (2.7.1). In Section 2.8, the results of previous sections are applied to a typical task of FIR modeling, and the results of modeling with different neural structures are presented. Section 2.9 describes the concept of model-free learning, which extends the results of the previous chapters. The main results of this section are given in Theorem (2.9.1). In Section 2.10 the results of the chapter are discussed. Section 2.11 gives the summary of the chapter.

2.2 Definitions

In this section, some of the basic concepts of stochastic learning theory, including m-dependence, PAC learning with i.i.d. data and the empirical risk minimization algorithm [50], are reviewed.

The first concept to be defined here is "σ-algebra".

Definition 2.2.1 *Suppose X is a set. A (nonempty) collection $\mathcal{S}$ of subsets of X is said to be a σ-algebra if it satisfies the followings.*

1. *$\mathcal{S}$ is closed under complementation; i.e., if $A \in \mathcal{S}$, then $A^c \in \mathcal{S}$.*
2. *$\mathcal{S}$ is closed under countable union; i.e., $A_i \in \mathcal{S}$ for $i = 1, 2, \ldots$, then $\bigcup_{i=1}^{\infty} A_i \in \mathcal{S}$.*

The smallest σ-algebra of subsets of X that contains every closed subset of X is called the "Borel σ-algebra".

The following elements will be used throughout the chapter:

- A set X,
- A σ-algebra $\mathcal{S}$ of subsets of X,
- A fixed known probability measure P on the measurable space $(X, \mathcal{S})$,
- A function set $\mathcal{F}$ of measurable functions $f : X \to [-1/2, 1/2]$. [The interval [-1/2,1/2] can be replaced throughout by any bounded interval.]

Now, consider a modeling task in which the unknown function $f \in \mathcal{F}$ is to be estimated. In order to perform the estimation, a set of training data is to be generated as: $z_n = \{(x_i, f(x_i))\}_{i=1}^n$. Also, assume that each x_i is independently and identically distributed according to the probability measure P. An algorithm $A_n : (X \times [-1/21/2])^n \to \mathcal{F}$, based on the training data z_n, generates a function $h_n \in \mathcal{F}$ as an approximator of f, i.e.:

$$h_n(f, z_n) = A_n\left[(x_1, f(x_1)), \ldots, (x_n, f(x_n))\right] \tag{2.1}$$

PAC learning is defined as follows:

Definition 2.2.2 *Suppose that based on z_n, where $\{x_1, \ldots, x_n\}$ are i.i.d. according to the probability P, the unknown function f is to be approximated by a function h_n. Then, a function set $\mathcal{F}$ is said to be PAC learnable iff an algorithm A_n can be found based on which for any ϵ and δ, there exists "n" such that:*

$$\sup_{f \in \mathcal{F}} \Pr\{d_P(f, h_n) \leq \epsilon\} \geq (1 - \delta) \tag{2.2}$$

where $d_P(f, h_n)$ is a distance between f and h_n defined in terms of the probability P.

Hereafter, assume that $d_P(f, h) = E_P(|f(x) - h(x)|)$. Wherever the meaning is clear the index n in A_n and h_n is dropped.

Another useful concept in function learning is an ϵ-cover of a function set.

Definition 2.2.3 *An ϵ-cover of a function set $\mathcal{F}$ is defined as a set of functions $\{g_i\}_{i=1}^{q}$ in $\mathcal{F}$ such that for any function $f \in \mathcal{F}$, there is a function g_j where: $d_P(f, g_j) < \epsilon$.*

It should be noted that an ϵ-cover for a function set $\mathcal{F}$ may or may not exist. If such a cover set exists, the cardinality (size) of the set depends both on the value of ϵ and the function set $\mathcal{F}$.

An ϵ-cover with minimal size is called a minimal ϵ-cover, and its cardinality is denoted as $N(\epsilon, \mathcal{F}, d_P)$. A specific type of learning algorithm known as "the empirical risk minimization algorithm" is now defined:

Definition 2.2.4 *Let $\epsilon > 0$ be specified, and let $\{g_i\}_{i=1}^{q}$ be an $\epsilon/2$-cover (not necessarily minimal) of $\mathcal{F}$ with respect to d_P.*

Then the empirical risk minimization algorithm is as follows. Consider a set of i.i.d. samples $\{x_1, \ldots, x_n\} \in X^n$, distributed in accordance with P. Define the cost functions: $\hat{J}_i = \frac{1}{n}\sum_{j=1}^{n} |f(x_j) - g_i(x_j)|$, $i = 1, \ldots, q$. Now, the output of the algorithm is a function $h = g_l$ such that: $\hat{J}_l = \min_{1 \le i \le q} \hat{J}_i$.

The last concept to be defined is m-dependence.

Definition 2.2.5 *A sequence of r.v.s $(Y_i)_{i=1}^{n}$ is said to be m-dependent iff for all j and k, r.v.s Y_j and Y_k are independent if $|j - k| > m$. In other words, in a sequence of m-dependent r.v.s, the radius of dependency is limited to the integer m.*

2.3 Extension of PAC learning to m-dependent cases

The existing results in the PAC learning theory are for the cases where input data are i.i.d., because the fundamental inequalities of the PAC learning theory are based on Hoeffding's inequality, which is true only for i.i.d data. This inequality is stated.

Theorem 2.3.1 *(Hoeffding [52]) Suppose that $(Y_i)_{i=1}^{n}$ is a sequence of independent zero-mean r.v.s such that $a_i \le Y_i \le b_i$. Set: $\sup_{1 \le i \le n} |b_i - a_i| = M$, then:*

$$\Pr\left\{\sum_{i=1}^{n} Y_i \ge \alpha\right\} \le \exp\left[\frac{-2\alpha^2}{nM^2}\right] \tag{2.3}$$

As mentioned before, in order to extend the learning theory to include learning with dependent data, it is necessary to find an inequality (similar to Hoeffding's inequality) to be used for m-dependent data. Using Hoeffding's inequality, a new inequality to bound the summation of a sequence of m-dependent r.v.s is now obtained. The method used for the proof is inspired by the proof of the central limit theorem for a sequence of dependent

data by Iosifescu et al. in [35].

Theorem 2.3.2 *Suppose* $(Y_i)_{i=1}^n$ *is a sequence of m-dependent zero-mean r.v.s such that* $a_i \leq Y_i \leq b_i$, $m \geq 1$ *and* $\sup_{1\leq i\leq n} |b_i - a_i| = M$. *Then assuming that* $n = k(m+1)$ *(where k is an integer), we have:*

$$\Pr\left\{\sum_{i=1}^{n} Y_i \geq \alpha\right\} \leq (m+1)\exp\left[\frac{-2\alpha^2}{n(m+1)M^2}\right] \tag{2.4}$$

Proof:

The proof starts with defining the following new variables :

$$Y_j' = \sum_{i=1}^{k} Y_{j+(i-1)(m+1)} \tag{2.5}$$

where: $1 \leq j \leq m+1$. Notice that Y_j' is a sum of independent r.v.s. Also observe that:

$$\sum_{i=1}^{n} Y_i = \sum_{j=1}^{m+1} Y_j' \ . \tag{2.6}$$

Now, define the following events:

$$Ev = \left\{\sum_{i=1}^{n} Y_i < \alpha\right\} \ , \ Ev_j = \left\{Y_j' < \alpha/(m+1)\right\} \ . \tag{2.7}$$

Showing the complement of an event E as E^C:

$$\Pr(Ev^c) \leq \sum_{j=1}^{m+1} \Pr(Ev_j^c) \ . \tag{2.8}$$

From the definition of Ev_j's and Hoeffding's inequality:

$$\Pr(Ev_j^c) \leq \exp\left[\frac{-2\alpha^2}{n(m+1)M^2}\right] . \tag{2.9}$$

Now (2.8) and (2.9) give (2.4) or equivalently:

$$\Pr\left\{\sum_{i=1}^{n} Y_i \leq \alpha\right\} \geq 1 - (m+1)\exp\left[\frac{-2\alpha^2}{n(m+1)M^2}\right] \tag{2.10}$$

which concludes the proof. □ □ □

Notice that if there exists no integer k such that: $n = k(m+1)$, by defining k as: $k = \lfloor n/(m+1) \rfloor$, a similar approach can be followed to extend the result of the theorem to such cases. However, since generally in modeling applications: $n \gg (m+1)$, the new term is negligible compared to the other parts, and as a result, the assumption of $k = n/(m+1)$ being an integer may merely result in gathering a few more data points.

It is important to notice that since in the proof of Theorem 2.3.2, no assumption has been made on the distribution of the data, the resulting inequality is distribution-free. This characteristic of the above theorem makes the results applicable to the learning tasks where the distribution under which the data are generated is unknown.

Theorem 2.3.2 provides a bound for the probability of the summation of a sequence of m-dependent r.v.s which can be applied to extend the definition of the conventional PAC learning to the PAC learning with m-dependent data as described below.

Definition 2.3.1 *Suppose that* $z_n = \{(x_i, f(x_i))\}_{i=1}^n$ *is a sequence of input-output data where* $(x_1, \ldots, x_n)$ *is an m-dependent sequence, marginally distributed according to the probability measure* $P \in \mathcal{P}$. *Then, with the rest of the assumptions exactly the same as the ones made in Definition 2.2.2, a function set* $\mathcal{F}$ *is said to be "PAC learnable with m-dependent data" iff an algorithm A can be found based on which for any* ϵ *and* δ, *there exists "n" such that:*

$$\sup_{f \in \mathcal{F}} \Pr\{d_P(f,h) \leq \epsilon\} \geq (1-\delta)\ . \tag{2.11}$$

Notice that in practice, one can only calculate the empirical distance between f and h based on the available data points. The main objective of the learning theory is to have a quantitative evaluation over the true distance between f and h, i.e. $d_P(f,h)$. Therefore, it is necessary to relate the true distance $d_P(f,h)$ to the empirical distance between the two functions. Next, a lemma is proved that applies Theorem 2.3.2 to evaluate the closeness of the mean value of a function to its empirical mean.

Lemma 2.3.1 *Suppose $\zeta : X \to [0,1]$ is a measurable function with respect to a σ-algebra $\mathcal{S}$ and P is a probability measure on $(X, \mathcal{S})$. A sequence of training data has been generated as: $\{(x_i, \zeta(x_i))\}_{i=1}^n$, where input data is a sequence of m-dependent r.v.s identically distributed in accordance with P. If the mean value, $E_P(\zeta)$, and the empirical mean of ζ, $\hat{E}(\zeta)$, are defined as follows:*

$$E_P(\zeta) = \int_X \zeta(x)\, P(dx)\ , \hat{E}(\zeta) = \frac{1}{n}\sum_{i=1}^{n} \zeta(x_i) \tag{2.12}$$

then:

$$\Pr\{\hat{E}(\zeta) - E_P(\zeta) \geq \epsilon\} \leq (m+1)\exp\left[-\frac{2n\epsilon^2}{(m+1)}\right] \tag{2.13}$$

$$\Pr\{E_P(\zeta) - \hat{E}(\zeta) \geq \epsilon\} \leq (m+1)\exp\left[-\frac{2n\epsilon^2}{(m+1)}\right]\ . \tag{2.14}$$

Proof: First define: $Y_i = \zeta(x_i) - E_P(\zeta)$. Notice that the Y_i's form a sequence of zero-mean m-dependent r.v.s to which Inequality (2.3.2) can be applied with $M = 1$ and $\alpha = n\epsilon$. This will result in Inequality (2.13). Inequality (2.14) can be obtained in the same fashion, assuming $Y_i = E_P(\zeta) - \zeta(x_i)$. □ □ □

Next, in order to further investigate the new learning paradigm, an algorithm A has to be defined. Here, a natural extension of "the empirical minimization algorithm" is introduced, that operates over m-dependent input data. The definition of such an algorithm is straightforward and is defined exactly the same as Definition 2.2.4, except that the algorithm accepts m-dependent rather than i.i.d. inputs. In this chapter, "the empirical risk minimization algorithm" refers to the extended definition.

Now, using the empirical risk minimization algorithm and the results of Lemma 2.3.1, the learning properties of a modeling procedure under the m-dependency of data is evaluated. The proof provided here parallels the proof of a similar learning task under i.i.d. data by Vidyasagar in [2].

Theorem 2.3.3 *With the assumptions of Definition 2.3.1, the empirical risk minimization algorithm results in PAC learning of $\mathcal{F}$ with m-dependent training data to the accuracy of ϵ. In particular:*

$$\sup_{f \in \mathcal{F}} \Pr\{d_P(f, h) \leq \epsilon\} \geq (1 - \delta) \tag{2.15}$$

whenever:

$$n \geq \frac{8(m+1)}{\epsilon^2} \ln \frac{q(m+1)}{\delta} \tag{2.16}$$

Proof: Since $\{g_i\}_{i=1}^q$ is an $\epsilon/2$-cover for $\mathcal{F}$, there exist an index t such that $d_P(f, g_t) \leq \epsilon/2$. Without loss of generality, suppose that $d_P(f, g_q) \leq \epsilon/2$. Again, without loss of generality, suppose that the g_i's are renumbered such that: $d_P(f, g_i) > \epsilon$ for $i = 1, \ldots, k$

and $d_P(f, g_i) \leq \epsilon$ for $i = k+1, \ldots, q$. Note that: $k \leq q-1$. Notice that the error involved in the empirical risk minimization algorithm would be introducing one of the g_i's, where $i = 1, \ldots, k$, as the model h. Define $\hat{J}_t = \hat{E}(|f - g_t|)$, and let the event E be as follows:

$$E = \{\hat{J}_q \leq 3\epsilon/4 \; and \; \hat{J}_i > 3\epsilon/4 \; , \; for \; all \; i = 1, \ldots, k\} \tag{2.17}$$

From the definition of E, it is known that $E \subseteq \{d_P(f, h) \leq \epsilon\}$. Also:

$$\Pr\{E^c\} \leq \Pr\{\hat{J}_q > 3\epsilon/4\} + \sum_{i=k}^{l} \Pr\{\hat{J}_i \leq 3\epsilon/4\} \tag{2.18}$$

Now observe that, for each index i, the cost $\hat{J}_i$ is the empirical mean of the function $|f(.) - g_i(.)|$ based on the available training set. Therefore, Lemma 2.3.1 can be used to evaluate the distance between $\hat{J}_i$, the empirical mean, and $d_P(f, g_i)$, the true mean of the function $|f(.) - g_i(.)|$:

$$\Pr\left\{\hat{J}_q - E_P(|f - g_q|) \geq \epsilon/4\right\} \leq (m+1)\exp\left[\frac{-n\epsilon^2}{8(m+1)}\right] . \tag{2.19}$$

where $n = k(m+1)$. Now, since $E_P(|f - g_i|) = d_P(f, g_q) \leq \epsilon/2$, then:

$$\Pr\left\{\hat{J}_q > 3\epsilon/4\right\} \leq (m+1)\exp\left[\frac{-n\epsilon^2}{8(m+1)}\right] . \tag{2.20}$$

For $\hat{J}_i \leq 3\epsilon/4$ where $i = 1, \ldots, k$, following a similar procedure, the following inequality is obtained:

$$\Pr\left\{\hat{J}_i \leq 3\epsilon/4\right\} \leq (m+1)\exp\left[\frac{-n\epsilon^2}{8(m+1)}\right] . \tag{2.21}$$

Now, since $k \leq q-1$, the maximum error involved in the overall learning process would be: $q(m+1)\exp\left[\frac{-n\epsilon^2}{8(m+1)}\right]$. In other words:

$$\sup_{f\in\mathcal{F}} \Pr\{d_P(f,h) \leq \epsilon\} \geq \left(1 - q(m+1)\exp\left[\frac{-n\epsilon^2}{8(m+1)}\right]\right) \tag{2.22}$$

which results to: $\delta \geq q(m+1)\exp\left[-n\epsilon^2/8(m+1)\right]$. Equivalently, when the values of ϵ and δ are fixed: $n \geq (8(m+1)/\epsilon^2)\ln\frac{q(m+1)}{\delta}$ which concludes the proof. □ □ □

Theorem 2.3.3 provides a constructive method of approximation when the probability is fixed. The extension of the results for larger probability sets is straightforward and requires some knowledge of the probability set. Also, notice that similar results may be obtained for other definitions of $d_P(f,h)$ such as the popular $E_P[f(.)-g(.)]^2$.

The value q in Theorem 2.3.3 is an indication of the complexity of the function set. In the case of the distribution-free learning (where no assumption regarding the probability distribution of data is made), the value of q can be related by another measure of complexity called "P-dimension", which is in turn an extension of Vapnik-Chervonenkis (VC-) dimension [49], [37]. Both q and P-dimension are essentially measures that describe the cardinality of an $\epsilon/2$-cover of the function set. However, in order to construct an $\epsilon/2$-cover of a function set, the prior knowledge of either q or P-dimension is not required.

Example 1 *Suppose that the desired accuracy and confidence of an approximation task are 0.08 and 0.92 respectively (i.e.* $\epsilon = \delta = 0.08$ *). Also assume that the cardinality of a 0.04-cover of the function set with accordance to* d_P *is less than* 10^{10} *(i.e.* $q < 10^{10}$*) and* $m = 2$*. Using Equation (2.15):* $n \geq 99938$*, meaning that 99938 data points in the empirical risk minimization algorithm would provide the desired accuracy and confidence.*

As mentioned before, the parameter q in Inequality (2.16), plays a vital role in estimation of the overall sample complexity and must be further investigated in the case of modeling with neural networks. Assume that the $\epsilon/2$-cover used is minimal. In that case, q can be replaced by $N(\epsilon/2, \mathcal{F}, d_P)$. As can be seen, $N(\epsilon/2, \mathcal{F}, d_P)$ depends on the prob-

ability measure P, and in case of a fixed-distribution learning procedure, prior knowledge of P is required to bound $N(\epsilon/2, \mathcal{F}, d_P)$.

In generating a set of input-output data to be used for training of a nonlinear model, the input samples must cover all the domain of the input. If it is possible to control the training input data, a set of uniformly distributed input data would be a reasonable approach. The uniform distribution scatters the input samples over the input domain and is often used in practice. Uniformly distributed input data are then applied to the unknown system and the resulting input-output data set is used for training. In the following section, uniformly distributed input data are considered in order to obtain more specific learning bounds.

Next, $N(\epsilon, \mathcal{F}, d_P)$ (as an indication of sample complexity) is expressed in terms of the Lipschitz constant of the function set for uniformly distributed data. Consequently, the sample complexity is related to the Lipschitz constant of the function set, which in turn can be expressed in terms of the function set parameters. The following lemma is based on Example 2 in [6] and Example 6.8 in [2].

Lemma 2.3.2 *(Kolmogorov [6]) Let $\mathcal{F}$ consist of all functions:*

$$f : [\alpha, \beta]^d \to \mathcal{R}$$

that satisfy $f(\mathbf{0}) = \mathbf{0}$, where α, $\beta \in \mathcal{R}$ and d is the dimension of the input variable. Also assume that there exists a finite L such that for all $f \in \mathcal{F}$:

$$|f(\xi) - f(\zeta)| \leq L\|\xi - \zeta\|_\infty \ , \ \forall \xi = (\xi_1, \ldots, \xi_d), \ \zeta = (\zeta_1, \ldots, \zeta_d) \in [\alpha, \beta]^d \qquad (2.23)$$

where:

$$\|\xi - \zeta\|_\infty = \max_{1 \leq i \leq d} |\xi_i - \zeta_i| \ .$$

Now, let P represent a uniform distribution over $[\alpha, \beta]^d$, and define d_P as above. Then:

$$N(\epsilon, \mathcal{F}, d_P) \leq 2^{\lceil \frac{L(\beta-\alpha)}{\epsilon} \rceil^d}. \tag{2.24}$$

Lemma (2.3.2) provides an upper bound of $N(\epsilon, \mathcal{F}, d_P)$ for a function set once an upper bound on the Lipschitz constant of the family is known. In the following sections, the learning properties of some important families of RBFN's, SNN's, Volterra Networks and linear models are studied by finding upper bounds on the Lipschitz constants of those families, and so investigating the sample complexity of the functions sets.

2.4 Learning of RBFN's With Uniformly-Distributed m-Dependent Data

A lemma, inspired by [53] and [40], to bound the Lipschitz constant of a general family of RBFN's is now given.

Lemma 2.4.1 *Suppose a set of RBFN's $\mathcal{F}$ has members expressed as:*

$$f(x) = \sum_{i=1}^{l} a_i \phi_i(r_i)$$

where: l is the number of neurons (basis functions), $a = (a_1, \ldots, a_l)$ forms the weight vector of the network with $|a_i| < M_a < \infty$ for all i, $\phi_i(.)$'s are the bounded differentiable radial basis functions in which $r_i = \|x - c_i\|$, and c_i is the center of the ith basis function. Define:

$$\eta_i = \sup_{x \in [\alpha,\beta]^d} \left| \frac{d\phi_i(r_i)}{dr_i} \right|$$

and form the vector $\eta = (\eta_1, \ldots, \eta_l)$. Further assume that:

$$\sup_{1 \leq i \leq l} \eta_i < \infty$$

and:

$$A = \sup_{\forall a,\eta} \sum_{i=1}^{l} |a_i|\eta_i < \infty \ .$$

Then:

1. *The Lipschitz constant of the function set $\mathcal{F}$ is bounded by:*

$$L \leq Ad \ .$$

2. *For the function set $\mathcal{F}$ defined above:*

$$N(\epsilon, \mathcal{F}, d_P) \leq 2^{\lceil \frac{Ad(\beta-\alpha)}{\epsilon} \rceil^d} \ . \tag{2.25}$$

Proof: Note that:

$$\|x - y\| \leq \sqrt{d}\|x - y\|_\infty$$

where $\|x - y\|$ is the Euclidean distance between x and y. Now, suppose that a finite L_E for $\mathcal{F}$ can be found such that:

$$|f(x) - f(y)| \leq L_E\|x - y\|$$

Setting $L = \sqrt{d}L_E$:

$$\begin{aligned} |f(x) - f(y)| &\leq L_E\|x - y\| \\ &\leq L_E\sqrt{d}\|x - y\|_\infty = L\|x - y\|_\infty \end{aligned} \tag{2.26}$$

This shows that once an upper bound for L_E is available, an upper bound for the Lipschitz constant L can be found. In order to find an upper bound for L_E, note that

for a bounded differentiable function such as the defined neural network f, the value $\sup_{\zeta\in[\alpha,\beta]^d} \|\nabla f(\zeta)\|$ bounds the variation of the function f over all the domain and along all the directions, i.e.:

$$\frac{|f(x)-f(y)|}{\|x-y\|} \leq \sup_{\zeta\in[\alpha,\beta]^d} \|\nabla f(\zeta)\| \tag{2.27}$$

Now assuming that $\zeta = (\zeta_1, \ldots, \zeta_d)$ and $r = \|\zeta - c_i\|$:

$$\begin{aligned}
|\frac{\partial f}{\partial \zeta_k}| &= |\sum_{i=1}^{l} a_i \frac{d\phi_i(r_i)}{dr_i}\frac{dr_i}{d\zeta_k}| \\
&= |\sum_{i=1}^{l} a_i \frac{d\phi_i(r_i)}{dr_i}\left(\frac{\zeta_k - c_{ik}}{\sqrt{\sum_{j=1}^{d}(\zeta_j - c_{ij})^2}}\right)| \\
&\leq \sum_{i=1}^{l} |a_i||\frac{d\phi_i(r_i)}{dr_i}|(1) \\
&\leq \sum_{i=1}^{l} |a_i|\eta_i \\
&\leq A\,.
\end{aligned}$$

Consequently:

$$L_E \leq \sup_{\zeta\in[\alpha,\beta]} \|\nabla f(\zeta)\| \leq A\sqrt{d}\,. \tag{2.28}$$

And finally, (2.26) and (2.28) give:

$$L \leq Ad \tag{2.29}$$

which gives a bound for the Lipschitz constant of the function set and proves the first part of the lemma. Now combining Lemmas 2.3.2 and the bound on the Lipschitz constant, an upper bound for $N(\epsilon, \mathcal{F}, d_P)$ for the function set $\mathcal{F}$ is found. Equation (2.25) is the

straightforward result of substituting (2.29) into (2.24).□□□

The bound on $|d\phi_i(r_i)/dr_i|$ depends on the choice of basis functions. The main result on the learning properties of RBFN's comes as a combination of Theorem 2.3.3, and Lemma 2.4.1 as follows.

Theorem 2.4.1 *Consider a set of radial basis function neural networks $\mathcal{F}$ as defined above. Assume that learning is considered under the assumption of m-dependency of the input training data that are uniformly distributed. Then the empirical risk minimization algorithm, performed over a minimal $\epsilon/2$-cover, results in the PAC learning with m-dependency with the sample complexity bounded by:*

$$n \geq \frac{8(m+1)}{\epsilon^2}\left\{\left[\frac{2Ad(\beta-\alpha)}{\epsilon}\right]^d \ln 2 + \ln\frac{(m+1)}{\delta}\right\} \tag{2.30}$$

or equivalently:

$$\delta \geq 2^{\left[\frac{Ad(\beta-\alpha)}{\epsilon}\right]^d}(m+1)\exp\left[-n\epsilon^2/8(m+1)\right] \tag{2.31}$$

where ϵ and δ are the positive real numbers that determine the accuracy and statistical confidence of the algorithm, respectively.

Proof: Knowing that the function set $\mathcal{F}$ has an ϵ-cover with finite size, according to Theorem 2.3.3 the empirical risk minimization algorithm learns the model. Moreover, direct substitution of q in (2.16) with the upper bound for $N(\epsilon/2, \mathcal{F}, d_P)$ indicated in (2.25) gives (2.30). □□□

Theorem 2.4.1 provides a framework for learning of a neural modeling task, assuming that the data are uniformly distributed. In such a modeling task, the information

regarding the structure of the network (such as the number of neurons and the size of the parameter space) is known, and the objective is to use a set of input-output samples to find the optimal values for the network's parameters. The empirical risk minimization algorithm provides a search method that has an associated sample complexity.

Besides the specific bounds on the sample complexity, Theorem 2.4.1 gives the form of dependency between the accuracy, the statistical confidence, the characteristics of RBFN's and the size of the training data. These dependencies can help define meaningful complexity measures based on learning properties of a neural model, as discussed in Chapter3.

The above results can be further specialized if the basis functions are known. The following sub-sections deal with the learning properties of two popular families of such neural networks. These two families consider Gaussian, and Reciprocal Multiquadratic (RMQ) basis functions.

2.4.1 Gaussian RBFN's

A Gaussian RBFN is defined as:

$$\phi_i(r_i) = \exp(-b_i r_i^2) - \exp(-b_i \|c_i\|^2) \tag{2.32}$$

where $0 < b_i < \infty$ is the width (or scattering) parameter of the ith basis function. The second term normalizes each basis function and guarantees that $f(\mathbf{0}) = 0$. For such a network, the following theorem can be proved.

Theorem 2.4.1.1 *Consider the neural model introduced in Lemma 2.4.1 and suppose that $\phi_i(r_i)$'s are given as (2.32). Forming b as:*

$$b = (b_1,\ b_2,\ \ldots,\ b_l)$$

define:

$$A_{rbfn} = \sup_{a,b} \sum_{i=1}^{l} |a_i| \sqrt{b_i} \ .$$

Then with all assumptions of Theorem 2.4.1, the empirical risk minimization algorithm with m-dependent data performed over a minimal $\epsilon/2$-cover results in the PAC learning with m-dependency and the sample complexity of the algorithm is given by:

$$\begin{aligned} n \ \geq \ & \frac{8(m+1)}{\epsilon^2} \times \\ & \left\{ \left[\frac{2\sqrt{2} A_{rbfn} d(\beta - \alpha)}{\epsilon \sqrt{e}} \right]^d \ln 2 + \ln \frac{(m+1)}{\delta} \right\} \end{aligned} \tag{2.33}$$

or equivalently:

$$\delta \geq 2^{\left[\frac{2\sqrt{2} A_{rbfn} d(\beta - \alpha)}{\epsilon \sqrt{e}} \right]^d} (m+1) \exp\left[-n\epsilon^2 / 8(m+1) \right] \ . \tag{2.34}$$

Proof: For the Gaussian set of basis functions:

$$\frac{d\phi_i(r_i)}{dr_i} = -2 b_i r_i \exp(-b_i r_i^2) \ .$$

It can be seen that the maximum of the absolute value of the above function occurs at $|r_i| = \frac{1}{\sqrt{2b_i}}$, and the maximum itself is: $\sqrt{\frac{2b_i}{e}}$. Therefore:

$$\eta_i = \sqrt{\frac{2b_i}{e}} \ ,$$

which results in (2.33). □□□

2.4.2 RMQ RBFN's

For RMQ basis functions:

$$\phi_i(r_i) = \frac{1}{\sqrt{1+b_i r_i^2}} - \frac{1}{\sqrt{1+b_i\|c_i\|^2}} \tag{2.35}$$

where $0 < b_i < \infty$ is the width (or scattering) parameter of the ith basis function. Similar to Gaussian functions, the second term normalizes each basis function so that $f(\mathbf{0}) = 0$. The following theorem describes the learning properties of RMQ RBFN's.

Theorem 2.4.2.1 *Consider the neural model introduced in Lemma 2.4.1 and suppose that $\phi_i(r_i)$'s are given as (2.35).*

Then with the assumptions of Theorem 2.4.1 and defining b and A_{rbfn} as in Theorem 2.4.1, the empirical risk minimization algorithm with m-dependent data performed over a minimal $\epsilon/2$-cover results in the PAC learning with m-dependency and the sample complexity of the algorithm is given by:

$$\begin{aligned} n \;\geq\; & \frac{8(m+1)}{\epsilon^2} \times \\ & \left\{ \left[\frac{4A_{rbfn}d(\beta-\alpha)}{3\sqrt{3}\,\epsilon} \right]^d \ln 2 + \ln \frac{(m+1)}{\delta} \right\} \end{aligned} \tag{2.36}$$

or equivalently:

$$\delta \geq 2^{\left[\frac{4A_{rbfn}d(\beta-\alpha)}{3\sqrt{3}\,\epsilon}\right]^d} (m+1)\exp\left[-n\epsilon^2/8(m+1)\right] \; . \tag{2.37}$$

Proof: For the introduced set of basis functions:

$$\frac{d\phi_i(r_i)}{dr_i} = -b_i r_i \frac{1}{\sqrt{(1+b_i r_i^2)\,(1+b_i r_i^2)}} .$$

The maximum of the absolute value of this function also occurs at $|r_i| = \frac{1}{\sqrt{2b_i}}$, and the maximum itself is: $\frac{2\sqrt{b_i}}{3\sqrt{3}}$. Therefore:

$$\eta_i = \frac{2\sqrt{b_i}}{3\sqrt{3}} ,$$

which results in (2.36). □□□

In the next section, following a similar approach, the learning properties of SNN's are evaluated.

2.5 Learning of Sigmoid Neural Networks With Uniformly-Distributed m-Dependent Data

In this section, first an upper bound for the Lipschitz constant of a general family of three-layer sigmoid neural networks is calculated. This bound is then used to find an upper bound for the sample complexity of the family.

Lemma 2.5.1 *Consider a set of feedforward neural networks $\mathcal{F}$ whose members are expressed as:*

$$f(x) = \sum_{i=1}^{l} a_i \sigma(b_i x)$$

where: $0 \leq \sigma(.) \leq 1$ is a smooth sigmoid activation function, l indicates the number of neurons, a_i's are the weights of the output layer and the vector b_i defined as: $b_i = (b_{i1}, \ldots, b_{id})$ represents the weights of the first layer. Further assume that:

$$\sum_{i=1}^{l} |a_i| \leq \infty \ , \ \sup_{i,j} |b_{ij}| < \infty$$

and:

$$\eta = \sup_{u \in \mathcal{R}} \left| \frac{d\sigma(u)}{du} \right| < \infty .$$

Define:

$$A_{snn} = \sup \sqrt{\sum_{k=1}^{d} \left[\sum_{i=1}^{l} |a_i||b_{ik}| \right]^2}$$

where the above "sup" is taken over the entire parameter space. Then:

1. *The Lipschitz constant of the function set $\mathcal{F}$ is bounded by:*

$$L \leq \eta A_{snn} \sqrt{d} .$$

2. *For the function set $\mathcal{F}$ defined here:*

$$N(\epsilon, \mathcal{F}, d_P) \leq 2^{\left[\frac{\eta A_{snn} \sqrt{d} (\beta - \alpha)}{\epsilon} \right]^d} . \tag{2.38}$$

Proof:

As in case of RBFN's:

$$|f(x) - f(y)| \leq L_E \|x - y\|$$

Setting $L = \sqrt{d} L_E$:

$$\begin{aligned} |f(x) - f(y)| &\leq L_E \|x - y\| \\ &\leq L_E \sqrt{d} \|x - y\|_\infty = L \|x - y\|_\infty \end{aligned} \tag{2.39}$$

As in RBFN's, it is seen that once an upper bound for L_E is available, an upper bound for the Lipschitz constant L can be found accordingly. Now:

$$\frac{|f(x)-f(y)|}{\|x-y\|} \leq \sup_{\zeta\in[\alpha,\beta]^d} \|\nabla f(\zeta)\| \,. \tag{2.40}$$

Now assuming that $\zeta = (\zeta_1, \ldots, \zeta_d)$ and $u = \sum_{j=1}^{d} b_{ij}\zeta_j$:

$$\begin{aligned} |\frac{\partial f}{\partial \zeta_k}| &= |\sum_{i=1}^{l} a_i \frac{d\sigma(u)}{du}\frac{du}{d\zeta_k}| \\ &= |\sum_{i=1}^{l} a_i \frac{d\sigma(u)}{du} b_{ik}| \\ &\leq \sum_{i=1}^{l} |a_i||\frac{d\sigma(u)}{du}||b_{ik}| \\ &\leq \eta \sum_{i=1}^{l} |a_i||b_{ik}| \end{aligned}$$

Consequently:

$$L_E \leq \sup_{\zeta\in[\alpha,\beta]} \|\nabla f(\zeta)\| \leq \eta A_{snn} \,. \tag{2.41}$$

eqnarray*

And finally, (2.39) and (2.41) give:

$$L \leq \eta A_{snn}\sqrt{d} \tag{2.42}$$

which shows that L is a bound for the Lipschitz constant of the function set and proves the first part of the lemma. As to the second part, Equation (2.38) is a straightforward result of substituting (2.42) into (2.24) □□□

The assumption of $|d\sigma(u)/du|$ being bounded is a requirement of the definition of sigmoid functions.

The main result on learning properties of sigmoid neural networks comes as a combination of Theorem 2.3.3, and Lemma 2.5.1 as follows.

Theorem 2.5.1 *Consider a set of feedforward neural networks $\mathcal{F}$ as defined above. Assume that learning is performed under the assumption of m-dependency of the input training data that are uniformly distributed. Then the empirical risk minimization algorithm performed over a minimal $\epsilon/2$-cover results in the PAC learning with m-dependency and the sample complexity of the algorithm is given by:*

$$n \geq \frac{8(m+1)}{\epsilon^2}\left\{\left[\frac{2\eta A_{snn}\sqrt{d}(\beta-\alpha)}{\epsilon}\right]^d \ln 2 + \ln\frac{(m+1)}{\delta}\right\} \tag{2.43}$$

or equivalently:

$$\delta \geq 2^{\left[\frac{2\eta A_{snn}\sqrt{d}(\beta-\alpha)}{\epsilon}\right]^d}(m+1)\exp\left[-n\epsilon^2/8(m+1)\right] \tag{2.44}$$

where ϵ and δ are the positive real numbers that determine the accuracy and statistical confidence of the algorithm, respectively.

Proof: Knowing that the function set $\mathcal{F}$ has an ϵ-cover with finite size, according to Theorem 2.3.3 the empirical risk minimization algorithm learns the model. Moreover, direct substitution of q in (2.16) with the upper bound for $N(\epsilon/2, \mathcal{F}, d_P)$ in (2.38) gives (2.43). □□□

Theorem 2.5.1 provides a framework for learning of a neural modeling task. As for RBFN's, it is assumed that the structure of the network is available, and the objective is to use a set of input-output samples to find the optimal values for the network's parameters: a_i's and b_{ij}'s (also referred to as "weights"). The empirical risk minimization algorithm determines one of the search methods to find the foresaid parameters.

Besides the specific bounds on the sample complexity, Theorem 2.4.1 gives the form of dependency between the accuracy, the statistical confidence, the characteristics of

sigmoid neural networks and the size of the training data. These dependencies can help define meaningful complexity measures based on learning properties of a neural model, as discussed in Chapter 3.

The above results can be further specialized if the form of the sigmoid function is given. The following sub-sections consider some popular families of feedforward neural networks.

2.5.1 "atan" Sigmoid Functions

First, consider neural networks that include the $\tan^{-1}(.)$ or "atan" sigmoid functions.

Thoerem 2.5.1.1 *Consider the neural model introduced in Lemma 2.5.1. Further assume that:*

$$\sigma(u) = \tfrac{2}{\pi}\tan^{-1}(u)$$

Then with all assumptions of Theorem 2.4.1, the minimum empirical risk algorithm with m-dependent data performed over a minimal $\epsilon/2$-cover results in the PAC learning with m-dependency and the sample complexity of the algorithm is given by:

$$n \geq \frac{8(m+1)}{\epsilon^2}\left\{\left[\frac{4A_{snn}\sqrt{d}(\beta-\alpha)}{\pi\epsilon}\right]^d \ln 2 + \ln\frac{(m+1)}{\delta}\right\} \tag{2.45}$$

or equivalently:

$$\delta \geq 2^{\left[\frac{4A_{snn}\sqrt{d}(\beta-\alpha)}{\pi\epsilon}\right]^d}(m+1)\exp\left[-n\epsilon^2/8(m+1)\right] . \tag{2.46}$$

Proof: For the introduced sigmoid function:

$$\frac{d\sigma(u)}{du} = \frac{2}{\pi}\frac{1}{1+u^2} \leq \frac{2}{\pi}$$

which results in: $\eta = \frac{2}{\pi}$. Now following the results of Theorem 2.5.1 a bound for the sample complexity of the model can be obtained as in (2.45).□□□

2.5.2 Bipolar Exponential Sigmoid Functions

Now the focus is given to neural networks that apply bipolar exponential sigmoid functions of form $\frac{1-e^{-(.)}}{1+e^{-(.)}}$.

Thoerem 2.5.2.1 *Consider the neural model introduced in Lemma 2.5.1. Further assume that:*

$$\sigma(u) = \frac{1-e^{-u}}{1+e^{-u}} .$$

Then with all assumptions of Theorem 2.4.1, the empirical risk minimization algorithm with m-dependent data performed over a minimal $\epsilon/2$-cover results in the PAC learning with m-dependency and the sample complexity of the algorithm is given by:

$$n \geq \frac{8(m+1)}{\epsilon^2}\left\{\left[\frac{A_{snn}\sqrt{d}(\beta-\alpha)}{\epsilon}\right]^d \ln 2 + \ln\frac{(m+1)}{\delta}\right\} \tag{2.47}$$

or equivalently:

$$\delta \geq 2^{\left[\frac{A_{snn}\sqrt{d}(\beta-\alpha)}{\epsilon}\right]^d}(m+1)\exp\left[-n\epsilon^2/8(m+1)\right] . \tag{2.48}$$

Proof: For the introduced sigmoid function:

$$\frac{d\sigma(u)}{du} = 2\frac{e^{-u}}{(1+e^{-u})^2} \leq \frac{1}{2}$$

which results in: $\eta = 1/2$. Now following the results of Theorem 2.5.1 a bound can be obtained for the sample complexity of the model as in (2.47).□□□

2.6 Learning of Volterra Neural Networks With Uniformly-Distributed m-Dependent Data

In this section, the learning properties of Volterra networks are discussed. Despite the similarities of Volterra nets to both RBFN's and Sigmoid neural networks, due to the polynomial structure of Volterra networks, their learning characteristics should be discussed separately. As for the previous structures, first an upper bound for the Lipschitz constant of a family of Volterra neural networks is calculated, and then this bound is used to find an upper bound for the sample complexity of the family. Although the basis functions in Volterra networks are defined as hyper-polynomial of degree less than or equal to an arbitrary p, in this thesis, only the networks up to degree $p = 3$ are considered. The calculation of the learning bounds for the networks of an arbitrary degree can be obtained in a similar manner with the cost of dealing with more complex notation and longer equations.

Lemma 2.6.1 *Suppose* $x = (\zeta_1, \ldots, \zeta_d)$. *Define the members of a set of Volterra neural networks* $\mathcal{F}$ *as:*

$$f(x) = \sum_{i=1}^{d} a_i\zeta_i + \sum_{i=1}^{d}\sum_{j=i}^{d} b_{ij}\zeta_i\zeta_j + \sum_{i=1}^{d}\sum_{j=i}^{d}\sum_{l=j}^{d} c_{ijl}\zeta_i\zeta_j\zeta_l$$

where: a_i*'s,* b_{ij}*'s, and* c_{ijl}*'s are the weights of the network. Further assume that:*

$$\sup_i |a_i| \leq \infty \ , \ \sup_{i,j\geq i} |b_{ij}| < \infty \ , \ \sup_{i,j\geq i,\, l\geq j} |c_{ijl}| < \infty$$

Assuming $M = \max[|\alpha|, |\beta|]$, *define:*

$$\begin{aligned} B_k &= |a_k| + M \sum_{i=1\ ,\ i\neq k}^{d} |b_{ik}| + 2M|b_{kk}| \\ &+ M^2 \sum_{i=1}^{d} \sum_{j=i\ ,\ j\neq k}^{d} |c_{ijk}| + 2M^2 \sum_{i=1\ ,\ i\neq k}^{d} |c_{ikk}| + 3M^2|c_{kkk}| \end{aligned}$$

and:

$$A_{vol} = \sup \left[\sqrt{\sum_{k=1}^{d} (B_k)^2} \right] \tag{2.49}$$

where the above "sup" is taken over the entire parameter space. Then:

1. *The Lipschitz constant of the function set $\mathcal{F}$ is bounded by:*

$$L \leq A_{vol} . \sqrt{d}$$

2. *For the function set $\mathcal{F}$ defined in this lemma:*

$$N(\epsilon, \mathcal{F}, d_P) \leq 2^{\left\lceil \frac{A_{vol}\sqrt{d}(\beta-\alpha)}{\epsilon} \right\rceil^d} .$$

(2.50)

Proof:

As in the case of RBFN's and sigmoid neural networks, it can be seen that once an upper bound for L_E is available, an upper bound for the Lipschitz constant L can be found accordingly. Also:

$$\frac{|f(x)-f(y)|}{\|x-y\|} \leq \sup_{\zeta\in[\alpha,\beta]^d} \|\nabla f(\zeta)\| \,. \tag{2.51}$$

Now:

$$\begin{aligned}
|\frac{\partial f}{\partial \zeta_k}| &= |a_k + \sum_{i=1\,,\,i\neq k}^{d} b_{ik}\zeta_i + 2b_{kk}\zeta_k + \sum_{i=1}^{d}\sum_{j=i\,,\,j\neq k}^{d} c_{ijk}\zeta_i\zeta_j + \sum_{i=1\,,\,i\neq k}^{d} 2c_{ikk}\zeta_i\zeta_k + 3c_{kkk}\zeta_k^2| \\
&\leq |a_k| + |\sum_{i=1\,,\,i\neq k}^{d} b_{ik}\zeta_i| + |2b_{kk}\zeta_k| + |\sum_{i=1}^{d}\sum_{j=i\,,\,j\neq k}^{d} c_{ijk}\zeta_i\zeta_j| + |\sum_{i=1\,,\,i\neq k}^{d} 2c_{ikk}\zeta_i\zeta_k| + |3c_{kkk}\zeta_k^2| \\
&\leq |a_k| + \sum_{i=1\,,\,i\neq k}^{d} |b_{ik}||\zeta_i| + 2|b_{kk}||\zeta_k| + \sum_{i=1}^{d}\sum_{j=i\,,\,j\neq k}^{d} |c_{ijk}||\zeta_i||\zeta_j| \\
&+ \sum_{i=1\,,\,i\neq k}^{d} 2|c_{ikk}||\zeta_i||\zeta_k| + 3|c_{kkk}||\zeta_k^2| \\
&\leq |a_k| + M\sum_{i=1\,,\,i\neq k}^{d} |b_{ik}| + 2M|b_{kk}| + M^2\sum_{i=1}^{d}\sum_{j=i\,,\,j\neq k}^{d} |c_{ijk}| + 2M^2\sum_{i=1\,,\,i\neq k}^{d} |c_{ikk}| + 3M^2|c_{kkk}| \\
&= B_k \,.
\end{aligned}$$

This means that:

$$\begin{aligned}
\sqrt{\sum_{k=1}^{d} |\frac{\partial f}{\partial \zeta_k}|^2} &\leq \sqrt{\sum_{k=1}^{d} (B_k)^2} \\
&\leq A_{vol} \,.
\end{aligned}$$

Consequently:

$$\begin{aligned}
L_E &\leq \sup_{\zeta\in[\alpha,\beta]} \|\nabla f(\zeta)\| \\
&\leq A_{vol} \,.
\end{aligned}$$

And finally, with a discussion similar to the ones given for RBFN's:

$$L \leq A_{vol}\sqrt{d}\,. \tag{2.52}$$

which shows that L is a bound for the Lipschitz constant of the function set and concludes the proof of the first part. As to the second part, Equation (2.50) is a straightforward result of substituting (2.52) into (2.24). □□□

The main result on learning properties of Volterra neural networks comes as a combination of Theorem 2.3.3, and Lemma 2.6.1 as follows.

Theorem 2.6.1 *Consider a set of feedforward neural networks $\mathcal{F}$ as defined above. Assume that learning is performed under the assumption of m-dependency of the input training data that are uniformly distributed. Define A_{vol} as above. Then the empirical risk minimization algorithm performed over a minimal $\epsilon/2$-cover results in the PAC learning with m-dependency and the sample complexity of the algorithm is given by:*

$$n \geq \frac{8(m+1)}{\epsilon^2}\left\{\left[\frac{2A_{vol}\sqrt{d}(\beta-\alpha)}{\epsilon}\right]^d \ln 2 + \ln\frac{(m+1)}{\delta}\right\} \tag{2.53}$$

or equivalently:

$$\delta \geq 2^{\left[\frac{2A_{vol}\sqrt{d}(\beta-\alpha)}{\epsilon}\right]^d}(m+1)\exp\left[-n\epsilon^2/8(m+1)\right] \tag{2.54}$$

where ϵ and δ are the positive real numbers that determine the accuracy and statistical confidence of the algorithm, respectively.

Proof: Knowing that the function set $\mathcal{F}$ has an ϵ-cover with finite size, according to Theorem 2.3.3 the empirical risk minimization algorithm learns the model. Moreover, direct substitution of q in (2.16) with the upper bound for $N(\epsilon/2, \mathcal{F}, d_P)$ in (2.50) gives (2.53). □□□

In some system identification and control literature, the use of Volterra networks to model an unknown systems is referred to as nonlinear system identification. Historically, Volterra polynomials were used in system identification before other types of nonlinear models were widely used. In this thesis, the name "nonlinear system identification", is used in its wide sense.

2.7 Learning of Linear Models With Uniformly-Distributed m-Dependent Data

Due to the importance of linear FIR models, and to compare the learning results of different neural structures with those of a simple linear model, linear models are now considered. Consider a family of linear functions. Suppose $x = (\zeta_1, \ldots, \zeta_d)$. Define the members of a set of linear models $\mathcal{F}$ as:

$$f(x) = \sum_{i=1}^{d} a_i \zeta_i \ .$$

The linear model is a special case of Volterra network where only the a_i's are assumed to be non-zero. As a results, the bounds on the learning properties of a linear model can be easily obtained as follows.

Theorem 2.7.1 *Consider a set of linear models $\mathcal{F}$ as defined above. Assume that learning is performed under the assumption of m-dependency of the input training data that are uniformly distributed. Define A_{lin} as:*

$$A_{lin} = \sup \left[\sqrt{\sum_{k=1}^{d} |a_k|^2} \right] \tag{2.55}$$

where the above "sup" is taken over the entire parameter space. Then the empirical risk minimization algorithm performed over a minimal $\epsilon/2$-cover results in the PAC learning with m-dependency and the sample complexity of the algorithm is given by:

$$n \geq \frac{8(m+1)}{\epsilon^2}\left\{\left[\frac{2A_{lin}\sqrt{d}(\beta-\alpha)}{\epsilon}\right]^d \ln 2 + \ln\frac{(m+1)}{\delta}\right\} \tag{2.56}$$

or equivalently:

$$\delta \geq 2^{\left[\frac{2A_{lin}\sqrt{d}(\beta-\alpha)}{\epsilon}\right]^d}(m+1)\exp\left[-n\epsilon^2/8(m+1)\right] \tag{2.57}$$

where ϵ and δ are the positive real numbers that determine the accuracy and statistical confidence of the algorithm, respectively.

Proof: As mentioned above, linear models are special cases of Volterra networks, and as a result, the proof for this theorem parallels that of Volterra networks, and will not be given here. □□□

2.8 Using The Learning Results in a Typical Modeling Procedure

In modeling of unknown systems, the structure as well as the parameter space of the neural model is often fixed. The characteristics of the parameter space, such as the number of neurons and the maximum size of the network's parameters, are fixed before training. The desired values of accuracy and statistical confidence are also pre-specified. The inequalities of the previous sections then determine a sufficient size of training data to guarantee the pre-specified levels of accuracy and confidence.

The following examples further clarify the approach.

Example 2.8.1 *Suppose that the i.i.d. sequence $(u_i)_{i=1}^{\infty}$ has been generated according to a uniform distribution over the interval of $[0,1]$. Also, assume that the stochastic process $y(t)$, $t = t_o, t_o+1, \ldots, \infty$, depends on the random variables $(u_i)_{i=t-t_o}^{t}$ through a function f, i.e.:*

$$y(t) = f(u_{t-t_o+1}, u_{t-t_o+2}, \ldots, u_t) \ .$$

Now define the random variable x_t *as:*

$$x_t = [u_{t-t_o+1} \ u_{t-t_o+2} \ \cdots \ u_t]^T \in X$$

where X *is* $[0,1]^{t_o}$. *As a result, the* x_t*'s are* t_o*-dimensional* t_o*-dependent random vectors and:*

$$y(t) = f(x_t) \ .$$

Now consider an NFIR modeling task for which f is assumed to be an unknown member of a family of atan SNN's. With the above formulation and using Theorem 2.5.1.1, the empirical risk minimization algorithm applied to this problem results in PAC learning with t_o*-dependent input data where:* $t_o = m$ *and* $m = d$.

In order to have a better evaluation on the sample of the algorithm, let: $t_o = 2$, $\epsilon = \delta = 0.08$, $l = 10$ *and for all i:* $|a_i| \leq 0.1$, $|b_i| \leq 0.1$ *which results to:* $A_{snn} \leq 0.0447$. *For the above choices, according to Inequality (2.45), a bound for the sample complexity of the algorithm is:* $n \geq 16225$. *This means that if the training set includes more than 16225 sample points, the algorithm guarantees the prespecified values of accuracy and confidence.*

The next example describes a similar identification task using a family of Gaussian RBFN's. In order to compare the bounds on sample complexity of SNN's with those of RBFN's, the comparison must be made for networks with similar computational capabilities. In other words, the number of hidden neurons as well as the size of parameters in both networks must be set to give similar modeling and approximation accuracies. In literature (see for example [10]), based on the results of applying both SNN's and RBFN's to the same applications, experimental rules have been presented to relate the structure of SNN's and RBFN's that often result in similar performances. In other words, these rules relate the number of hidden neurons of SNN's and RBFN's such that equivalent classification and modeling performances are achieved. According to the table given in [10], assuming the same size of $|a_i|$'s and $|b_i|$'s , RBFN's normally often use almost 4

times as many neurons as SNN's for the same level of accuracy. This ratio is attributed to the fact that RBFN's are local approximators, needing more neurons than in the case of non-locally distributed approximators such as SNN's. However, since these are experimental rules, and in order to have a more objective comparison, here it is assumed that all the settings (the size of the parameter space and the number of hidden neurons) are the same, as indicated in the following examples.

Example 2.8.2 *Consider the identification task described in the previous example and assume that instead of a set of sigmoid functions, a family of Gaussian RBFN's are used. Further assume that:* $t_o = 2$, $\epsilon = \delta = 0.08$, $l = 10$ *and for all i:* $|a_i| \leq 0.1$, $|b_i| \leq 0.1$ *which results in:* $A_{rbfn} \leq 0.3162$. *For the above choice of values, using Theorem 2.4.1.1, the modeling task is learnable and the sample complexity of the algorithm is bounded by:* $n \geq 491710$.

As can be seen, according to our bounds, even for the same number of neurons, Gaussian RBFN's require more training data points than atan SNN's. The next example deals with the same procedure when RMQ basis functions are used.

Example 2.8.3 *With all assumptions of Example 2.8.2 except using RMQ basis function for the network instead of Gaussian ones, using Theorem 2.4.2.1 one can show that the modeling task is learnable. Furthermore, in the case of the given numerical values in Example (2.8.2), an upper bound on the sample complexity of the algorithm is given by:* $n \geq 109860$.

The results obtained above will be discussed in details in Section 2.10; however a simple comparison of the above bounds, indicates that based on these conservative bounds, SNN's exhibit more desirable learning properties and require fewer examples to learn a typical example. Comparison of the above examples also shows that based on the results obtained in this chapter, from the standpoint of sample complexity, Gaussian RBFN's are the worst type of neural networks among the networks evaluated here.

Network	m=2	m=5	m=10	m=20
Atan SNN	16225	37648	77357	164655
Bipolar Exponential SNN	15216	35630	73657	157598
Gaussian RBFN	109865	988616	420693	820112
RMQ SNN	491710	224923	1820849	3493081

Table 2.1: Bounds on sample complexity of different families of neural networks for different filter lengths (m).

It is insightful to observe the way sample complexity grows with the filter length "m". In order to see this, assume that the model parameters are chosen as follows: $\epsilon = \delta = 0.08$, $l = 10$ and for all i: $|a_i| \leq 0.1$, $|b_i| \leq 0.1$. Then, for different values for m and using the bounds obtained in this chapter, Table (2.1) for sample complexity bounds for different families of neural networks can be formed.

As can be seen from the table, from the point of view of sample complexity, bipolar exponential SNN's and Gaussian RBFN's are the most and least desirable forms of networks, respectively. The results of this table are further discussed in Section 2.10 .

Another practically important issue in neural modeling is now addressed. The conventional form of PAC learning assumes that the data are generated by a non-noisy system. In many physical systems, either the data generating unit is a stochastic system (with no notion of any particular function set), or the output of the function set is accessible only after it is corrupted by additive noise. In such general cases, an extension of PAC learning known as "Model-Free PAC Learning" is required.

2.9 Model-Free PAC Learning

The learning schemes described in the previous sections assume that $h \in \mathcal{F}$. This may not be true in many real applications, as either $\mathcal{F}$ might be an unknown function set, or the data might be noisy. In such cases, another learning scheme called "model-free learning" is a more realistic formulation. In this scheme, the approximator function comes from a known function set $\mathcal{H}$, but no assumption is made about the function set of which the unknown function f is a member. Then, instead of trying to estimate f directly,

the algorithm estimates the function in $\mathcal{H}$ that best approximates f. Among different formulations of "model-free learning", the one given by Vidyasagar in [2] seems to be the most accurately formulated one. The formulation introduced here parallels Vidyasagar's model-free learning, extending it to a more general type of learning (i.e. learning with dependent data). The exact definition of model-free learning, extended to cover learning with m-dependent data is given below:

Definition 2.9.1 *Introduce the following notation:*

- *Sets X, Y, U and a σ-algebra $\overline{\mathcal{S}}$ on $X \times Y$.*
- *A fixed probability measure P on X.*
- *A family $\overline{\mathcal{P}}$ of probability measures on $X \times Y$, where all $\bar{P} \in \overline{\mathcal{P}}$ have the same marginal probability P on X.*
- *A family $\mathcal{H}$ of measurable functions mapping X into U, called the "hypothesis class."*
- *A function $\boldsymbol{l} : Y \times U \to [0,1]$, called the "loss function." Based on the loss function $\boldsymbol{l}$, for each $h \in \mathcal{H}$ define an associated function $\boldsymbol{l}_h : Y \times U \to [0,1]$ by:*

$$\boldsymbol{l}_h(x,y) = \boldsymbol{l}[y, h(x)], \forall x, y,$$

and a family of functions $\mathcal{L}_{\mathcal{H}} = \{\boldsymbol{l}_h : h \in \mathcal{H}\}$

With each hypothesis function h and each probability measure $\bar{P}$, associate a cost function:

$$J(h, \bar{P}) = \int_{X \times Y} \boldsymbol{l}[y, h(x)] \bar{P}(dx, dy) . \tag{2.58}$$

Also, define the "minimum achievable cost function" as:

$$J^*(\bar{P}) = \inf_{h \in \mathcal{H}} \int_{X \times Y} \boldsymbol{l}[y, h(x)] \bar{P}(dx, dy) . \tag{2.59}$$

Samples $z_n = \{(x_i, y_i)\}_{i=1}^n$ *are generated where* $(x_i)_{i=1}^n$ *is a sequence of m-dependent random variables distributed according to* P. *Then, an algorithm is considered as an index family of mappings* A_n, *where* $n \geq 1$, $A_n : (X \times Y)^n \to \mathcal{H}$ *and* $h_n = A_n(z_n)$.

Then the algorithm is said to be "model-free probably approximately correct to accuracy ϵ *with m-dependent data" if for any choice of* ϵ *and* δ, *there exists "n" such that:*

$$\Pr\left\{J(h_n, \bar{P}) > J^*(\bar{P}) + \epsilon\right\} < \delta \,. \tag{2.60}$$

If when $n \to \infty$ *the above probability approaches zero, the algorithm is said to be "model-free probably approximately correct with m-dependent data".*

Next, define "the model-free empirical risk minimization algorithm" which is a natural extension of the empirical risk minimization to the model-free learning scheme.

Definition 2.9.2 *Suppose U defined in Definition 2.9.1 is a subset of* $\mathcal{R}$. *Assume that the loss function defined in Definition 2.9.1 satisfies a uniform Lipschitz condition, i.e. there exists a finite constant* μ *such that:*

$$|\boldsymbol{l}(y, u_1) - \boldsymbol{l}(y, u_2)| \leq \mu |u_1 - u_2|, \ \forall u_1, u_2 \in \mathcal{R}, \forall y \in Y \,. \tag{2.61}$$

Also, assume that an ϵ_0/μ*-cover* $\{g_1, \ldots, g_q\}$ *of* $\mathcal{H}$ *is available where* $\epsilon_0 < \epsilon$. *Next, calculate the empirical cost functions for all members of the* ϵ_0/μ*-cover, i.e. find* $\hat{J}_i$*'s as:*

$$\hat{J}_i = \frac{1}{n} \sum_{j=1}^{n} \boldsymbol{l}[y_j, g_i(x_j)], \ 1 \leq i \leq q \,. \tag{2.62}$$

Then, the output of the algorithm is g_{io} in the ϵ_0/μ-cover with minimum empirical cost function, i.e. choose g_{io} such that:

$$\hat{J}_{io} = \min_{1 \le i \le q} \hat{J}_i \,. \tag{2.63}$$

Next, a theorem is proved that deals with learnability as well as sample complexity of the model-free empirical risk minimization algorithm.

Theorem 2.9.1 *Suppose:*

1. *$\overline{\mathcal{P}}$ is a family of probabilities with the property that every $\bar{P} \in \overline{\mathcal{P}}$ has the same marginal probability P on X.*

2. *The hypothesis class $\mathcal{H}$ has the property that:*

$$N(\epsilon, \mathcal{H}, d_P) < \infty, \forall \epsilon > 0$$

3. *A loss function $\boldsymbol{l}$ that satisfies the uniform Lipschitz condition of (2.61).*

Then $\left(\mathcal{H}, \overline{\mathcal{P}}, \boldsymbol{l}\right)$ is model-free PAC learnable. In particular, given any $\epsilon > 0$, choose $\{g_1, \ldots, g_q\}$ to be an ϵ_0-cover of $\mathcal{H}$ with respect to d_P for some $\epsilon_0 < \epsilon$. Then the model-free empirical risk minimization algorithm applied to $\{g_1, \ldots g_q\}$ is PAC to accuracy ϵ, and the sample complexity of the algorithm is bounded by:

$$n \ge \frac{8}{\epsilon^2} \ln \frac{q}{\delta} \,. \tag{2.64}$$

Proof: First prove that for all $\bar{P} \in \overline{\mathcal{P}}$ and every $h_1, h_2 \in \mathcal{H}$:

$$|J(h_1, \bar{P}) - J(h_2, \bar{P})| \leq \mu d_P(h_1, h_2) \; . \tag{2.65}$$

This can be proved as follows:

$$\begin{aligned} \left|J(h_1, \bar{P}) - J(h_2, \bar{P})\right| &= \left|\int_{X\times Y} \boldsymbol{l}(y, h_1(x)) - \boldsymbol{l}(y, h_2(x))\bar{P}(dx, dy)\right| \\ &\leq \int_{X\times Y} |\boldsymbol{l}(y, h_1(x)) - \boldsymbol{l}(y, h_2(x))|\, \bar{P}(dx, dy) \\ &\leq \mu \int_{X\times Y} |h_1(x) - h_2(x)|\bar{P}(dx, dy) \\ &= \mu \int_X |h_1(x) - h_2(x)|P(dx) \\ &= \mu d_P(h_1, h_2) \; . \end{aligned}$$

Now let $\bar{P} \in \overline{\mathcal{P}}$ be arbitrary, and select $h = h(\epsilon, \bar{P})$ such that:

$$J(h, \bar{P}) \leq J^*(\bar{P}) + \frac{\epsilon - \epsilon_0}{2} \; . \tag{2.66}$$

From the definition of $J^*(\bar{P})$, one can assure that such an h does exist. Since $\{g_1, \ldots, g_q\}$ is an $\epsilon_0/2\mu$-cover of $\mathcal{H}$, h must be within a distance $\epsilon_0/2\mu$ from a (possibly unknown) function in the cover set. Assume without loss of generality that the cover set is renumbered such that $d_P(h, g_q) \leq \epsilon_0/2\mu$, which in turn implies that:

$$J(g_q, \bar{P}) \leq J(h, \bar{P}) + \epsilon_0/2 \leq J^*(\bar{P}) + \epsilon/2 \tag{2.67}$$

Assume that the renumbering is such that:

$$J(g_i, \bar{P}) > J^*(\bar{P}) + \epsilon, \; 1 \leq i \leq k, \text{ and} \tag{2.68}$$

$$J(g_i, \bar{P}) \leq J^*(\bar{P}) + \epsilon, \; k+1 \leq i \leq q \tag{2.69}$$

Note that $k \leq q-1$. Now suppose a multi-sample set $z_n = \{(x_i, y_i)\}_{i=1}^n$, where $(x_i)_{i=1}^n$ is a sequence of m-dependent r.v.s identically distributed according to probability P, is available. Based on this data set, the model-free empirical risk minimization algorithm introduces $h_n \in \{g_1, \ldots, g_q\}$ as the output of the model. Notice that the error involved in the algorithm would be introducing one of the g_i's, where $i = 1, \ldots, k$, as h_n. Let the event E be:

$$E = \{\hat{J}(g_q, z_n) \leq J^*(\bar{P}) + 3\epsilon/4, \text{ and } \hat{J}(g_i, z_n) > J^*(\bar{P}) + 3\epsilon/4, \ i \leq k\} \tag{2.70}$$

From the definition of E, it can be seen that: $E \subseteq \{J(h_n, \bar{P}) > J^*(\bar{P}) + \epsilon\}$. Now:

$$\Pr\{E^c\} \leq \Pr\{\hat{J}(g_q, z_n) > J^*(\bar{P}) + 3\epsilon/4\} + \sum_{i=1}^{k} \Pr\{\hat{J}(g_i, z_n) \leq J^*(\bar{P}) + 3\epsilon/4\} \tag{2.71}$$

Next, considering (2.67):

$$\Pr\{\hat{J}(g_q, z_n) > J^*(\bar{P}) + 3\epsilon/4\} \leq \Pr\{\hat{J}(g_q, z_n) - J(g_q, \bar{P}) > \epsilon/4\} \tag{2.72}$$

Now, notice that $\hat{J}(g_q, z_n)$ and $J(g_q, \bar{P})$ are the empirical and true means of the function l_{g_q}. Using Lemma 2.3.1 on the difference between the empirical and true means of a function, the above probability can be bounded as:

$$\Pr\{\hat{J}(g_q, z_n) > J^*(\bar{P}) + 3\epsilon/4\} \leq (m+1) \exp\left[-\frac{n\epsilon^2}{8(m+1)}\right] . \tag{2.73}$$

In the same manner, other probabilities in (2.71), i.e. the ones for $1 \leq k$, can be also bounded resulting in the same bound as above. Now, since $k \leq q-1$, the maximum error involved in the overall procedure would be: $(m+1) \exp\left[-\frac{n\epsilon^2}{8(m+1)}\right]$. In other words:

$$\Pr\{J(h_n, \bar{P}) < J^*(\bar{P}) + \epsilon\} \leq (m+1) \exp\left[-\frac{n\epsilon^2}{8(m+1)}\right] . \tag{2.74}$$

This means that for any arbitrary choices of ϵ and n, model-free PAC learning with $\delta \geq q(m+1)\exp\left[-\frac{n\epsilon^2}{8(m+1)}\right]$ is achievable. Equivalently, when ϵ and δ are fixed:

$$n \geq \frac{8(m+1)}{\epsilon^2}\ln\frac{q(m+1)}{\delta} \tag{2.75}$$

which concludes the proof. □□□

As mentioned in the proof, q is the cardinality of a minimal ϵ_0/μ-cover of the function set. This means that the same procedure of calculating an upper bound of this value for different function sets as in the case of the standard PAC learning paradigm yields similar results for the model-free PAC learning. In other words, for any of the previous results on different families of neural networks, the sample complexity of the model-free learning can be easily extended by replacing ϵ with ϵ_0/μ, where ϵ_0 can be chosen arbitrarily close to ϵ. Thus it is possible to avoid repeating all the proofs for the model-free learning case and simply use the inequalities of the standard PAC learning, as mentioned above.

The model-free scheme provides a learning framework that can handle modeling of non-functional stochastic sources of data assuming different types of loss functions. It also shows that when using a loss function with $\mu \approx 1$, the computational complexity of a standard PAC learning scheme is very close of that of a model-free one. This explains why even though the assumptions made in a standard PAC learning are more restrictive and less realistic, the learning properties of many algorithms are normally assessed within the standard PAC, and the results are easily extended to the model-free framework.

2.10 Discussion

Considering the results of the previous sections as well as the Examples given above, the followings remarks can be made.

1. With some modifications, all the results can be extended to PAC learning with other metric measurements of the distance f and h, such as the popular second order norm. The "model-free learning" paradigm allows the metric measure to belong to a family of measures rather than just being a particular form of measure such as

$d_P(f, h)$.

2. The bounds on sample complexity are sufficient bounds and not necessary ones, i.e. learning might be achievable with fewer number of training points. Therefore, the performance of different models cannot be confidently compared with each other merely based on the given bounds for the sample complexities. Nevertheless, the presented bounds provide some meaningful functionalities and dependencies between the parameters of the models and the overall learning performance of the models. Also, the bounds on the sample complexity are all based on the extension of Hoeffding's inequality in the case of i.i.d. data to m-dependent variables. Since Hoeffding's inequality is known to give one of the tightest available bounds on summation of random variables, one can expect the presented bounds to be reasonably close to the best achievable bounds with the available inequalities. However, numerical examples (such as the ones given in this chapter) reveal that the bounds are indeed conservative.

3. The empirical risk minimization algorithm not only provides a framework for comparing the sample complexity of different function approximation procedures, but also gives an insight to the behavior of "more practical" types of algorithms. As mentioned before, all the results presented above are for a standard form of the empirical risk minimization algorithm, where the availability of a minimal $\epsilon/2$-cover is pre-assumed. However, the results obtained for the empirical risk minimization algorithm can be useful to describe the properties of other learning algorithms which do not require an $\epsilon/2$-cover. Such algorithms perform empirical minimization over the entire family of $\mathcal{F}$ rather than the finite set of minimal $\epsilon/2$-cover. This can be interpreted as performing the standard empirical risk minimization where $\epsilon \to 0$. Notice that all the results given above tell us that when $\epsilon \to 0$, $N(\epsilon/2, \mathcal{F}, d_P) \to \infty$, and that such an algorithm will require an infinite number of training examples. However, the relative rates at which the required sample sizes grow for different models are more important than the exact value of the bound for each model. As an example, suppose that the sample complexity of the neural model A, with a

certain number of neurons and a certain size of parameters, grows faster than a model B with some other structure and parameter space. Then, it can be expected that the neural model B requires less effort and fewer training data not only for the empirical risk minimization algorithm but also for many algorithms that perform global optimization. This idea is further described in the next chapter.

4. It is well-known that the models with larger complexity call for greater sample complexities. The idea behind this belief comes from the fact that the computation time (and as a result the computational complexity) of an algorithm increases as the training algorithm processes more training data. Therefore, the sample complexity can be considered as an indication of the overall computational complexity of the model used for approximation. Considering the sample complexity as an indication of model complexity is the fundamental idea of the structural risk minimization method (as described in the next chapter).

5. The bounds given for all models indicate that the sample complexity depends not only on the number of neurons and the dimension of the input, but also on the size of the parameter space. This means that even without adding new neurons to the model, and only by increasing the size of the parameter space, one can achieve a more complex neural model. This is the main point a group of researchers including Bartlett [41] has been making since the early 1990's. They believe that the common trend of adding new neurons to enhance the computational capabilities of neural networks without paying attention to the size of the parameter space may not be the best approach in neural modeling. They recommend that the computational performance of a neural network can be enhanced more systematically (from the point of view of learning theory) by keeping the number of the neurons the same and allowing the parameter space to grow larger. This will be further discussed in the next chapter.

6. The above Examples give a frameworks to assess learning properties for the identifi-

cation of a dynamic system using NFIR models. It is known that in some practical applications, ARX dynamic systems can be effectively approximated with NFIR models [16]. However, some properties of ARX systems (including stability) can not be appropriately addressed with the NFIR approximators.

7. In the bounds given by Theorems (2.4.1.1) and (2.4.2.1) for the sample complexity of RBFN's the effect of centers of the basis functions on the overall sample complexity cannot be observed explicitly. The results give the same bound for the sample complexity of two RBFN's that use the same parameters but have different centres. This can be regarded as one of the reasons for our bounds being very conservative. A more careful investigation of the Lipschitz constants of such families with more assumptions on the input space as well as the centers might result in a set of bounds in which the centers play an explicit role. In order to obtain results that incorporate the centers into the bounds, assumptions on the way the centers are distributed over the input space would be made. By assuming a grid-type arrangement of the centers, the above bounds can be further improved. However, such assumptions would make the results less general as the resulting bounds would be applicable only if the same grid arrangement is used to form the centers.

8. The bounds for sample complexity of SNN's and RBFN's suggest RBFN's require more training data than SNN's. This is in contradiction with the results [46] that prefer GSLN's over SNN's. The main reason for this contradiction is that the bounds used in [46] for comparison do not consider the size of the parameter space and are usually based on highly conservative assumptions. Most of the bounds used in [46] are the ones introduced for binary neural networks in which the values of the weights are disregarded. They also do not consider the fact the RBFN's normally (but not always) require more neurons (10 to 4 times) than SNN's to perform the same task of identification or classification with a similar level of accuracy. The two structures are compared only when the number of hidden neurons for both networks is assumed to be the same. As shown in this chapter, even with the same number

of hidden neurons, based on our bounds, SNN's require fewer training data points. This observation is more compatible with the fact that RBFN's are more "local" than SNN's, and as a result, they might need more data points to reliably cover the entire input space.

The use of bounds rather than actual values of sample complexity may not be a reliable approach for such a comparison. Note also that the types of approximations used in obtaining the above bounds have been slightly different from one neural structure to another. However, the comparison uses the best available results. This in turn indicates that if less conservative bounds are obtained, then the results of the comparison made in this thesis become more reliable.

9. From Table 2.1, it can be seen that atan SNN's and bipolar exponential SNN's have close rates of growth in term of m. Also, these two structures are far superior to the RBFN's, as far as sample complexity is concerned. Moreover, even with $m = 20$, the sample complexities of these two structures are within the typical range of databases of many real applications such as biomedical and marketing systems.

   In order to sort the most popular forms of neural networks based on their learning properties, the diagram of Figure (2.1) can be used. This diagram shows that among the four structures compared in the diagram, bipolar exponential SNN's and Gaussian RBFN's are the most and the least preferred types of neural networks.

   Also, comparing RMQ and Gaussian RBFN's, based on the bounds of Table 2.1 as well as the theoretical bounds given above, it can be observed that RMQ-RBFN's exhibit a more desirable learning behaviour. This result is likely to be reliable since the process of obtaining the bounds is almost the same for both structures. The only spot where the two bounds become different is at the final stage when the η_i's are to be bounded, and even at that point, the way the two values are bounded is reasonably optimal for both structures.

   The superiority of RMQ-RBFN's over the Gaussian RBFN's, and the fact they exhibit a better performance as far as the overfitting problem is concerned is reported in many practical applications, including [38]. Again, this superiority can be attributed to the fact that RMQ-RBFN's (for the same range of parameters b_i's) are

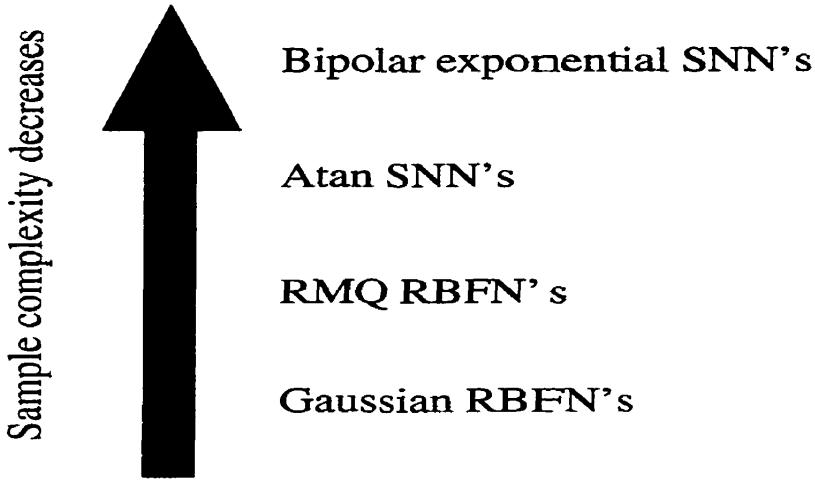

Figure 2.1: Sorting neural models based on their sample complexity

less local than the Gaussian ones. If two networks are trained with the same set of scattered data points, and tested against a set of data points that fall in between the scattered training points (and not particularly close to any of them), then the less local structures (i.e. RMQ-RBFN's) create a better approximation.

The above discussion encourages the use of RMQ-RBFN's. Since RMQ-RBFN's have become the most popular RBFN's used in practical applications (see [38] for example), in the next section this family is chosen among the different families of RBFN's and used for the simulations.

Considering the sample complexity bounds, and using Table 2.1 and the diagram of Figure 2.1 the same type of comparison can be performed between atan and bipolar exponential SNN's. It is found that from the point of view of learning, the bipolar exponential networks require slightly fewer training data points and might be preferred. However, no practical evidence was found the in the literature to support the above statement. In order to comply with the trend of the practical applications, in the next section, atan SNN's are chosen to represent the different families of SNN's.

10. As for the linear and Volterra approximators, since linear models are special cases of Volterra networks, the comparison is straightforward. The number of neurons in a linear model is limited to the dimensionality of the input, while in Volterra networks

it also depends on the order of approximation. Thus, in a Volterra network, the higher the order of the input variables to be considered in the approximation process, the more neurons are added to the hidden layer. The computational capabilities as well as the need for more training data then increase accordingly.

The bounds on the Volterra networks become more conservative as the order of the approximation is increased. This is due to the fact that for higher orders, the approximation of higher moments of the input with the corresponding orders of M becomes more conservative. This is not the case in other forms of neural networks. However, since in many real applications of Volterra networks, the second order approximation is used, the reported results might be more useful.

Since the focus of the rest of the research is on SNN's and RBFN's, no simulations are performed with Volterra or linear models. However, in the next section and during our description of the minimum complexity modeling, the characteristics of modeling with both Volterra or linear models are covered.

11. The value of these results goes beyond their theoretical significance. The size of the data set that results in PAC learning is a guide to practically useful results, and can therefore be used to guide a user towards the approximate model and complexity. Furthermore, the functional dependence of the bounds upon model structure can be used in designing cost functions that optimize both model accuracy and model complexity during identification. This issue will be further addressed in the next chapter.

2.11 Summary

The main issues addressed in this chapter are as follows.

- Using Hoeffding's inequality for i.i.d. data, a new equality that bounds the summation of a sequence of m-dependent r.v.s was derived.
- The definition of PAC learning paradigm and all its elements was extended to learning with m-dependent data.

- Assuming that the data are identically distributed according to the uniform distribution, the learning properties (including the sample complexity) of a general modeling task were presented.

- In the case of modeling with general families of RBFN's, sigmoid neural networks (SNN's) and Volterra networks, the sample complexity was evaluated in terms of the number of neurons as well as the size of the corresponding parameter space.

- The learning properties of some specific types of RBFN's and sigmoid neural networks were further specified, analyzed and compared with each other.

- The obtained results on learning of different families of neural networks were applied to assess typical examples of nonlinear system identification with neural networks.

- The functional dependence of the bounds on sample complexity upon model structure can be used to design learning-based cost functions to be minimized during the training phase.

- The model-free PAC learning with m-dependent data was defined. The general properties of such a scheme were evaluated and related to those of the standard PAC learning framework.

Chapter 3

Learning and Practical FIR Modeling

3.1 Introduction

The modeling approaches described in the previous chapter suffer from the following shortcomings.

1. None of the learning schemes searches for a model of the unknown function taking complexity into account as well as accuracy. In practice the algorithm should find an accurate model and at the same time avoid creating over-complex models.

2. Both conventional and model-free learning schemes require huge training data sets to assume learning, while in many applications the size of training set is small and fixed. Even if the learning inequalities ask for more data points, obtaining new data may not be possible.

Thus, a modeling task based on the previous approaches works best when some information about the complexity of the unknown function along with large training data sets are available. When such conditions can not be satisfied, a more sophisticated method has to be applied so as to provide the optimal set of parameters, as well as the minimal structural complexity given the size of the available data set [27], [28], [29].

One such method, introduced by Vapnik in [51], is known as "structural risk minimization." This algorithm, adapted and tuned towards our formulation of neural modeling, is given in Section 1. Practical algorithms that create a degree of balance between the theoretical justification and practical limitations of a typical modeling task are then given.

The chapter is organized as follows: Section 3.2 describes different versions of the structural risk minimization algorithm. In Section 3.3, complexity measures for different

families of RBFN's, sigmoid and Volterra neural models are found and the corresponding cost functions to be minimized during the training procedure are devised. The issue of the number of hidden neurons in different neural models is discussed in Section 3.4. In Section 3.5, based on the idea of Evolutionary Programming (EP), algorithms are introduced that can be applied to minimize the non-smooth cost function obtained in Section 3.3. The proposed EP method, along with two systematic evolutionary methods for neural modeling, are presented. Section 3.6 describes the results of a number of numerical simulations that test the performance of the proposed algorithms. The results of the chapter are discussed in Section 3.7. Finally, Section 3.8 gives the summary of the chapter.

3.2 Structural Risk Minimization Algorithm

The proposed neural modeling algorithms are extensions of the structural risk minimization algorithm [51], which is now described. Two versions of the structural risk minimization algorithm designed for neural modeling are given. In the following formulations, attention is focused on RBFN's and SNN's. Using the results of the previous chapter, similar algorithms can be devised easily for other neural networks. In the following formulations, the focus is given to RBFN's while all the fomulations can be extended to SNN's.

In the first formulation, assume that for all functions in $\mathcal{F}$: $|a_i|\sqrt{b_i} \leq M < \infty$ where $i = 1, \ldots, l$ and M is a fixed radius of an assumed hyper-sphere of parameters $|a_i|\sqrt{b_i}$. Now, notice that: $A_{rbfn}(l) = lM$ is an upper bound for $\sum_{i=1}^{l} |a_i|\sqrt{b_i}$. The assumption on the magnitude of the parameters makes the available learning bounds more conservative, but is necessary for the nested families of neural networks required by the structural risk minimization algorithm. The first version of the structural risk minimization algorithm can be described as follows.

Definition 3.2.1 *Assuming the pre-specified values of the accuracy ϵ, the confidence $(1 - \delta)$, M and the size of the available set of training data n, "The Structural Risk Minimization with Variable Structure Algorithm " is described as the following steps:*

Step 1: Consider a nested sequence of function sets $\mathcal{F}_l \subset \mathcal{F}_{l+1}$, where $\mathcal{F}_l$ is a family of RBFN's with l neurons, where $l_{low} \leq l \leq \kappa$. Notice that $A_{rbfn}(l)$ is a fixed value for the function set $\mathcal{F}_l$.

Step 2: Find the largest natural number l such that the corresponding inequality of sample complexity is satisfied and name it as "k",

Step 3: Perform the empirical risk minimization algorithm over the function set $\mathcal{F}_k$. The resulting function (for the particular family of RBFN's used in modeling) is the output of the algorithm.

As can be seen, the structural risk minimization finds the simplest function set (within a certain nested family of function sets) that provides learning with the pre-specified values of accuracy and confidence, using a training data set with a fixed size. In other words, this method searches for the simplest possible candidate in $\bigcup_{l=1}^{\infty} \mathcal{F}_l$ to approximate the unknown function, considering the size of the available training data set.

In order to define the second formulation, notice that a nested family of neural networks can be generated assuming a fixed number of neurons but grading the size of the parameter space. The nested family of functions with $\mathcal{F}_p \subset \mathcal{F}_{p+1}$ is thus formed assuming a fixed number of neurons l but a nested parameter space. In order to formulate this idea, and develop an alternative to the previous algorithm, fist consider a sequence of real numbers M_p, $p = 1, 2, \ldots$, such that $0 < M_p \leq M_{p+1}$. Next, for each integer p, define a set of functions in $\mathcal{F}_p$ for which: $|a_i|\sqrt{b_i} \leq M_p$. Then, with $A_{rbfn}(p) = lM_p$ where $0 < M_p \leq M_{p+1}$, a nested family of functions $\mathcal{F}_p \subset \mathcal{F}_{p+1}$ is formed. The second version of the structural risk minimization algorithm is:

Definition 3.2.2 *With all assumptions as in Definition 3.2.1, and assuming a sequence of non-decreasing values of $M_1, M_2, \ldots, M_{p_{high}}$, "The Structural Risk Minimization with Variable Parameter Size Algorithm" is described by the following steps:*

Step 1: Consider a nested sequence of function sets $\mathcal{F}_p \subset \mathcal{F}_{p+1}$ where $\mathcal{F}_p$ is a family of RBFN's with l neurons and parameter size of M_p as described above. Notice that $A_{rbfn}(p)$ is a fixed value for the function set $\mathcal{F}_p$.

Step 2: Let "k" be the largest integer p such that the corresponding inequality for sample complexity is satisfied,

Step 3: Perform the empirical risk minimization algorithm over the function set $\mathcal{F}_k$. The resulting function is the output of the algorithm.

Although the second method of generating the nested family gives no clear intuitive insight to the structural format of the network, it can be seen that if the algorithm assumes an l that is "large enough", the need to manipulate the network's structure is eliminated. Similar algorithms can be defined for sigmoid neural networks defining M_p as: $|a_i||b_{ij}| \leq M_p$ for all i and j. The discussion considers RBFN's, although similar arguments can be made regarding the algorithms with sigmoid neural models.

The following comments describe major disadvantages of the above algorithms.

1. It can be seen that both algorithms search for a neural model of the unknown function with some degree of minimal complexity. However, because of the use of conservative bounds the need for large training data sets remains as the main disadvantage of both algorithms. Consider a hypothetical modeling procedure where only a few hundred training data points are available. The first method tries to find an upper bound for l such that some levels of accuracy and confidence (e.g. $\epsilon = \delta = 0.05$) are guaranteed. Using the learning inequalities of the previous chapter for l, the typical upper bound obtained for the above-mentioned typical values is well below $l = 1$. In other words, it will be found that according to our bounds there is no model structure for which PAC learning can be assured.

2. In order to use either of the algorithms, some assumptions have to be made regarding either the number of hidden neurons or the size of the parameter space. For example, in the second algorithm, it might be difficult to guess what values of l are large enough for the problem at hand. Also, the size of the step under which M_p has to be increased is an issue that requires further attention.

3. Both algorithms bound $A_{rbfn} = \sum_{i=1}^{l} |a_i|\sqrt{b_i}$ using some assumption either on the structure or on the size of the parameter space. For example, the structural risk

minimization with variable structure algorithm assumes a fixed size of the parameter space, and uses the number of neurons to bound A_{rbfn} for each function set in the nested family of function sets. However, there is no guarantees in either approaches that such assumptions result in tight bounds on A_{rbfn}.

The learning results of the previous chapter thus may not be directly used for applications with a small number of training data points. These formulations, although carefully designed and theoretically correct, are successful only if the size of the training data set is large, since they depend on conservative bounds. Roughly speaking in all modeling tasks where the size of the training data is larger than a few thousand, the results of the learning results can be applied even though these results might be conservative estimates. Availability of large data sets is the main characteristic of an emerging field of research called "data mining". In a typical data mining procedure, the knowledge contained in a large database is to be extracted using a certain type of algorithm. Two of the most important data mining applications are marketing analysis and image processing, where the quantitative evaluation of the accuracy and confidence of the models is most important. For applications such as medical image processing and financial analysis, where huge training data sets (typically with more than a few hundred thousands or a few million data points) are available, the above algorithm can be directly applied to create accurate and reliable models of data. However, there exist many applications in science and engineering where the size of the available training set never exceeds a few hundred points.

Another issue that discourages the use of the empirical risk minimization is the difficulty of creating an $\epsilon/2$-cover of a function set. There exist specific procedures of forming function cover sets that are practically very time-consuming (see for example the procedure given in Chapter 5, for a distribution-free version of forming such cover sets). Although an ϵ-cover of a function set is not easy to form practically, the cardinality of a minimal cover, however, is known to be a highly informative indication of the complexity of a function family.

To obtain insight into the performance of neural models of practical systems, the theoretical structure of the statistical learning theory is used as a guide to practical modeling methods. The more practical methods lack the exact theoretical guarantees

provided by the learning theory.

3.3 Learning-Based Complexity Measures

In the next sections, neural modeling algorithms are proposed that use the results of the statistical learning theory to provide some degree of confidence over the similarity of the testing and training performances of the algorithms. In order to do so, a complexity term is introduced for each neural structure and this term becomes part of a cost function to be minimized during the training.

Overfitting and poor testing performance are caused by the use of over-complex models. In order to maintain a testing-training balance, the complexity of the model must be limited. The results of the statistical learning theory are used to present new measures of complexity that target the overfitting issue directly and provide an acceptable level of the testing-training balance.

As mentioned before, given a fixed size of training data set, n, and a certain family of networks, the deviation of the testing performance from the training performance is set by ϵ and δ. In order to avoid overfitting in an ideal case, both these parameters must be as small as possible, at the same time. However, from the learning results of the previous chapter, it can be seen that with a fixed number of training data points, reducing the value of the one of these two parameters would increase the bound on the other one. This means that with a fixed model and fixed number of training data, if higher accuracy is desired, one has to compromise the level of statistical confidence and vice versa. This issue parallels the bias-variance problem reported in the fields such as identification and modeling and asserts that often, both accuracy and confidence may not be minimized at the same time. To create a systematic modeling algorithm, the main objectives of a typical modeling task must be used with the learning properties to create a balance between these two parameters. In order to do this, one of the two parameters is fixed and the complexity measure is based on the other one. Notice that for a fixed level of disparity between the empirical error and the true error, δ gives the amount of uncertainty over the model. Therefore, it seems that for a typical modeling application, δ would be a good choice for a complexity measure to be minimized throughout the training process.

Defining the complexity measure based on δ will result in complexity measures that are closely related to the level of smoothness (Lipschitz constant), the size of the training set as well as the dimension of the input. This is not surprising, because modeling with functions that possess higher Lipschitz constants and larger dimensionality is known to be more complex and to require more data points.

Assuming that ϵ is constant, if δ (or a non-decreasing function of this parameter) is minimized during the training procedure, a model will be found for which the likelihood of having the difference between testing and training performances limited by ϵ is maximized. At this point, consider a set of Gausssian RBFN's. Similar procedures that give similar results for other neural structures are described later. Now, consider Inequality (2.34):

$$\delta \geq 2^{\left[\frac{2\sqrt{2}A_{rbfn}d(\beta-\alpha)}{\epsilon\sqrt{e}}\right]^d}(m+1)\exp\left[-n\epsilon^2/8(m+1)\right] \tag{3.1}$$

where:

$$A_{rbfn} = \sup_{\forall a,b}\sum_{i=1}^{l}|a_i|\sqrt{b_i}$$

Assume that: $\alpha = 0$, $\beta = 1$. As mentioned in Chapter 2, the choices of α and β are arbitrary and there is nothing special in the above choices. Also, note that the values of m, n, and δ are fixed during the training procedure. If the value of ϵ is given, the value of δ (or $\ln(\delta)$) is an indication of uncertainty of the model, i.e. minimizing $\ln(\delta)$ leads to models that are more reliable and perform more similarly on training and testing data sets. Therefore, define the complexity term $C_{gauss-np}$ as $\ln(\delta)$, or:

$$C_{gauss-np} = \left[\frac{2\sqrt{2}A_{rbfn}d}{\epsilon\sqrt{e}}\right]^d \ln(2) + \ln(m+1) - \frac{n\epsilon^2}{8(m+1)}\,. \tag{3.2}$$

This complexity measure gives an indication of the testing-training balance and shows how unreliable the training of such a network is. It is important to notice that the above measure combines the parameter-space complexity with the complexity due to the number

of neurons (structural complexity), without making any assumptions on the structure or parameters. This means that any algorithm that minimizes (3.2) addresses the parameter space complexity and the structural complexity at the same time. It can be observed that the complexity term introduced in (3.2) takes into consideration the size of available training data set n, the total dimension of input d, as well as the Lipschitz constant as an indication of smoothness of the function set.

Now, suppose that during the training algorithm the number of neurons l is fixed. Then the objective of the optimization algorithm is to include both the above complexity term and the empirical error in a cost function, i.e. :

$$\begin{aligned} J_{gauss-np} &= \frac{1}{n}\left[\sum_{i=1}^{n} \boldsymbol{l}(f(x_i), y_i)\right] \\ &\quad + \lambda\left[\left(\frac{2\sqrt{2}A_{rbfn}d}{\epsilon\sqrt{e}}\right)^{d} \ln(2) + \ln(m+1) - \frac{n\epsilon^2}{8(m+1)}\right] \end{aligned} \tag{3.3}$$

In the above cost function, λ is a weighting factor that determines the relative importance of the empirical error and the complexity term in the overall cost function. Lower values of λ result in models that create small training errors but exhibit poor performance over the testing data. The higher values of λ create a desirable testing-training balance with both testing and training errors undesirably increased (due to resulting large empirical errors). A practical approach in choosing the value of λ is described later in this chapter. The function $\boldsymbol{l}(.,.)$ is chosen to be a loss function that satisfies a uniform Lipschitz condition as defined in (2.61).

The function introduced as $J_{gauss-np}$ is not a practical objective function since it includes a term A_{rbfn} that nests the parameter space. In other words, for a chosen value of A_{rbfn}, the space in which the parameters are allowed to vary is restricted. Then a higher value of A_{rbfn} defines a new parameter space which includes the previous space. In practice it is very difficult to create such a nested sequence of function sets based on a term such as

A_{rbfn} and then search for the simplest function set with satisfactory performance. Also, performing minimization of the error for a fixed value of A_{rbfn} requires a solid method of constrained optimization to make sure that the resulting set of optimal parameters is indeed within the space specified by the prespecified value of A_{rbfn}. These limitations make the entire process of optimization within each of the function sets impractical.

In order to obtain a sub-optimal solution, compromise is necessary. An optimization algorithm that is capable of minimizing non-smooth functions over the entire parameter space allows the use of a more practical complexity term which is formed by replacing A_{rbfn} with $\sum_i^l |a_i|\sqrt{b_i}$. During the consecutive iterations, the optimization algorithm now moves in favor of functions that fall within a smaller parameter space and as a result are less complex. This leads to the following more practical complexity term:

$$C_{guss} = \left(\frac{2\sqrt{2}(\sum_i^l |a_i|\sqrt{b_i})d}{\epsilon\sqrt{e}}\right)^d \ln(2) + \ln(m+1) - \frac{n\epsilon^2}{8(m+1)} \tag{3.4}$$

and the corresponding cost function:

$$\begin{aligned} J_{guss} &= \frac{1}{n}\left[\sum_{i=1}^{n} l(f(x_i), y_i)\right] \\ &\quad + \lambda\left[\left(\frac{2\sqrt{2}(\sum_i^l |a_i|\sqrt{b_i})d}{\epsilon\sqrt{e}}\right)^d \ln(2) + \ln(m+1) - \frac{n\epsilon^2}{8(m+1)}\right] . \end{aligned} \tag{3.5}$$

In the new cost functions, the values of $\sum_i^l |a_i|\sqrt{b_i}$ are kept as small as possible without being concerned about the over-all grading of the space through the fixed values of A_{rbfn}. Even with the relaxed definitions, since (3.5) is a non-differentiable function of the a_i's and perhaps of the b_i's, gradient-based minimization methods cannot be used for the optimization phase. Performing optimization over a_i's and b_i's thus calls for an algorithm that can minimize nonlinearly and non-smoothly parameterized models. The issue of

minimizing such cost functions will be discussed later in this chapter.

Next, following the same type of approach, complexity terms and cost functions are defined for the neural structures discussed in the previous chapter. Just as for Gaussian RBFN's, complexity terms are introduced that reflect the functional dependency of $\ln(\delta)$ and contain information about the learning properties of the modeling task.

Starting with RMQ-RBFN's, use Inequality (2.37) and apply the logarithm on both sides to define the complexity term and the corresponding cost function as follows:

$$C_{rmq} = \left(\frac{4(\sum_i^l |a_i|\sqrt{b_i})d}{3\sqrt{3}\epsilon}\right)^d \ln(2) + \ln(m+1) - \frac{n\epsilon^2}{8(m+1)} \tag{3.6}$$

and:

$$\begin{aligned} J_{rmq} &= \frac{1}{n}\left[\sum_{i=1}^{n} l(f(x_i), y_i)\right] \\ &\quad + \lambda\left[\left(\frac{4(\sum_i^l |a_i|\sqrt{b_i})d}{3\sqrt{3}\epsilon}\right)^d \ln(2) + \ln(m+1) - \frac{n\epsilon^2}{8(m+1)}\right] . \end{aligned} \tag{3.7}$$

For an "atan" neural network, following a procedure similar to that of RBFN's, define:

$$C_{atan} = \left[\frac{4d\sqrt{\sum_{k=1}^{d}\left[\sum_{i=1}^{l}|a_i||b_{ik}|\right]^2}}{\pi\epsilon}\right]^d \ln(2) + \ln(m+1) - \frac{n\epsilon^2}{8(m+1)} \tag{3.8}$$

and:

$$J_{atan} = \frac{1}{n}\left[\sum_{i=1}^{n} \boldsymbol{l}(f(x_i), y_i)\right] + \lambda\left[\left[\frac{4d\sqrt{\sum_{k=1}^{d}\left[\sum_{i=1}^{l}|a_i||b_{ik}|\right]^2}}{\pi\epsilon}\right]^d \ln(2) + \ln(m+1) - \frac{n\epsilon^2}{8(m+1)}\right] . \tag{3.9}$$

In the case of bipolar exponential sigmoid functions:

$$C_{bexp} = \left[\frac{\sqrt{d\sqrt{\sum_{k=1}^{d}\left[\sum_{i=1}^{l}|a_i||b_{ik}|\right]^2}}}{\epsilon}\right]^d \ln(2) + \ln(m+1) - \frac{n\epsilon^2}{8(m+1)} \tag{3.10}$$

and:

$$J_{bexp} = \frac{1}{n}\left[\sum_{i=1}^{n} \boldsymbol{l}(f(x_i), y_i)\right] + \lambda\left[\left(\frac{\sqrt{d\sqrt{\sum_{k=1}^{d}\left[\sum_{i=1}^{l}|a_i||b_{ik}|\right]^2}}}{\epsilon}\right)^d \ln(2) + \ln(m+1) - \frac{n\epsilon^2}{8(m+1)}\right] \tag{3.11}$$

Finally for a family of second order Volterra networks as described in the previous chapter:

$$C_{Volt} = \left\lceil \frac{2B\sqrt{d}}{\epsilon} \right\rceil^d \ln(2) + \ln(m+1) - \frac{n\epsilon^2}{8(m+1)} \tag{3.12}$$

where B is as defined as:.

$$\begin{aligned} B &= |a_k| + (1/2) \sum_{i=1\,,\,i\neq k}^{d} |b_{ik}| + |b_{kk}| + (1/4) \sum_{i=1}^{d} \sum_{j=i\,,\,j\neq k}^{d} |c_{ijk}| \\ &+ (1/2) \sum_{i=1\,,\,i\neq k}^{d} |c_{ikk}| + (3/4)|c_{kkk}| \,. \end{aligned}$$

Based on this complexity term, define the following cost function:

$$\begin{aligned} J_{volt} &= \frac{1}{n} \left[\sum_{i=1}^{n} \mathit{l}(f(x_i), y_i) \right] \\ &\quad + \lambda \left[\left\lceil \frac{2B\sqrt{d}}{\epsilon} \right\rceil^d \ln(2) + \ln(m+1) - \frac{n\epsilon^2}{8(m+1)} \right] . \end{aligned} \tag{3.13}$$

As mentioned above, the complicated form of these cost functions calls for an optimization algorithm which can minimize non-differentiable nonlinear functions. Before describing the minimization algorithms, some issues regarding the number of hidden neurons in different neural structures must be addressed.

3.4 Number of Neurons in Hidden Layer

As seen in the structural risk minimization algorithm, it is often necessary to define a range of values for the number of hidden neurons: $l_{low} \leq l \leq \kappa$. The upper limit κ depends on the size of the training set and the dimension of the input, i.e. $\kappa = \kappa(n, d)$.

In practice, the number of parameters should be significantly smaller that the number of training samples. This is an idea also supported by the results of the learning theory. However, the number of neurons alone does not determine the complexity of the network. As shown in the previous chapter, the size of the parameter space along with the number of neurons also influence the complexity.

In practical neural modeling tasks, $\kappa(n, d)$ is often set as a multiple of the total number of parameters in the network. This explains why in general κ is also a function of d. For SNN's, since the number of parameters (which is the same as the number of weights) is equal to: $(d+1)l$, follows that:

$$n > Q(d+1)\kappa \tag{3.14}$$

where Q is an integer that is normally set to a number between 2 and 10. The range 2 to 10 comes from the general idea of having 2 to 10 times as many training points as the parameters (see [10] for example). Condition (3.14) suggests the following expression for $\kappa(n, d)$:

$$\kappa(n, d) = \frac{n}{Q(d+1)} \tag{3.15}$$

A similar argument for the number of parameters for RBFN's, can suggest an appropriate functional dependency for κ. Notice that for a RBFN, the number of parameters does not explicitly depend on the dimensionality of the input and is equal to: $2l$. This suggests the following expression for κ:

$$\kappa(n) = \frac{n}{2Q} \tag{3.16}$$

3.5 Evolutionary Neural Modeling

The cost functions introduced in the previous section are unsuitable for gradient-based optimization methods. On the other hand, evolutionary algorithms are often designed to optimize complicated non-differentiable cost functions. Like other types of optimization algorithms, there are many different versions of evolutionary optimization methods. Evolutionary algorithms use "natural selection", "reproduction", and"mutation" operations to search among the members of a pool of possible solutions. During the process of natural selection, the members of a generation of species (i.e. possible solutions) are evaluated based on their level of "fitness" to the environment (i.e. by a cost function), and the best solutions are selected. Then these selected candidates are reproduced to create a new pool of solutions called "offspring". Finally, the candidates in the new pool of solutions are randomly mutated according to their level of fitness, i.e. the ones with better fitness are mutated less. Notice that due to the process of mutation the new solutions (offspring) are no longer the same as their parents, but somewhat similar to them. The resulting pool of candidates is now treated as the new generation of solutions and goes through the next evolution cycle. Some other evolutionary operations such as "cross-over" are also used in a family of evolutionary algorithms (including the Genetic Algorithm), that might help the process of evolution. The use of cross-over operation, however, requires creating chromosome strings for each of the solutions, which makes the entire process more computationally intensive. Also, there exist infinite methods to map the information of the network to a chromosome, while little is known on how these methods influence the optimization process. It is also claimed that the cross-over operation distorts the topology of the neural networks, and as a result may not be an appropriate operation when dealing with optimization of neural networks. As a result, here the three fundamental operations of natural selection, reproduction, and mutation that form the basis of Evolutionary Programming (EP). Performing optimization tasks using EP often calls for enormous computation time. This is considered as the main disadvantage of EP-based algorithms.

EP algorithms can be easily designed for a particular problem so that better solutions of the problem are selected and reproduced throughout the process. A careful design of

the algorithm can result in a faster approach to a sub-optimal solution. Here, the EP method is adapted to the training of a feedforward neural network as described below.

First an EP method for the fixed-structure neural networks is introduced. This EP method first creates an initial generation of the networks, all with a fixed number of neurons. The cost function for each of the networks is then evaluated. Networks with the lowest cost function are then selected and the rest of the models are discarded. Then, taking into account the cost function, the parameters of the selected networks are mutated. By adding a vector of normally-distributed random numbers with the variance proportional to the cost of the network, the selected networks are mutated. The mutation process will be further specified later in this chapter. Then the cycle is repeated by selecting the networks with the lowest cost function in the new generation.

A second algorithm is used for variable-structure neural networks. At each stage of this algorithm, the EP method with the fixed-structure (as described above) is first performed for the networks with a certain number of neurons. After iterating for a number of generations, if the cost function is not decreasing fast enough as the new generations are produced, the method increases the number of neurons in the hidden layer and searches for the desirable network within the new structure, again using the fixed-structure algorithm. When even adding new neurons does not make the cost function decrease fast enough, the algorithm stops the search. Since increasing the number of neurons creates a nested family of functions, this method systematically searches for the simplest function of the nested family of the neural networks that can provide us with the desired level of the defined cost function.

The cost functions evaluated in the above algorithms are the ones defined in this chapter, based on the learning properties of each neural structure. The condition that determines the termination of the search is expressed in terms of the slope of the curve of the cost function versus the generation number.

The proposed fixed-structure algorithm can be described in more details as the following stepwise procedure:

Definition 3.5.1 *Suppose n input-output training data are given. Consider a family of*

neural networks with l neurons. Given a certain value of λ, the objective is to find the simplest neural model of the family that minimizes the cost function J. Also, assume that the size of each generation is N, and that the integer N_p is selected such that $N_s = N/N_p$ too is an integer. In each selection procedure, select the best N_s members of the networks of the generation. Set the values ω_a and ω_b as the mutation rates (annealing rates) of parameters a_i's and b_i's, respectively. Define the slope of the curve "cost function vs. generation number" as:

$$Slope = \frac{J(l,GenStep)-J(l,GenStep-BackStep)}{BackStep}.$$

where: $J(l, GenStep)$ is the cost function of the network of generation GenStep with minimal cost function and BackStep is the number of back steps, based on which the slope of the cost function curve is calculated.

Also assume the minimum acceptable slope of the cost function before the training is stopped is given as MinSlope. The "evolutionary fixed-structure neural modeling algorithm" is performed as follows:

Step 0: Create N networks (each having l neurons) to form the initial generation. In order to do so, randomly assign parameters (a_i's and b_i's) of the networks to standard normally-distributed random numbers.

Step 1: Evaluate the cost function for the networks of the present generation, i.e. find J for all networks of the pool. Also set: $GenStep = 1$

Step 2: Select the best N_s networks that generate the smallest cost functions, and discard the rest.

Step 3: In order to generate a new generation with N members, mutate each of N_s remaining networks in N_p forms. More specifically:

$$\begin{aligned} a_{new,r} &= a_{old} + w_a \, J_{old} \, Rand_{a,r} \\ b_{new,r} &= b_{old} + w_b \, J_{old} \, Rand_{b,r} \end{aligned}$$

where $1 \leq r \leq p$, variables $Rand_{a,r}$ and $Rand_{b,r}$ are normally distributed random vectors, and J_{old} is the value of the cost function calculated for the old network.

Step 4: Find the network that gives the smallest J, and name its cost function as: J_{best}. Assign $J(l, GenStep) = J_{best}$. If $GenStep \leq BackStep$, increment $GenStep$ by one and go to Step 2. Otherwise, calculate:

$$Slope = \frac{J(l,GenStep)-J(l,GenStep-BackStep)}{BackStep}.$$

Step 5: If $Slope < MinSlope$, increment $GenStep$ by one and go to step 2; otherwise, save the parameters of the best network of the pool, and introduce this network as the output of the algorithm.

A simple flow chart of this algorithm is shown in Figure (3.1).

The slope of the cost function curve versus generation number determines when the training is stopped, i.e. when producing new generations of solutions does not make the cost function decrease fast enough the algorithms stops. Also, notice that the weights of each network are mutated according to the fitness (i.e. the cost function) of the network. Parameter *BackStep* determines how many generations are being used to evaluate the slope of the cost function, while *MinSlope* gives the threshold value for the slope before the training is stopped.

The variable-structure version, which includes the above algorithm as its one special case, is a combination of the two versions of the structural risk minimization algorithm and can be describes as follows:

Definition 3.5.2 *Suppose n input-output training data are given. A nested family of neural networks is formed of a family of neural networks with $l_{low} \leq l \leq \kappa$. Given a certain value of λ, the objective is to find the simplest neural model of the nested family that minimizes the cost function J. Also, assume that the size of each generation is N, and that the integer N_p is selected such that $N_s = N/N_p$ too is an integer. In each selection procedure, select the best N_s members of the networks of the generation. Set the values ω_a and ω_b as the mutation rates (annealing rates) of parameters a_i's and b_i's, respectively. Also assume the minimum acceptable slope of the cost function before the training is stopped is given as MinSlope, and the number of back steps, based on which*

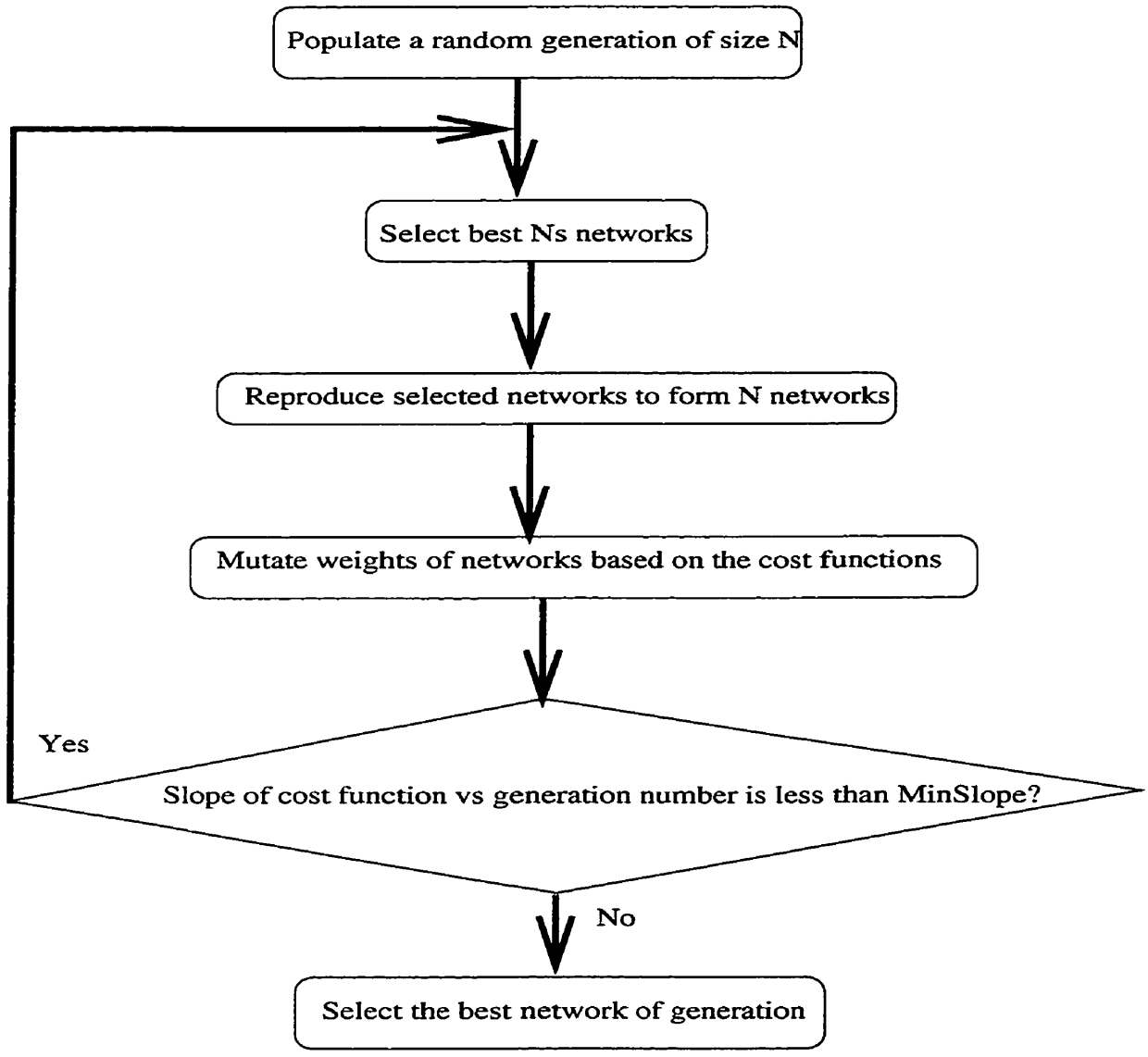

Figure 3.1: Flow chart of fixed-structure evolutionary neural modeling

the slope of the cost function curve is calculated, is set to BackStep. The "evolutionary variable-structure neural modeling" is performed as follows.

Step 0: Set $l = l_{low}$. *Also, set GenSlope to a negative number with large absolute value, and* $J_{opt}(l_{low})$ *to a very large number.*

Step 1: If $l > \kappa - 1$ *or* $GenSlope \geq MinSlope$ *or* $J_{opt}(l) \geq J_{opt}(l-1)$ *go to Step 6; otherwise* $l = l + 1$. *Create N networks (each having l neuron) to form the initial generation. In order to do so, if* $l = l_{low}$, *randomly assign parameters (* a_i*'s and* b_i*'s) of the networks to arbitrary numbers. Otherwise use parameters of the last generation of networks with* $l - 1$ *neurons and add a new neuron with small randomly generated parameters to each of N networks. Evaluate the cost function for the networks of the present generation, i.e. find J for all networks of the pool.*

Step 2: Select the best N_s *networks that generate the smallest cost functions, and discard the rest.*

Step 3: In order to generate a new generation with N members, mutate each of N_s *remaining networks in* N_p *forms. More specifically:*

$$\begin{aligned} a_{new,r} &= a_{old} + w_a \, J_{old} \, Rand_{a,r} \\ b_{new,r} &= b_{old} + w_b \, J_{old} \, Rand_{b,r} \end{aligned}$$

where $1 \leq r \leq p$, *variables* $Rand_{a,r}$ *and* $Rand_{b,r}$ *are standard normally distributed random vectors, and* J_{old} *is the value of the cost function calculated for the old network.*

Step 4: Find the network that gives the smallest J, and name as: J_{best}. *Assign* $J(l, GenStep) = J_{best}$. *If* $GenStep \leq BackStep$ *go to Step 2. Otherwise, calculate:*

$$Slope = \frac{J(l,GenStep) - J(l,GenStep - BackStep)}{BackStep}.$$

Step 5: If $Slope < MinSlope$ *go to step 2; otherwise, save the parameters of the best network of the pool, set* $GenSlope = \frac{J(l,GenStep) - J(l,1)}{GenStep - 1}$, $J_{opt}(l) = J(l, GenStep)$, *and go to step 1.*

Step 6: Among the best networks with l and $l-1$ *neurons, find the one with the smaller cost function, and introduce this network as the output of the algorithm.*

As can be seen, in the evolutionary variable-structure neural modeling algorithm the number of hidden neurons is determined by the algorithm and throughout the training phase. The assumption of $l_{low} = \kappa$ reduces the second algorithm to the first one, and as a result, the first algorithm is a special case of the second one. The evolutionary fixed-structure neural modeling algorithm will be used in simulations to show the practical importance of the cost functions in avoiding the overfitting problem.

The above algorithms are presented in a general format, and when they are to be applied to a particular family of neural networks, some items must be further specified. Here is a list of suggestions about the use of the algorithm to train SNN's and RBFN's:

1. For SNN's, all random numbers can be generated according to a zero-mean normal distribution with variance of 1.

2. For RBFN's, the same form of random generator can be used. However, in order to avoid negative values of b_i, a mechanism should be included in the algorithm that assigns b_i to a small positive real number if the random generator tries to set b_i to a negative value.

3. In learning of variable-structure RBFN's, in order to make sure that families of neural networks are actually nested, the set of centers $\{c_i,\ i = 1, \ldots, l\}$ for a network of l neurons should be included in the set of centers for any network with $l + 1$ neurons, for all l.

Some of the advantages of the above algorithms are as follows.

1. The cost function to be minimized reflects the learning complexity of the model and can create a balance between the testing and training performances of the resulting network.

2. Algorithms using gradient-based optimization methods can get stuck in a local minimum, while the EP method used here is less susceptible to the local minima problem.

3. The variable-structure algorithm performs optimization on both parameters and structure of the network at the same time and tries to avoid overfitting.

The disadvantages of the algorithm can be listed as follows.

1. From the design of the algorithms, it can be seen that the form of the model is used only in evaluation of the cost function and plays no role in the way the parameters are updated. This is in contrast to methods such as gradient-based algorithms where the exact form of the model is explicitly used in updating the weights of the model. This lack of attention to the exact model form makes EP algorithms applicable to wider families of optimization tasks, while it makes each particular optimization process less efficient and more time-consuming.

2. Different choices of variables λ, *MinSlope*, w_a, w_b and *BackStep* may result in different models. The designer must use a series of runs of algorithms with different settings to find the suitable set of the above variables.

3. In both algorithms, the extensive use of random numbers throughout the learning process causes the entire process of learning to be a stochastic procedure. The end result of the training task may be different if different sets of random numbers are used.

4. The evolutionary variable-structure neural modeling implicitly assumes the convexity of the curve of cost function versus generation number. This is because as soon as $J_{opt}(l) \geq J_{opt}(l-1)$, the algorithm stops and introduces the network with $l-1$ neurons as the output function. Considering the stochastic nature of the algorithm, it can be imagined that in some cases, if the algorithm is allowed to continue the search, a network with $l+1$ or even more neurons can be found for which the cost function is smaller than that of the network with $l-1$ neurons.

5. Just like any other EP-based algorithm, the introduced neural modeling algorithms are computationally intensive and require a significant amount of computation time to obtain a sub-optimal solution.

The first disadvantage, on the other hand, allows the designer to have more control over the performance of the algorithm. The second disadvantage is characteristic of any random search and is the price paid to obtain some level of flexibility in dealing with local

minima. The only solution for the third disadvantage might be allowing the algorithm to continue the search for all $l_{low} \leq l \leq \kappa$, and find the best solution after the search for all l is done. This solution, although theoretically appealing, may not be desirable in many practical applications, as for larger values of l the search process becomes extremely time-consuming.

3.6 Simulation Results

In this section, simulation results using the proposed algorithms in FIR modeling of a simulated system are presented. In all simulations, the loss function is defined as:

$$l(y, h(x)) = |y - h(x)| \,. \tag{3.17}$$

3.6.1 Simulation 1

In order to generate the data, first generate a set of independently and uniformly distributed r.v.s $\{\zeta_i\}_{i=1}^{401}$, where $0 \leq \zeta_i \leq 1$ for all i. Then form a sequence of uniformly distributed m-dependent r.v.s $x_1, x_2, \ldots, x_{400}$, where: $x_i = (\zeta_i \ , \ \zeta_{i+1})$, $i = 1, \ldots, 400$, and $m = 2$. The system to be modeled is generated by a nonlinear function as follows:

$$\forall \xi = (\xi_1 \ \xi_2) \in [0 \ 1]^2, \ f(\xi) = \frac{1}{15}\left(2\xi_1^2 + \xi_2^2 - \xi_1\xi_2 - 2\xi_2 + 2\right)(1 - 2\sin(7\xi_2)) \tag{3.18}$$

The 3-dimensional graph of this function for all values in $[0 \ 1]^2$ is shown in Figure 3.2. From the random samples described above, the first 100 samples are used for training of the network and the remaining points for testing. The neural structure which is used for modeling is a family of atan-SNN's with 3 neurons.

Assuming $\lambda = 0$, the evolutionary fixed-structure neural modeling algorithm is applied with $l = 3$ neurons to observe the resulting neural model when the cost function is based only on the empirical error. Later in the second simulation, the results of this simulation are compared with a case where λ is non-zero. The parameters of the algorithm are set

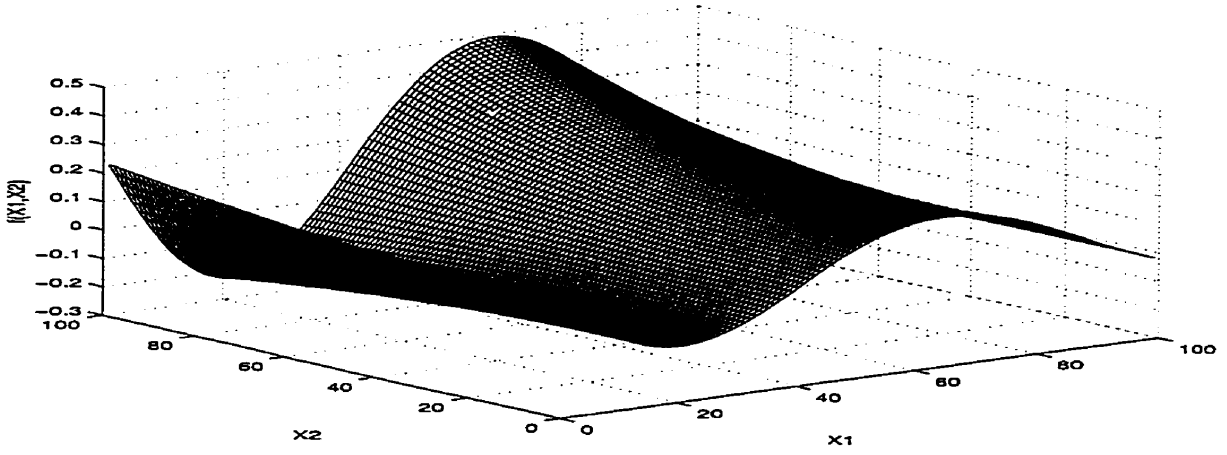

Figure 3.2: Three dimensional graph of function f

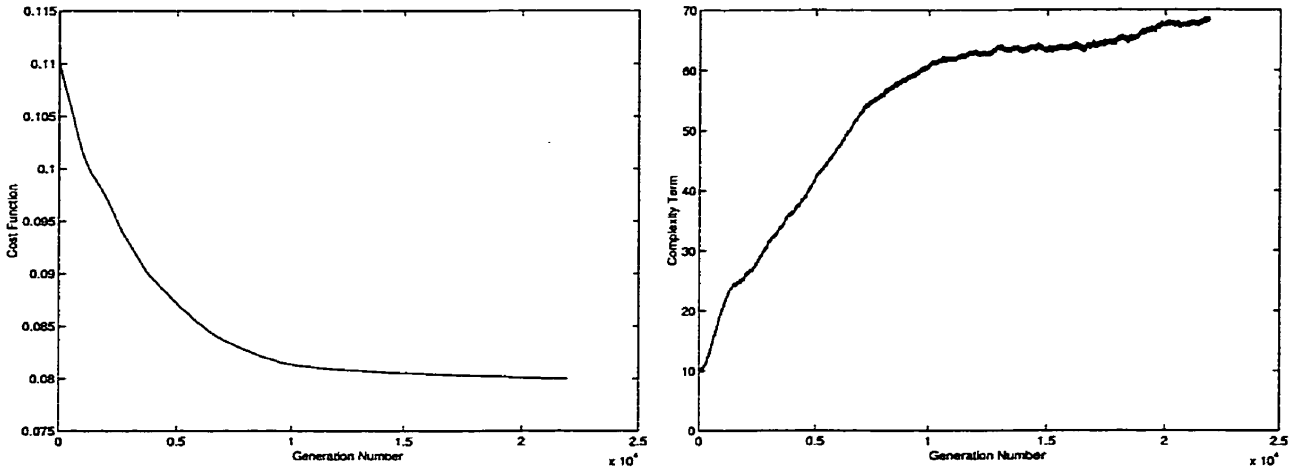

Figure 3.3: (a) Cost function (left), and (b) Complexity term (right) for Simulation 1

as follows: $MinSlope = -1 \times 10{-}8$, $BackStep = 1000, w_a = w_b = 0.01$, $\lambda = 0$, $\kappa = 100$, $N = 50$.

With this choice of settings, the training curve of Figure (3.3) portrays the evolution of the best network of each generation. Figure (3.3.a) depicts the cost function for the best network of each generation versus the generation number. The complexity term C_{atan} for the best network of each generation is shown in Figure (3.3.b), which indicates that during the training and in order to find a better fit, the overall size (magnitude) of parameters has increased. The performance of the resulting network is shown in Figure (3.4) with the actual (solid) and estimated (dashed) outputs for 50 points in the testing

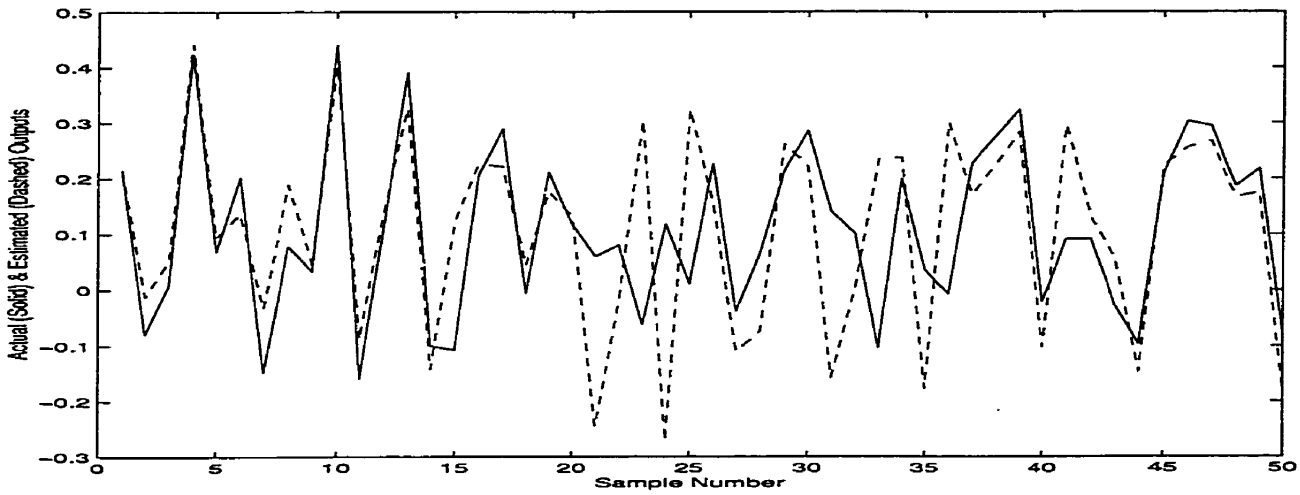

Figure 3.4: Actual (solid) and estimated (dashed) outputs for testing data (Simulation 1)

set.

Considering the best network of the final generation as the output of the algorithm, the training cost function (which is the purely averaged sum of absolute errors between the actual and estimated outputs) is 0.079, while testing the network against the rest of the data points (i.e. the testing set) results in the empirical error of 0.097. Even though 0.079 may not be a desirable level of accuracy, the significant difference of the testing and training errors seems to be a more important cause of concern. A testing error that is significantly larger than the training error shows that the model is not a reliable one and can mislead the user. In other words, the purpose of this simulation is not to find very accurate models but to exhibit the effect of using a learning-based cost function on the issue of testing-training balance.

3.6.2 Simulation 2

In the second simulation, the same function, data, model, and algorithm as Simulation 1 are used, but λ is set to 0.001 rather than zero. The complexity term C_{atan} is now included in the cost function. All the parameters of the algorithm (including all random numbers used for mutation as well as initialization of weights) are exactly as used in the previous simulation.

Figures (3.5.a) and (3.5.b) depict the cost function and complexity term C_{atan} curves

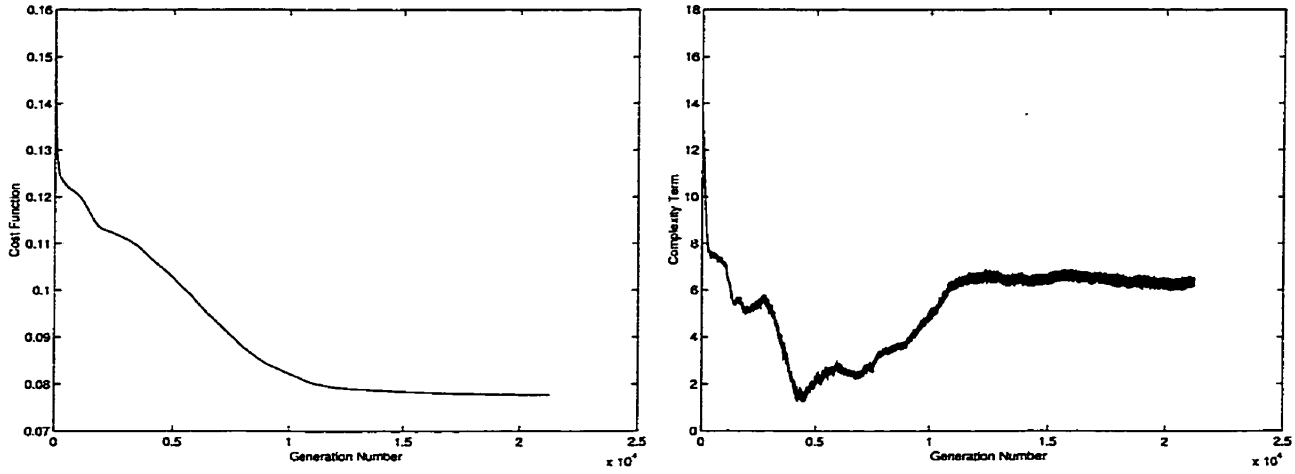

Figure 3.5: (a) Cost function (left), and (b) Complexity term (right) for Simulation 2

of the learning task, respectively. The training cost function of the resulting neural model is 0.077 of which 0.070 comes from the empirical error. This shows that the resulting empirical error is very close to that of Simulation 1. By comparing the complexity curve of Simulation 1 (Figure (3.3.b)) with that of Simulation 2 (Figure (3.5.b)), the significant difference between the magnitude of C_{atan} in the two cases can be seen. From Chapter 2 it is known that the smaller C_{atan} leads to a model that performs more similarly on testing and training data sets. Now, observe that the value of C_{atan} in Simulation 1 (i.e. 70) is considerably larger than that of Simulation 2 (i.e. 6). This suggests that the neural model obtained in Simulation 2 might outperform that of Simulation 1 on the testing data.

The neural model of Simulation 2 is assessed using the same testing data set as for Simulation 1. The testing cost function for the resulting neural model is 0.081, which is close to the training cost. Although none of the errors may be small enough, the proximity of testing and training errors indicates that the cost function with non-zero λ has helped the model avoid overfitting. In order to assess the quality of modeling for the network of Simulation 2, in Figure (3.6) the actual and estimated outputs of this model for the first 50 points of the testing set are depicted. Comparing Figures (3.4) and (3.6), one can observe the superiority of the neural model of Simulation 2 over that of Simulation 1, over the testing data set. This shows that the testing performance of the

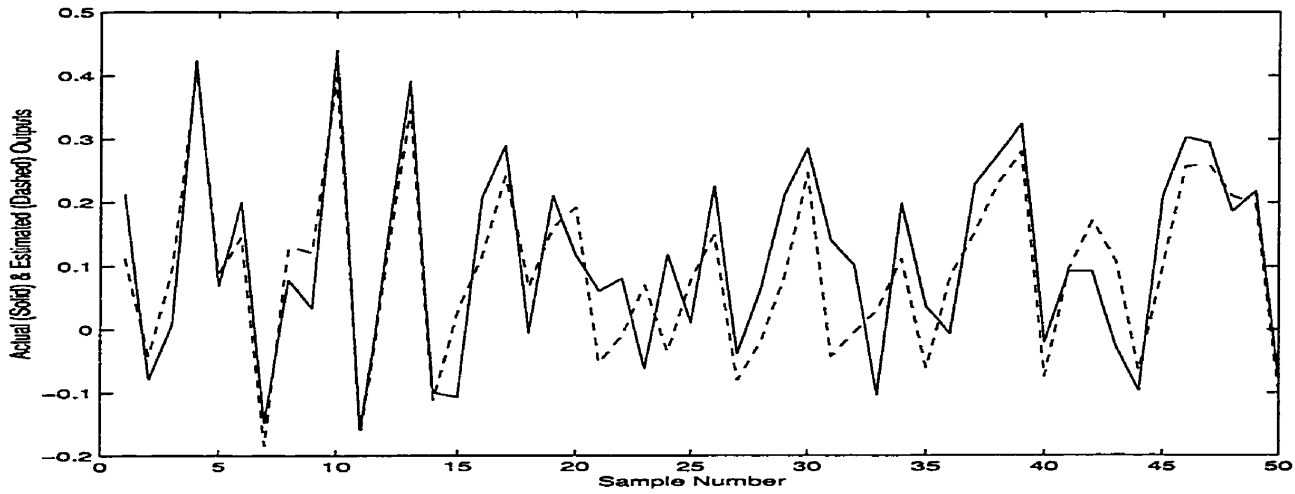

Figure 3.6: Actual (solid) and estimated (dashed) outputs for testing data (Simulation 2)

model of Simulation 1 is significantly less than its training performance, while the testing and training performances of the model obtained in Simulation 2 are very close to each other. This further supports the idea that the use of the learning-based cost functions can avoid overfitting.

3.6.3 Simulation 3

In the next simulation, the same function and set of training data (with 100 data points) are used, but in this simulation, the evolutionary variable-structure neural modeling algorithm has been used. The algorithm assumes $l_{low} = 2$ and $\kappa = 50$. Similar to the previous simulations, the remaining 300 points are used for testing evaluation, as before. The weighting factor λ is set to 0.001, and the rest of the parameters are the same as the previous simulations. Figures (3.7.a) and (3.7.b) show the cost function and complexity curves of this algorithm, respectively. The sharpest falls of the cost function occur when neurons 4, 9, and 12 are added at iterations 26280, 72990, and 82898, respectively.

The resulting network has 29 neurons in its hidden layer and its training cost function is 0.038 . For the testing phase, the cost function value increases to 0.043 which is slightly larger than the training cost error. Figure (3.8) shows the behaviour of the resulting model against the first 50 points of the testing set. A brief glance at Figure (3.8) shows that the resulting network outperforms the networks of Simulations 1 and 2 on the testing

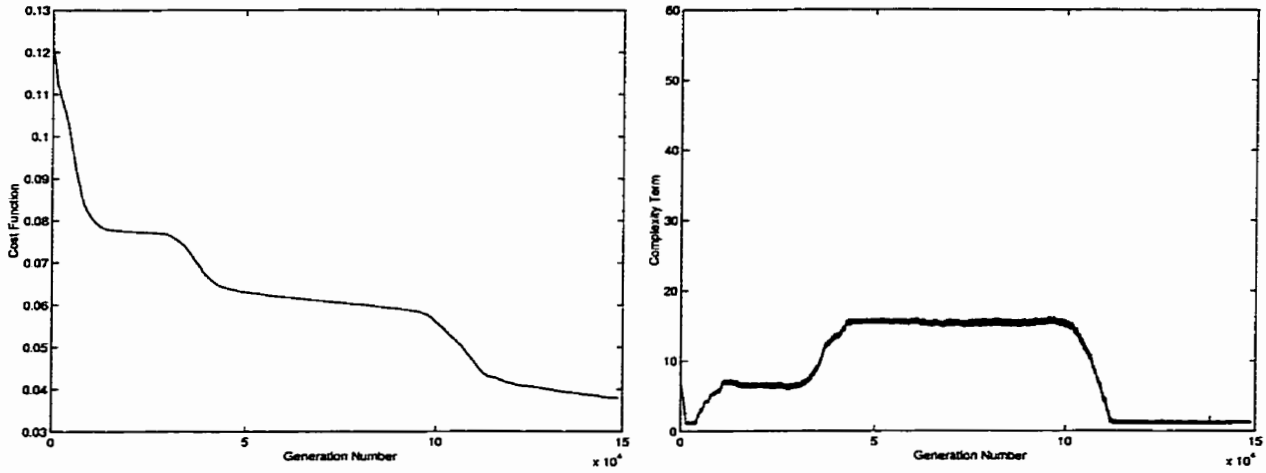

Figure 3.7: (a) Cost function (left), and (b) Complexity term (right) for Simulation 3

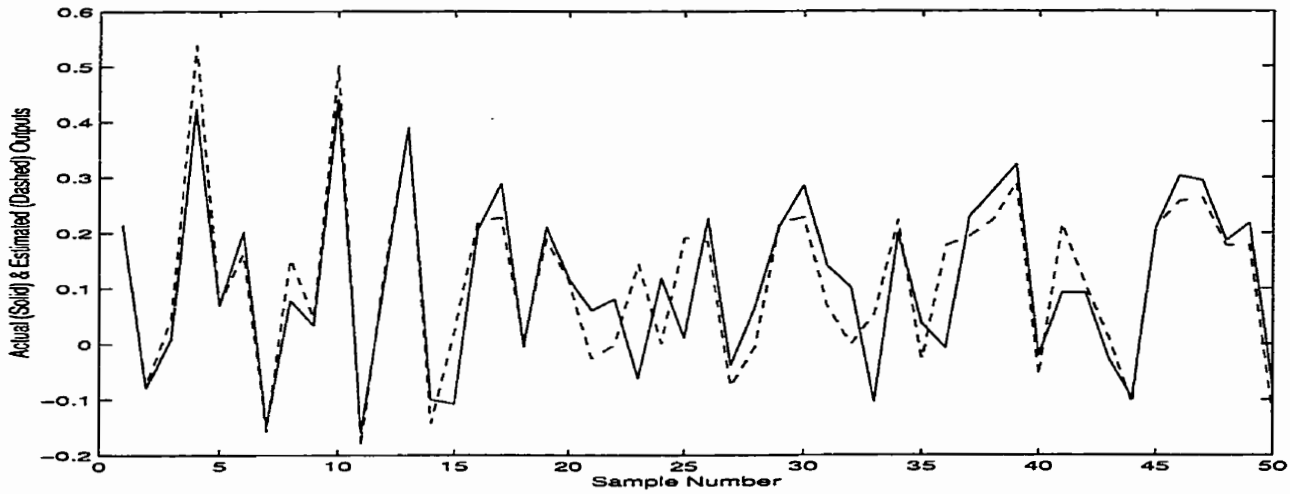

Figure 3.8: Actual (solid) and estimated (dashed) outputs for testing data (Simulation 3)

data, as far as training and testing error are concerned.

In the next three simulations, RMQ-RBFN's are used. A set of reasonable centers for the basis functions is chosen as follows. Set the first centre as: $c_1 = (0.5\ 0.5)$. For the second center, choose $c_2 \in [0\ 1]^2$ such that the distance between c_2 and c_1 is maximized. This would give us one of the vertices of $[0\ 1]^2$. For c_3, search for a two dimensional point in $[0\ 1]^2$ whose distance from the closest existing center is maximum. In this special case, another vertex of $[0\ 1]^2$. The rest of the centers are selected in the same recursive manner, i.e. find a point on $[0\ 1]^2$ whose distance from the closest existing centers is maximum and add it to the set of centers. This method of forming the centers makes sure that the

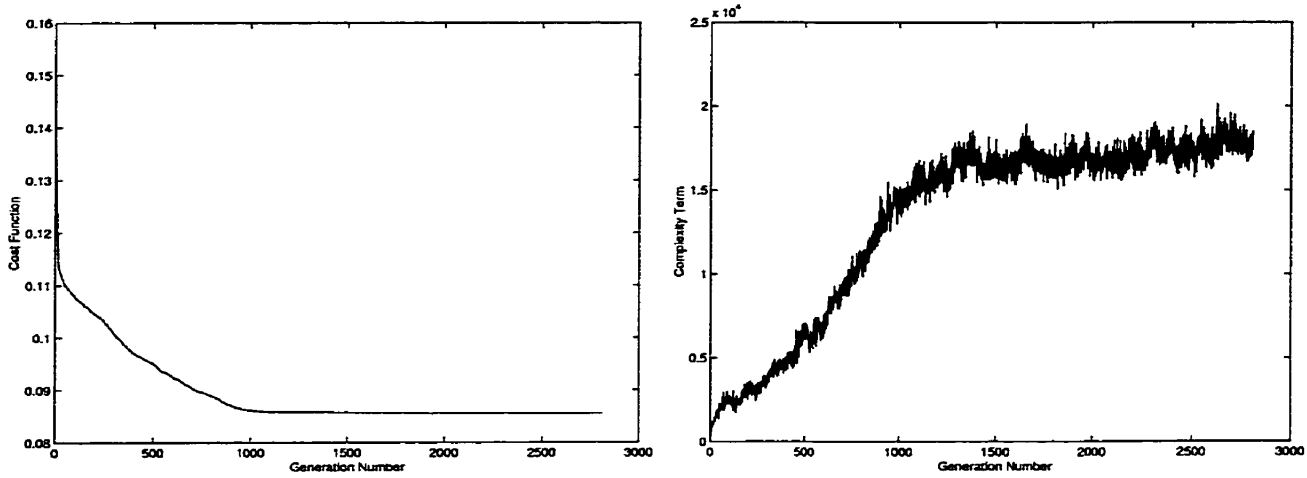

Figure 3.9: (a) Cost function (left), and (b) Complexity term (right) for Simulation 4

centers are reasonably scattered over the two-dimensional cube $[0\ 1]^2$.

3.6.4 Simulation 4

In this simulation, all the settings are exactly the same as Simulation 1, except that instead of using SNN, a family of RMQ-RBFN's with 10 neurons is used. Also, set: $\lambda = 0$, $w_a = w_b = 0.01$ and $N = 50$. Figures (3.9.a) and (3.9.b) show the cost function and complexity curves of this leaning task.

Looking at Figure (3.9.b), notice that C_{rmq} reaches a very large value throughout the training phase. Next, the model is tested against the same testing data set as in previous simulations. The cost function for the training and testing sets of data are 0.0854 and 0.0962 respectively. Again, as can be seen, the difference between the testing and training errors for simulation with $\lambda = 0$ is observable. Figure (3.10) shows how poorly the output is estimated for the testing set of data.

3.6.5 Simulation 5

In Simulation 5, all the settings are exactly the same as Simulation 4, except that $\lambda = 1 \times 10^{-6}$. Since the magnitude of the complexity term is large, the value of λ is chosen

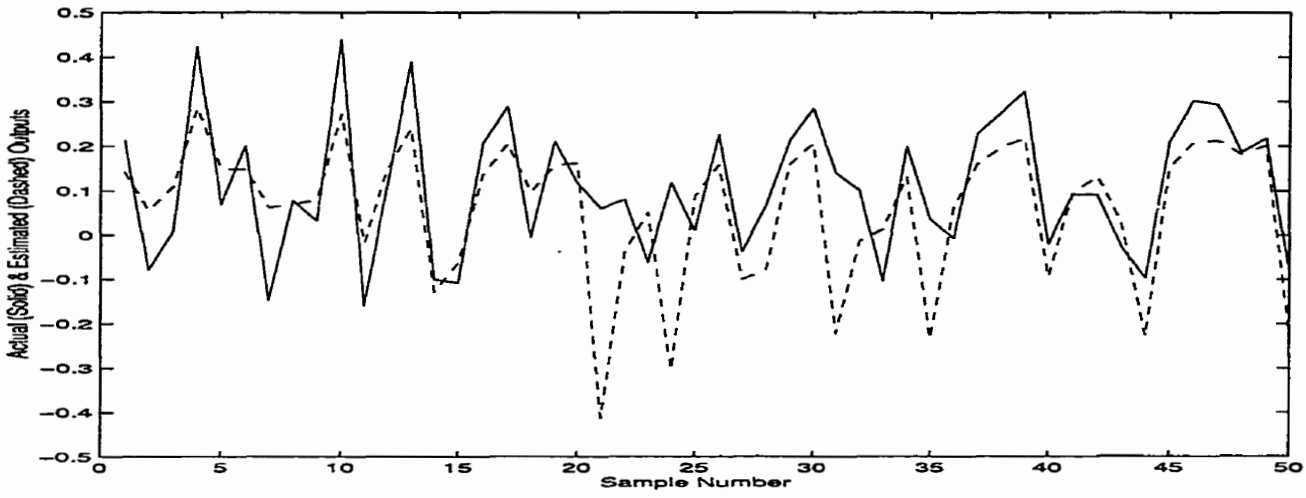

Figure 3.10: Actual (solid) and estimated (dashed) outputs for testing data (Simulation 4)

to be small so that "λC_{rmq}" remains comparable to the empirical error. A systematic practical method of choosing an appropriate value for λ is explained later in this chapter.

Figures (3.11.a) and (3.11.b) show the cost function and complexity curves of this learning task. The comparison of Figures (3.11.b) and (3.9.b) indicates that the complexity term of the network of Simulation 5 is smaller than that of Simulation 4. This suggests the theory that the difference between the testing and training cost functions for the model in Simulation 5 must be less than that of Simulation 4. The value of cost function for the training and testing set of data are 0.0881 and 0.0930 respectively, which verifies our theory. The testing performance of the resulting network for the first 50 samples is shown in Figure (3.12).

3.6.6 Simulation 6

In the next simulation, the same function set (RMQ-RBFN's) and set of training data (with 100 data points) are used, but in this simulation the evolutionary variable-structure neural modeling algorithm is applied. The algorithm assumes $l_{low} = 5$ and $\kappa = 150$. Similar to the previous simulations, the remaining examples are used for testing evaluation. From Simulation 5, one can guess that with $\lambda = 1 \times 10{-}6$, the smaller values of the empirical errors may not be achieved. As a result, in this simulation, the weighting factor is reduced to $\lambda = 1 \times 10^{-7}$, while the rest of the parameters are kept the same as the

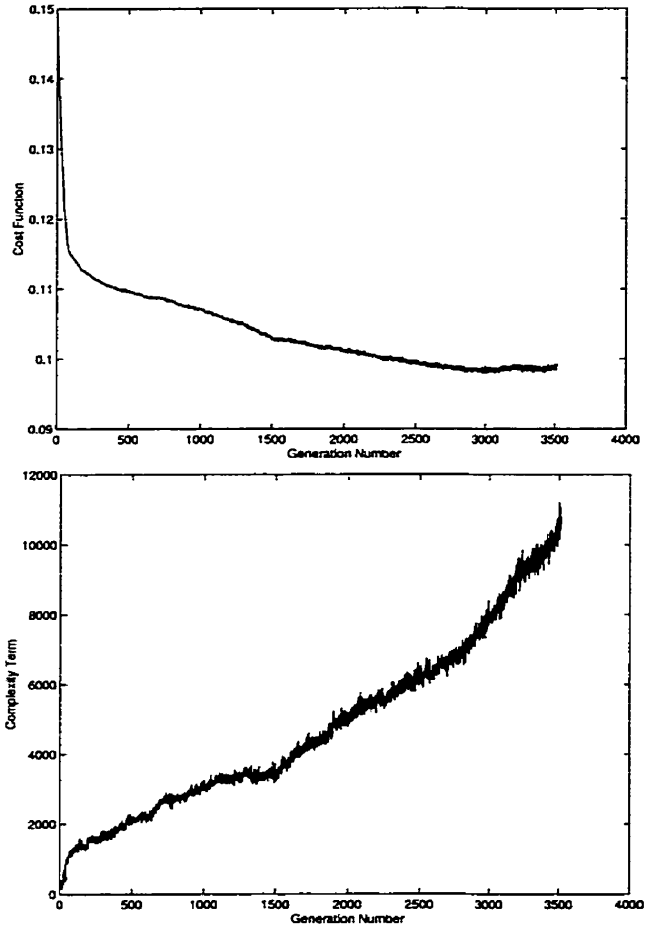

Figure 3.11: (a) Cost function (top),and (b) Complexity term (bottom) for Simulation 5

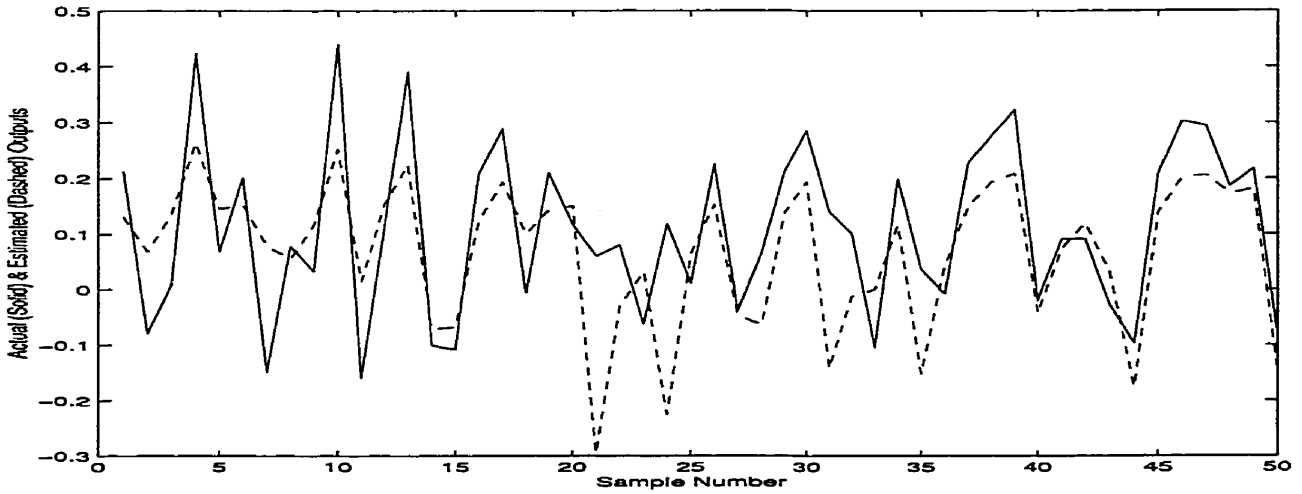

Figure 3.12: Actual (solid) and estimated (dashed) outputs for testing data (Simulation 5)

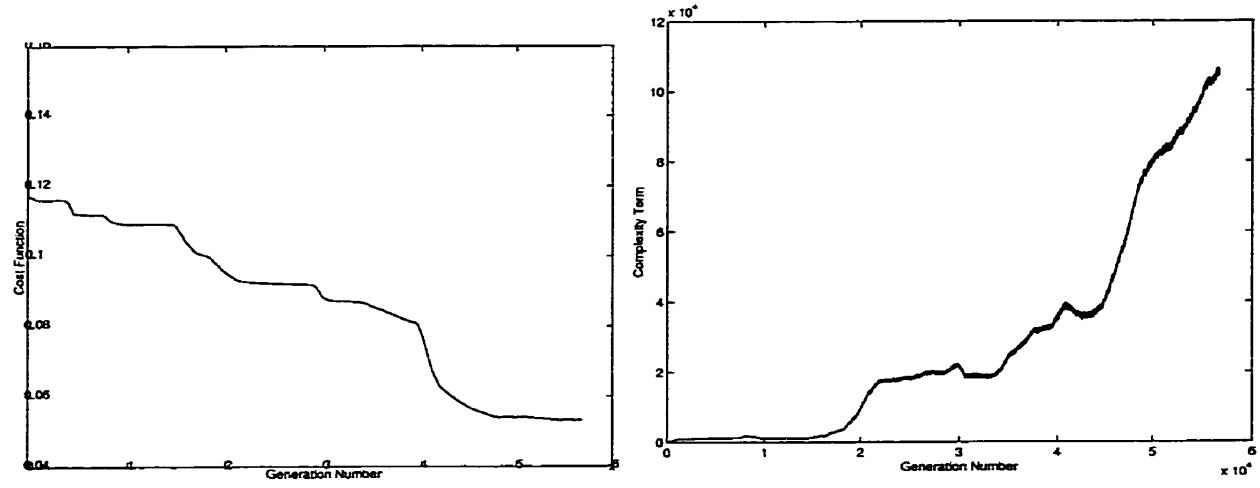

Figure 3.13: (a) Cost function (left), and (b) Complexity term (right) for Simulation 3

previous simulation. Figures (3.13.a) and (3.13.b) show the cost function and complexity curves of this algorithm, respectively.

The resulting network has 18 neurons in its hidden layer and its training cost function and empirical error are 0.0533 and 0.0428 , respectively. In the testing phase the values of the cost function and empirical error go up to 0.0628 and 0.0523 respectively, which might be still acceptable but slightly larger than the training ones. Clearly, obtaining small empirical errors (through reducing λ) has affected the difference between the training and testing errors.

Figure (3.14) shows the behaviour of the resulting model against the first 50 points of the testing set. The testing result indicate that the resulting network can be regarded as both accurate and repeatable.

3.7 Discussion

In this section, the simulations results are discussed.

1. All the simulations indicate that the EP algorithm is capable of performing optimization of non-smooth functions. Repeating any of the simulations with a different initialization as well as random numbers that are used throughout the optimization

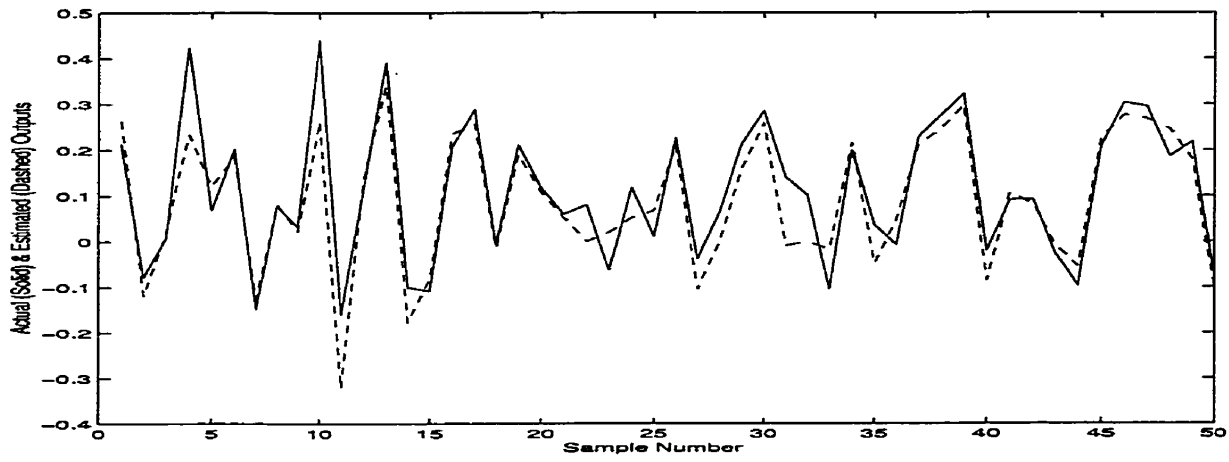

Figure 3.14: Actual (solid) and estimated (dashed) outputs for testing data (Simulation 6)

procedure results in relatively similar solutions. This demonstrates the robustness of the presented EP algorithm for the data originally generated.

2. In all simulations, the choices of N, $BackStep$, $MinSlope$, w_a, w_b and λ directly affect the resulting neural network. From the logic of the algorithm, it seems that larger values of N and $BackStep$ together with smaller values of w_a, w_b and $MinSlope$ will result in more reliable solutions. However, this requires a longer training process.

3. Parameters w_a and w_b influence the extent to which the weights of the networks are mutated. In many EP-based algorithms such parameters are annealed throughout the training process, i.e. the values of w_a and w_b are reduced as the new generations are produced. This is the main idea of "Simulated Annealing" used in many EP-based algorithms. In all simulations, constant w_a and w_b are used, but these values can be chosen to follow sequences of non-increasing numbers. Using this annealing process, the results of all simulations might be further improved. However, to obtain a smooth annealing process, appropriate sequences of w_a and w_b must be used. Normally, such appropriate sequences are found throughout a process of trial-and-error.

4. Throughout the simulations with a certain family of neural networks, it was observed that using cost functions with non-zero λ gives solutions that perform more similarly

over the training testing sets. Simulations with SNN's and RBFN's indicate that where the complexity terms are kept small (through selecting a non-zero λ), the difference between the testing and training empirical error has been significantly reduced. This supports the use of the defined complexity measures in the cost functions and indicates that the learning-based complexity terms avoid overfitting.

5. Considering the fact that the complexity terms used here are of order $\ln(\delta)$, and by looking at the values obtain for such complexity terms in our simulation, it can be observed that the bounds obtained in the previous chapter are extremely conservative. This suggests that the bounds may not be directly used unless the size of the training data set is large.

6. As to the selection of λ for a practical application of neural modeling with a fixed-structure, the following procedure can be suggested. First perform the modeling assuming $\lambda = 0$. Then, use the set of parameters obtained from this procedure as the initial weights for a new training process with small λ. If the value of the empirical error for the resulting network is still acceptable, use the parameters of the resulting network to initialize a new training process with even larger λ. Repeat this process until the resulting empirical error becomes unacceptably large. This iterative method can give us a reasonable value for λ. As to how the value of λ has to be selected for a variable-structure neural modeling, one can start with the above-mentioned method for a network with a medium size of hidden layer. Then the appropriate λ found for this network can be used throughout the variable-structure neural modeling.

7. Similar simulations can be performed for other types of neural networks discussed in the previous chapter to assess the performance of the proposed cost functions as well as the evolutionary algorithms. Due to similarities between the neural networks used for the above simulations and the remaining neural structures, one can expect to obtain similar results from a set of similar simulations with the rest of the neural structures.

8. Considering the magnitude of the complexity terms (and consequently values of

δ), it appears that either the results on SSN's are less conservative than those of RBFN's or the SNN's are indeed easier to learn. As a result, in the next chapter, SNN's are used for a real modeling application.

9. In order to test the variability of the proposed cost functions and evolutionary algorithms more thoroughly, all the above simulations have to be repeated several times for different training and testing sets as well as different sets of algorithm settings. This process, although necessary for a solid statistical evaluation, takes a tremendous amount of computation time and is avoided here.

3.8 Summary

The results of this chapter can be summarized as follows:

- The conservative bounds presented here on the sample complexity of Nonlinear FIR modeling procedures suggests that in many practical applications where the size of the training data is small, the results of the learning theory may not be used directly.
- New complexity measures of the neural models are introduced. They are based on the learning complexity of the networks, and incorporate them into the cost function to be minimized throughout the training process.
- Two algorithms based on Evolutionary Programming are introduced and used to perform the optimization process. One of the EP methods assumes a fixed structure for the neural network, while the other one searches through different structures to find an appropriate neural model.
- The simulation results indicate the relatively successful performance of both algorithms in a typical modeling procedure.

Chapter 4

Two Dimensional Sheet-Scanning System

4.1 Introduction

In sheet-forming processes, such as processes used in paper, steel and plastic sheet production, monitoring the quality of the produced sheet is an important and challenging process. In the paper industry, as the sheet of paper is being produced, quantities such as basis weight (weight per unit area), moisture content and caliper (thickness) of the paper are measured using highly sophisticated and expensive sensors. Basis weight is considered to be the most important characteristic of the paper, and has a vital role in assessing the overall performance of the paper machine. Because of the economic issues and the complexities involved with the re-starting of the paper machine, the production line is rarely shut down, even when the sensors are off-line. When a sensor failure occurs or during the normal sensor calibration, paper is produced for which no valid measurement is available. In this chapter, the neural models are used to extrapolate forward the measurements of paper basis weight before the sensor failure (or maintenance check-ups) occurs to estimate this quantity for the paper produced while the sensors are not operating. It is necessary to realize that this chapter is not meant to claim that neural networks are the best models for estimation of paper machine data. It seems that the methods that consider the physical properties of the paper machine may be more successful in modeling of the paper machine data. This is mainly because when neural networks are used, all such knowledge is simply disregarded. Here, the main objective is to illustrate that the careful use of neural networks can avoid overfitting of the data for a complicated industrial system. Therefore, the use of neural network for such a system if the physical characteristics of the machine is known may not be the best modeling approach.

This chapter is organized as follows. Section 4.2 briefly describes the main units of a paper machine and the procedure of scanning the paper basis weight. In Section 4.3, the

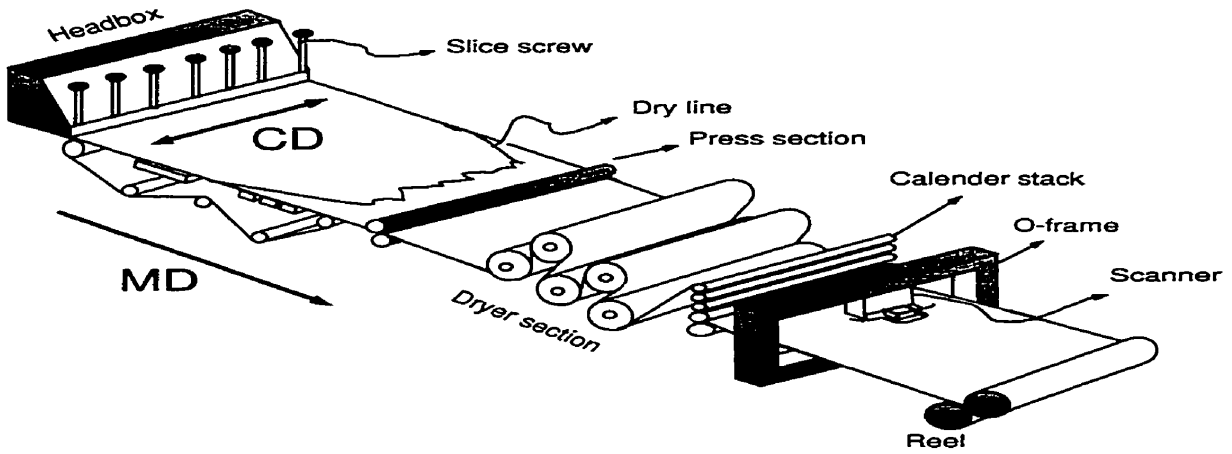

Figure 4.1: Schematic structure of a typical paper machine

results of applying the presented systematic neural modeling to model the basis weight are presented. Section 4.4 discusses the results of the chapter and is followed by the summary.

4.2 Monitoring of Paper Quality in a Paper Machine

In this section, the main components of a paper machine are described, as is the scanning mechanism of the basis weight sensors. Also some technical specifications are presented regarding our task of predicting the next weight scans.

A typical paper machine, as shown in Figure (4.1), consists of different units including the headbox, the dryer section, the press section and the reel. The headbox contains a number of actuators that control the amount of fibre delivered on to a mesh conveyer (also called the wire). The fibre mat is dried throughout the dryer section, and passes the scanner where its basis weight, moisture content (per unit square), and caliper are measured by the sensors.

A basis weight sensor is normally composed of a transmitter and a detector of β-radiation that are mounted on the two sides of a closed rectangular frame called "O-frame". For each measurement, the transmitter located above the belt sends a signal with a known power. This signal is then attenuated according to the density and texture of the paper. The power of the signal received by the detector (mounted underneath the frame) depends on the weight as well as the moisture of the paper at that particular point.

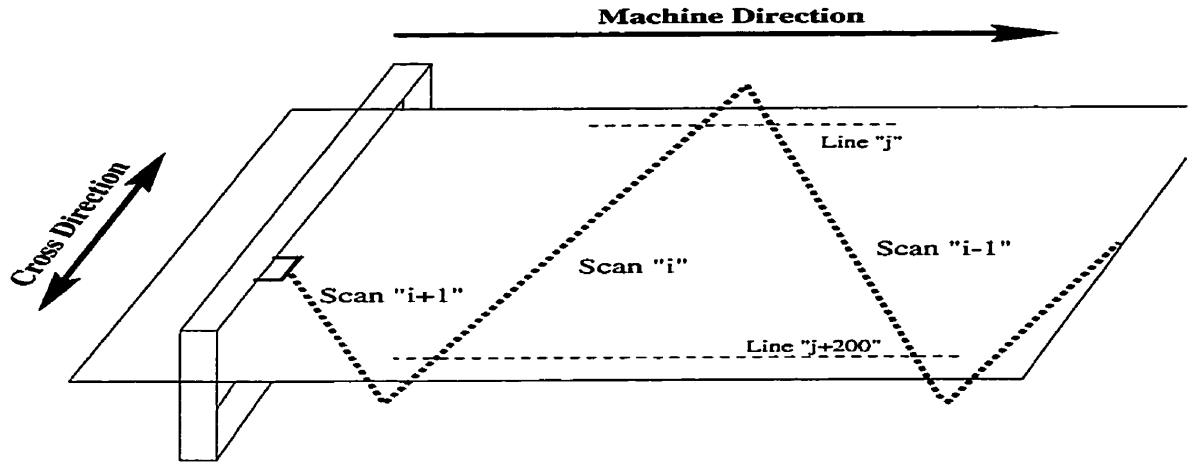

Figure 4.2: Schematic structure of the scanning unit

As shown in Figure (4.2), in a typical scanning system, the transmitter and receiver move along a rail that is mounted on the frame. As shown in Figure (4.2), as the paper moves in the machine direction (MD), the sensor scans the paper in a zigzag pattern. In each sweep of the paper, a number of measurements is taken by the sensors, which form one row of a measurements matrix M. Hereafter, each scan of the sheet is referred to as a row (in the measurement matrix).

Now, consider two consecutive rows "$i-1$" and "i" in the measurement matrix M. In the notation used in the following paragraphs, $M(i,j)$ represents the basis weights of measured at row "i" and measurement point "j" (along jth line in machine direction). From the zigzag nature of the scan, one can notice that the dependency between the values $M(i,j)$ and $M(i+1,j)$ depends on j. Consider the correlation between $M(i-1,j)$ and $M(i,j)$, and compare that with the correlation between $M(i-1,j+200)$ and $M(i,j+200)$, as shown in Figure (4.2). The correlation between the first pair ($M(i-1,j)$ and $M(i,j)$) is expected to be greater than that of the second pair, as the points on the first pair are separated by a smaller distance. Now, compare the correlation between $M(i,j)$ and $M(i+1,j)$, with the correlation between $M(i,j+200)$ and $M(i+1,j+200)$. As can be imagined, this time the correlation between the second pair will be larger.

In the modeling of the scanning system, one has to take the above problem into consideration. However, since each scan takes about 30 seconds, in practice, two consecutive scans (whether they are separated by only a fraction of a second or by 32 seconds) are

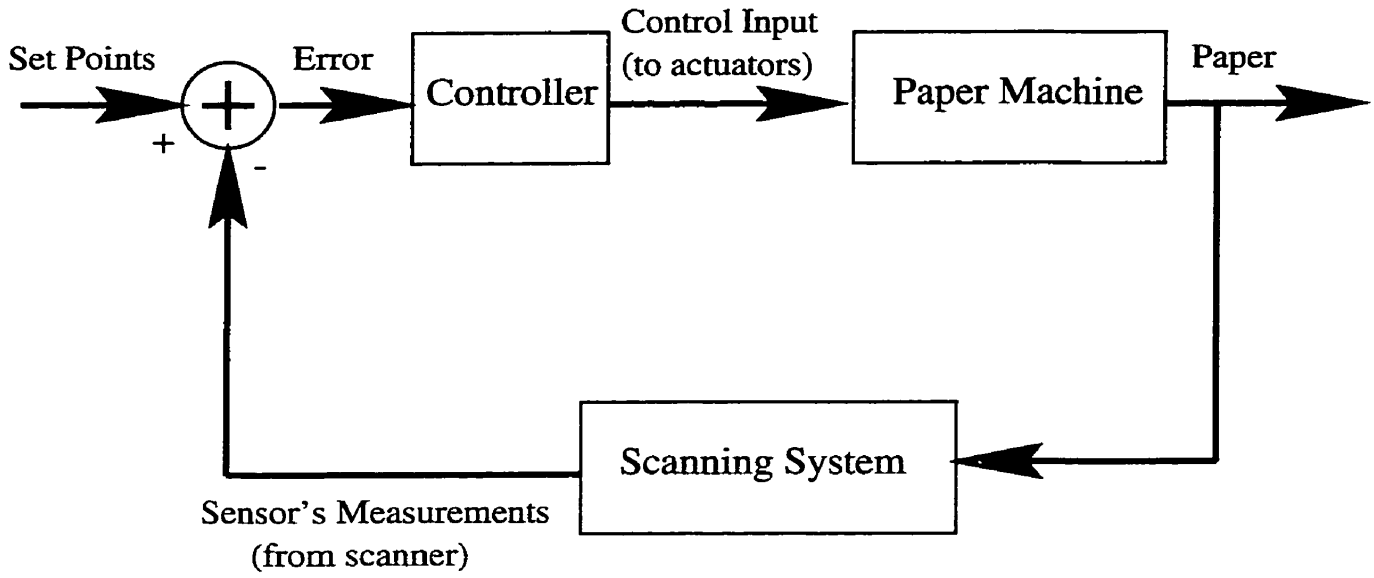

Figure 4.3: Schematic control structure for a typical paper machine

treated the same. As a result, in this research, this issue is ignored and different rows of the measurement matrix are treated similarly.

A paper machine normally operates in a closed loop, i.e. there exist separate control loops for each of the above mentioned quantities that control the variations of the corresponding quantity. In such a control system, as shown in Figure (4.3), the error between the sensor measurements (such as basis weight) and the set points are fed to a controller which generates appropriate control actions for the actuators so as to eliminate unwanted variations in the paper weight and moisture.

If the sensors are taken off-line, the controller often stops generating new control input, and the activation of all actuators is set to some average value. While the sensors and controller are not operational, the system works in open loop. This period of open loop operation for a typical task of calibration might take up to a few minutes. Meanwhile, due to the high speed of the paper machine (up to 150 km/hour), a huge amount of paper (up to 100 meters of paper with the width of 7 meters) is produced. Since no measurements are taken while the system operates in open loop, it is worthwhile to predict the behaviour of the system during this period.

4.3 Neural Modeling of Basis Weight

The scans prior to the stoppage of the scanning system are used to predict the basis weights of the paper produced for up to five scans afterwards. The data set used here is taken from a mill and was provided by Honeywell-Measurex Devron Inc.

The target value of the basis weight for the paper being produced is 300 grams per square meter (gsm). The sheet is 7 meters wide and its weight is measured at 240 locations across the sheet. Some of the measurements on both edges are discarded, because they represent sensor readings while the sensor has passed the edges of the paper (see Figure (4.2)). Omitting the first 7 sensor readings on each edge leaves us with a matrix of measurements with 226 columns. The scanner completes an entire scan in 32 seconds. According to the engineers in the mill, most of the error variation cannot be eliminated by controlling the actuators and therefore the system essentially runs in open loop. As a result, the activation of the actuators (the control input) is treated as a source of disturbance and so is excluded from our modeling. Moreover, during the estimation time, the control input is assumed to be a constant signal. Excluding the control input from the estimation process enables us to train the model in open loop (while the sensors are on) and perform the prediction when the sensors are off-line, without having to be concerned about the change in the way the control input is generated. The available data set consists of 50 scans (rows), which form a measurement matrix of the size $M(i,j)_{50\times 227}$.

The objective of the neural modeling is to estimate the next scans (rows) in the machine direction from the previous measured scan in the cross direction. More specifically, $M(i+k,j)$ is to be estimated using the values of at $M(i,j-1)$, $M(i,j)$ and $M(i,j+1)$, where $1 \leq k \leq 5$. The input vector $(M(i,j-1),\ M(i,j),\ M(i,j+1))$ is formed along the cross direction and for a fixed i forms a sequence of m-dependent r.v.s with $m = 3$. The correlation between the reading along the cross direction is assumed to be negligible beyond three points. The extension to higher assumed ranges of dependency is straightforward, and it can be expected to result in better estimation performances at the expense of increased calculations.

In order to see how the training and testing data are generated, consider a fixed k.

Rows i =1, 7, 13, 19, 25, 31, and 37 form the input, i.e. for each i, set of vectors $(M(i,j-1),\ M(i,j),\ M(i,j+1))$ is formed, where $2 \leq j \leq 225$. The corresponding output for each of the above vectors is $M(i+k,j)$. The training set is thus formed as the input-output data points generated by rows 1, 7, 13, and 19 as the input rows. The remaining data points will be used as the testing set. Since $1 \leq k \leq 5$, there will be five training sets and five testing sets to be processed separately.

Specific rows are used in order to separate the inputs and the outputs completely. Notice that in a real scenario where the sensors are taken off-line, the available information up to row $i = i_0$ is used to estimate the next five rows. This means that the prediction is made for only rows $i_0 + 1$, ..., $i_0 + 5$, i.e. the model is not allowed to use our estimated values as the inputs to estimate the rows beyond five rows. These rules in generating the data guarantee that the model is actually an FIR one and makes sure that no measurement has been used both as input and output information.

Having generated the training and testing data sets for each value of k, the evolutionary variable-structure neural modeling is applied to the corresponding training set. The parameters of the algorithm for all modeling tasks are chosen as follows: $N = 50$, $w_a = w_b = 0.001$, $\lambda = 0.0001$, $l_{low} = 2$, $\kappa = 100$, $\epsilon = 0.05$, $MinSlope = -1 \times 10^{-8}$, $BackStep = 1000$. In order to follow the exact structure of the theoretical results of the previous chapters and have a better insight to the level of error, the input variable is normalized to the interval of $[0\ 1]^3$, and the output variable is mapped to the interval of $[-\frac{1}{2}\ \frac{1}{2}]$.

4.3.1 Simulation 1: One-Row-Ahead Prediction

For $k = 1$, the resulting network has 6 hidden neurons in its structure. The training cost function and the empirical error are 0.0291 and 0.0238, respectively. Figure (4.4) depicts the evolution of the cost function and complexity term of the best network of each generation. As can be seen, the resulting complexity term C_{atan} levels off at 53.1376.

Next, this network is tested against the testing set which consists of rows 25, 31 and 37 as the inputs and rows 26, 32, and 38 as the outputs. The actual and estimated basis weights for row 26 (i.e. actual and estimated profiles of basis weight across the sheet) are

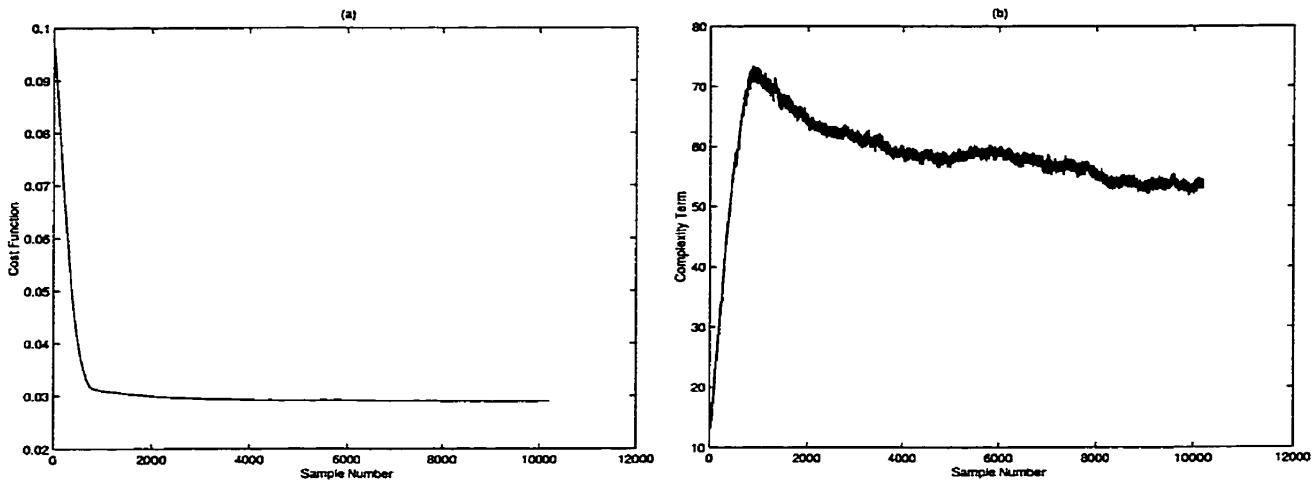

Figure 4.4: (a) Cost function (left), and (b) Complexity term (right) for Simulation 1

shown in Figure (4.5.a). In order to see the prediction quality more clearly, the estimate and actucal basis weights of almost 57 points located in the middle of the profile are shown Fgigure (4.5.b). Similar profiles are obtained for estimation of rows 32 and 38, which indicate that the estimation process can accurately predict the rows ahead. The testing empirical error (calculated over all estimated rows) is 0.0244, close to the training value. Also, notice that the estimated and the actual profiles are highly similar and the estimation process seems to be accurate.

In order to see how neural networks help the prediction, here the results are compared with a rather trivial average modeling, in which $M(i+k,j)$ is estimated as the simple linear average of $M(i,j-1)$, $M(i,j)$, and $M(i,j+1)$, where $k=1$. For this model, the value of the empirical error for the testing data is 0.0307 which is considerably larger than that of the neural model. This shows that the neural model of Simulation 1 is not only reliable but also accurate (compared to a simple linear estimation).

The variability of the model can be tested by considering the standard deviation of the testing prediction error (i.e. the actual output minus the estimated output) over different estimated rows. The calculated values of standard deviation for the prediction error of rows 26, 32 and 38 is 0.0326, 0.0361 and 0.0326 respectively. The close values of standard deviation for different rows indicates the similar performance of the algorithm over different rows of the data.

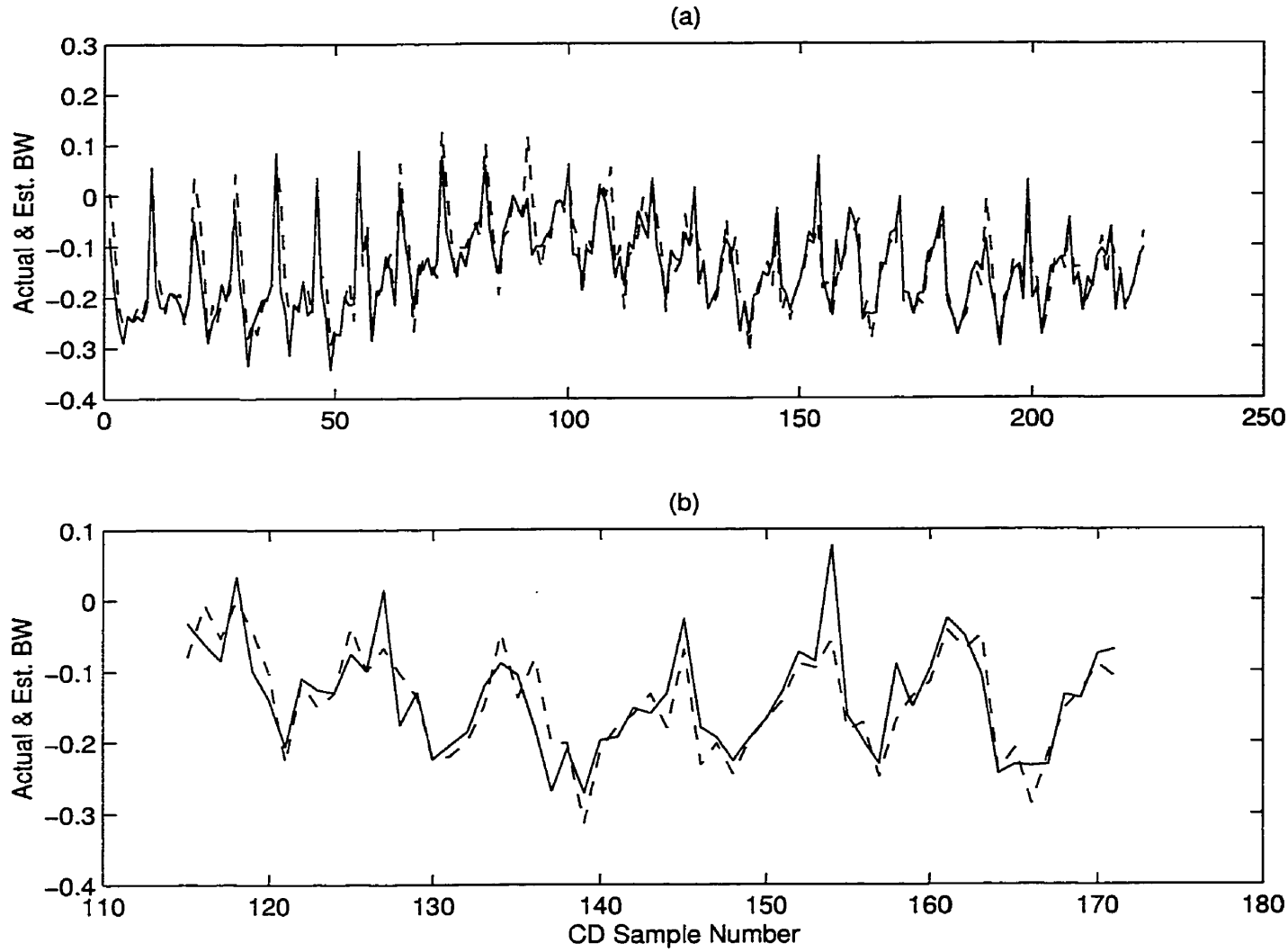

Figure 4.5: (a) Actual (solid) and estimated (dashed) normalized output across row 26 (the entire profile) for Simulation 1 (b) Actual (solid) and estimated (dashed) normalized output across row 26 (the middle portion of the profile) for Simulation 1

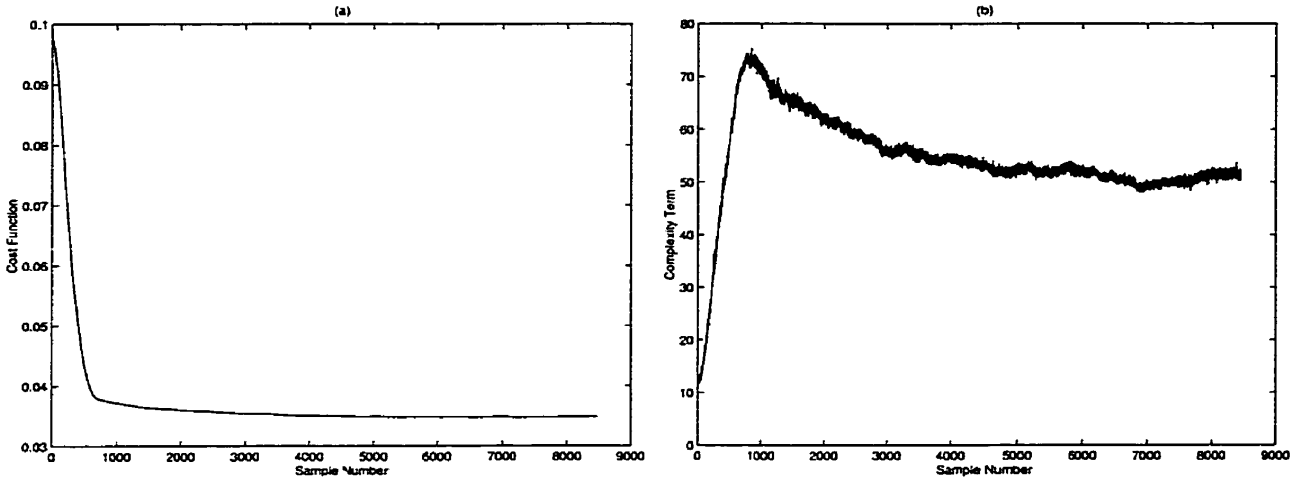

Figure 4.6: (a) Cost function (left), and (b) Complexity term (right) for Simulation 2

4.3.2 Simulation 2: Two-Row-Ahead Prediction

In this simulation, all the settings are the same as those of Simulation 1, except that $k = 2$, i.e. a two-row-ahead estimation is performed. The resulting network has 4 hidden neurons in its structure. The training cost function and the empirical error are 0.0350 and 0.0299, respectively. Figure (4.6) shows the curves of the cost function and complexity term. The resulting complexity term C_{atan} is 50.9978.

In testing as with the previous simulation, rows 25, 31 and 37 are used to form the inputs, while rows 27, 33, and 39 now form the outputs. The actual and estimated profiles for row 27 are shown in Figure (4.7). The resulting testing empirical error (calculated over all estimated rows) is 0.0298 that is smaller than the training one. Also notice that the estimated and the actual profiles are still very similar and the estimation process seems to be acceptable. In this case, the empirical error of the trivial estimation on the testing data is 0.0431 which is significantly larger than the error of the neural model.

The calculated values of standard deviation for the prediction error of rows 27, 33 and 39 are 0.0395, 0.0422 and 0.0432 respectively. The closeness of the above values again shows the small variability of the algorithm over different parts of the data.

The three-row-ahead and four-row-ahead estimations result in networks with 6 and 7 neurons, respectively. The main results of these simulations are given in Table (4.1).

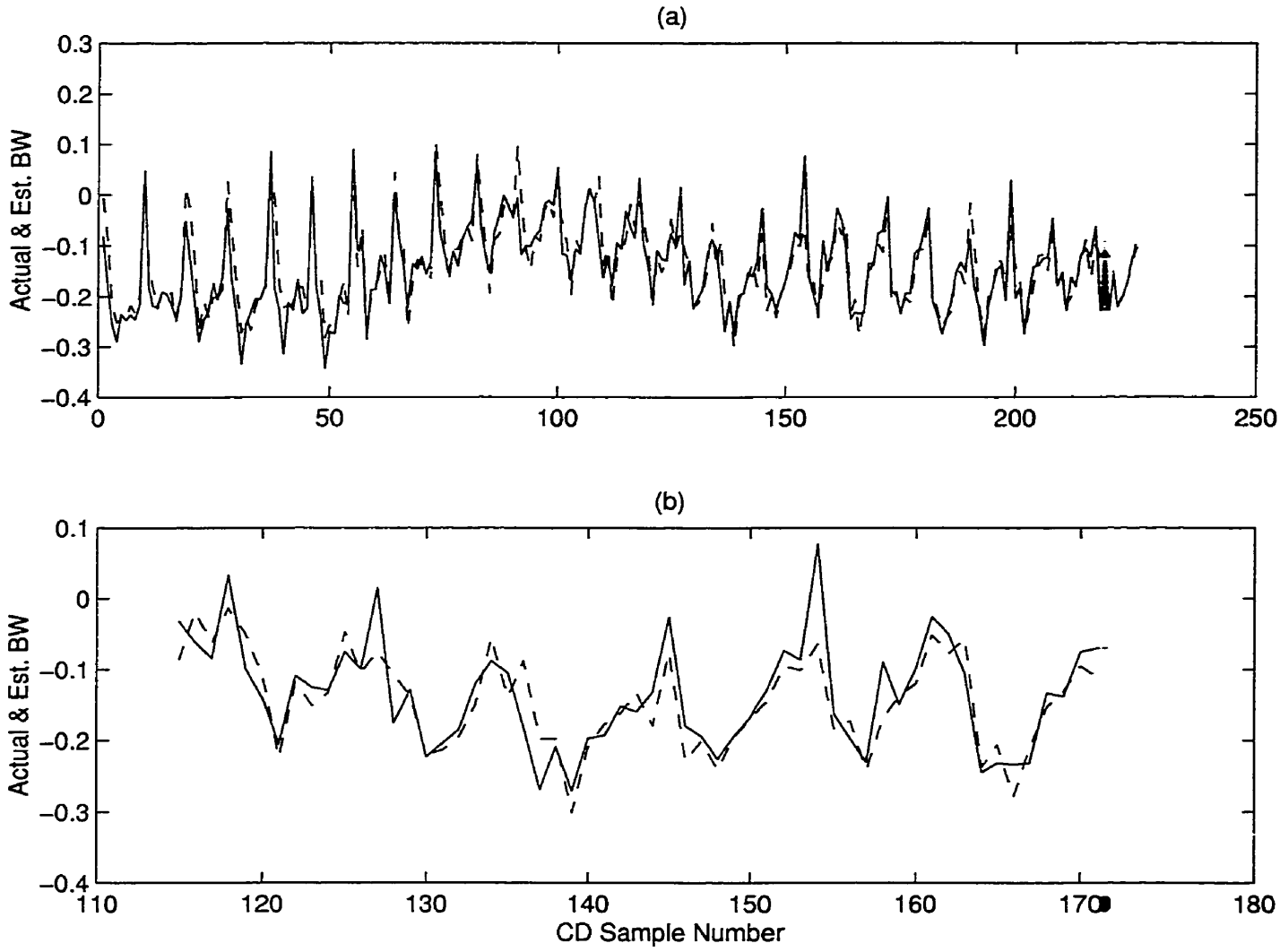

Figure 4.7: (a) Actual (solid) and estimated (dashed) normalized output across row 27 (the entire profile) for Simulation 2 (b) Actual (solid) and estimated (dashed) normalized output across row 27 (the middle portion of the profile) for Simulation 2

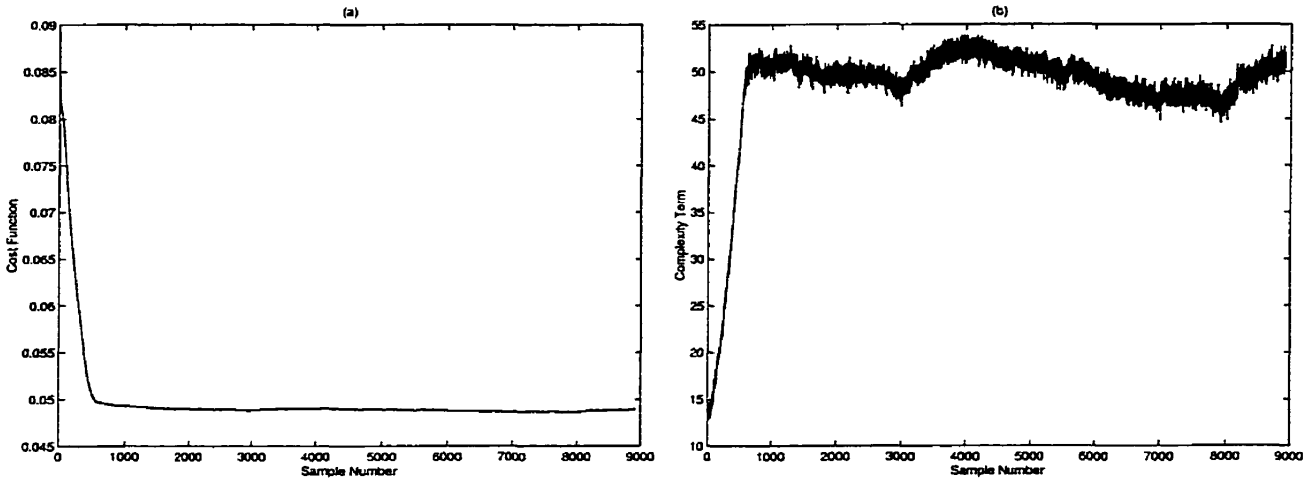

Figure 4.8: (a) Cost function (left), and (b) Complexity term (right) for Simulation 3

4.3.3 Simulation 3: Five-Row-Ahead Prediction

With all the settings the same as those of the previous simulations, a five-row-ahead estimation is now performed. Here, the rows 1, 7, and 13 are used to form the inputs, and rows 6, 12, and 18 to form the outputs. The resulting network has 9 hidden neurons in its structure. The training cost function and the empirical error are 0.0464 and 0.0413, respectively. Figure (4.8) shows the curves of the cost function and complexity term. The resulting complexity term C_{atan} reaches to 51.0522.

In testing as with the previous simulations, rows 25, 31 and 37 are used to form the inputs, and rows 30, 36, and 42 to form the outputs. The actual and estimated profiles for row 30 are shown in Figure (4.9). The resulting testing empirical error is 0.0441 and relatively close to the training value. The error of the trivial linear estimation for five-row-ahead prediction is 0.0571 . The calculated values of standard deviation for the prediction error of rows 30, 36 and 42 is 0.0578, 0.0576 and 0.0583 respectively. The closeness of the above values further indicates the small variability of the algorithm over different parts of the paper machine data.

Each of the above simulations takes about 210 minutes on a machine with 400 MHz Pentium II processor.

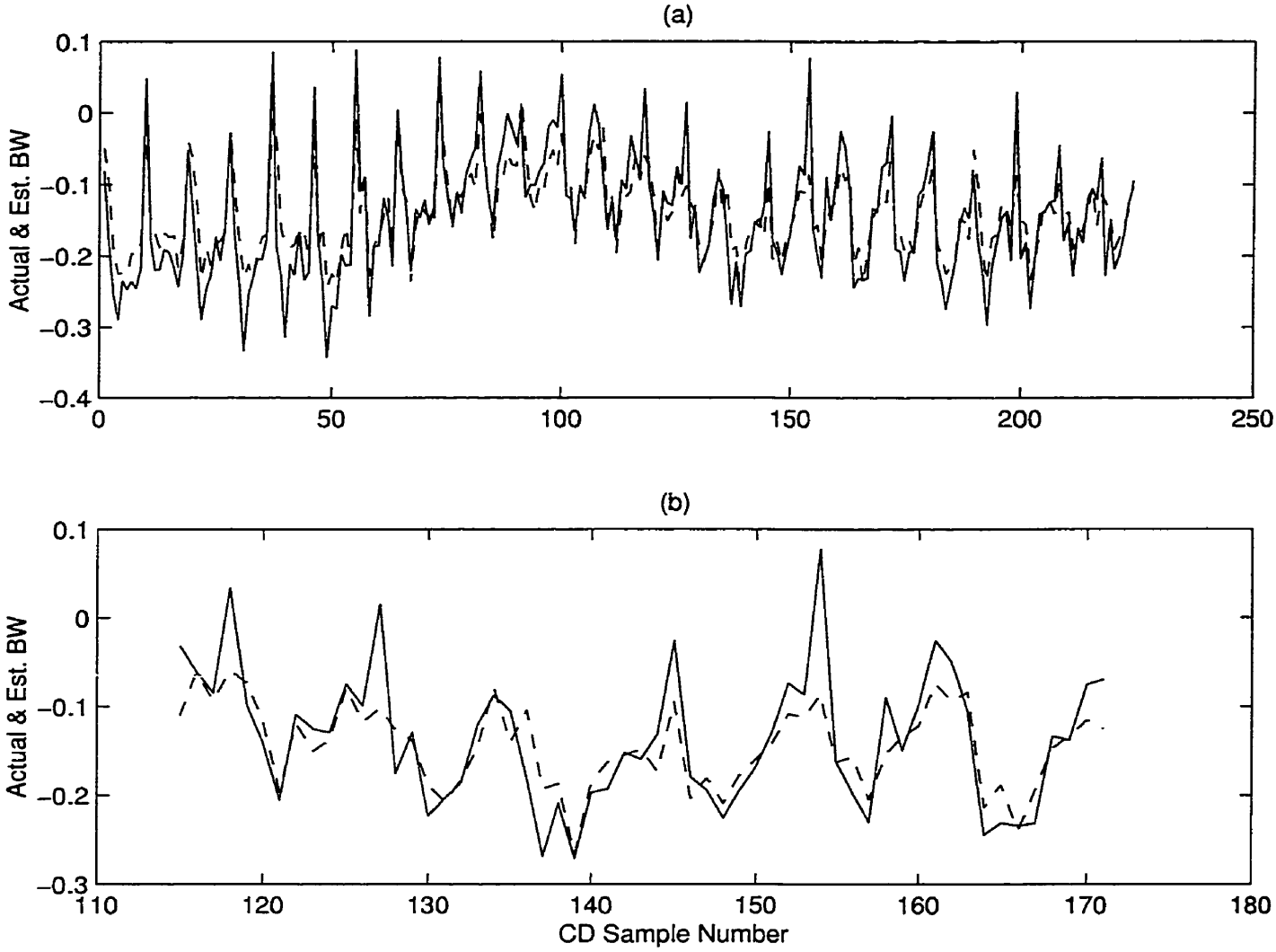

Figure 4.9: (a) Actual (solid) and estimated (dashed) normalized output across row 30 (the entire profile) for Simulation 5 (b) Actual (solid) and estimated (dashed) normalized output across row 30 (the middle portion of the profile) for Simulation 5

Prediction	Train. Cost	Train. Err.	Test. Err.	Comp. Term.	Num. of Neurons
1-Row-Ahead	0.0291	0.0238	0.0244	53.14	6
2-Row-Ahead	0.0350	0.0299	0.0298	50.10	4
3-Row-Ahead	0.0406	0.0358	0.0361	47.90	6
4-Row-Ahead	0.0470	0.0423	0.0432	47.14	7
5-Row-Ahead	0.0464	0.0413	0.0441	51.05	9

Table 4.1: Training cost function, training empirical error, testing empirical error and complexity term for one-row-ahead, two-row-ahead, three-row-ahead, four-row-ahead and five-row-ahead predictions.

4.4 Discussion

Here, the results obtained in this chapter are discussed.

1. From Table (4.1), it is seen that the quality of prediction deteriorates as further rows ahead are estimated. This may be explained by the fact that the outputs in closer rows are more highly correlated with the inputs and is thus to be expected.

2. The results obtained from different simulations and shown in Table (4.1) indicate that the number of hidden neurons increases with the prediction interval. This shows that the prediction of further rows involves a more difficult estimation process and requires more neurons.

3. Performing the estimation with different algorithm settings (such as λ, w_a, w_b, and N) may result in different neural models. However, it seems that different simulations with a fixed λ result in models with relatively similar cost functions and complexity terms.

4. The estimation process may be continued for the prediction of rows further than five rows; however the quality of the estimation can be expected to deteriorate. Since the predicted values might be the only information available on the produced paper, extending this estimation process even with a lower prediction quality may be useful.

5. Once a neural model for the system is generated off-line, one can update the model through an adaptive process. In order to do so, some of the old training points must be replaced with the newly measured data and the training process may be kept running throughout the entire modeling task. Since each scan takes almost 32 seconds, updating the model after every few scans seems to be feasible, specially when fast computers are available. Since the paper machine is a time-variant process (especially after a machine start-up), the adaptive approach is desirable.

6. The accuracy of the estimation process is expected to improve if the input vectors are formed of the information from the last two rows rather than merely the last

row as in the present formulation. The price to pay, however, is an increase in the computation time.

7. In order to have more confidence over the simulation results, one may want to repeat the simulations using different sets of training and test data as well as different algorithm settings. Due to the fact that each run of the algorithm takes a considerable amount of time, repeating the simulations may be a more feasible task once faster machines are available.

4.5 Summary

The chapter can be summarized as follows.

- The above simulation results indicate that the proposed evolutionary algorithms are capable of performing optimization in nonlinear and noisy real applications and generate useful estimates.
- The relatively close training and testing performances indicates that proposed complexity terms can successfully deal with noisy complex systems.
- While the sensors in the paper machine are taken off-line, it is possible to use neural networks to estimate the properties of the produced paper based on the previously recorded data.

Chapter 5

ARX Models: Stability and Learning

5.1 Introduction

In a dynamic modeling task in the presence of additive noise, the output of a system is expressed as a function of the history of the input as well as the output. In the case of a nonlinear ARX (also known as NARX), assuming that u_{t-q+1}, u_{t-q+2}, $\ldots$ u_{t-d} describe the history of the input variable and y_{t-k}, y_{t-k+1}, $\ldots$ y_{t-1} that of the output:

$$y_t = f(y_{t-k},\ y_{t-k+1},\ \cdots\ y_{t-1},\ u_{t-q+1},\ u_{t-q+2},\ \cdots\ u_{t-d}) + \zeta_t \tag{5.1}$$

where d, $q-d-1$, k and ζ_t represent the degree of the input, the delay from the input to the output, the degree of the output and the additive noise on the system, respectively. Although the model can represent multi-dimensional models, here the single-input/single-output (SISO) case is considered. It is also assumed that u_t and ζ_t are uncorrelated sequences of independently and identically distributed (i.i.d.) random variables. The Markov process formed as (5.1) includes a wide range of dynamic models used in engineering applications including dynamic neural networks. One of the most important properties of a NARX model to be investigated is the stochastic stability of the model. Stochastic stability not only guarantees the issue of boundedness of the output for bounded inputs, but also establishes the necessary conditions for any definition of learning for ARX models (as described later). The concept of stochastic stability has been addressed in the literature assuming different definitions for stochastic stability, resulting in different sufficient stability conditions for NARX models. The concept of Lagrange stability [26] defines a notion of stability based on a Lyapounov function defined in terms of the process. Kushner's work on stochastic stability [24],[23] has provided a more com-

prehensive mathematical framework for testing stochastic stability of discrete systems. Important results in this field come from the relation between the stochastic Lyapounov stability (as in Kushner's work) and the concept of geometric ergodicity [43], [22]. The results of this line of research not only provide simple practical notions of stochastic stability, but also create a foundation for assessment of other statistical properties such as the learning properties of dynamic models [39]. In all learning paradigms presented for dynamic models, the assumption of processes being geometrically ergodic is treated as a fundamental requirement, i.e. learning properties of dynamic models can not be evaluated unless the assumption of models being geometrically ergodic is verified [30], [11], [42]. This further calls for evaluation of geometric ergodicity for important families of nonlinear dynamic models. Here, the general results of [43] are applied to the special case of neural modeling with sigmoid neural networks and specific sufficient conditions under which the model is geometrically ergodic are presented.

Here, first a set of sufficient conditions for geometric ergodicity of SNN's is obtained. Then, the results are used to assess the learning properties of neural ARX models. In order to do so, first, learning theory is extended to learning with strong mixing data. Then, specific upper bounds on the sample complexity of such models are given. Next, the learning properties of neural ARX are applied to define complexity measures (along with their corresponding cost functions) that can be used in practical applications. Finally, using the evolutionary neural modeling algorithms introduced in Chapter 3. The performance of such cost functions is evaluated in a number of simulations.

This chapter is organized as follows. Section 5.2 gives the basic definitions of stochastic stability, as well as the existing results on the geometric ergodicity of a general family of NARX models. Section 5.3 contains a set of sufficient conditions over the parameters of a sigmoid neural network, which guarantees the stochastic stability of the model. The main results of this section are given in Theorems (5.3.1) and (5.3.2). In section 5.4, PAC learning theory extended to learning with α-mixing data. In the same section, the resulting learning theory is applied to SNN's and the sample complexity of such learning tasks are bounded. Theorem (5.4.2) acts as the main theorem of this section. In Section 5.5 a distribution-free complexity measure of SNN's is introduced. Theorem

(5.5.1) contains the main results of this section. Section 5.6 uses the results of the previous section to describe learning-based algorithms that search for neural models with minimum complexity. In Section 5.7, the simulation results of applying the proposed algorithms on a Continuously-Stirred Tank Reactor (CSTR) are presented. The results of the chapter are discussed in Section 5.8, and finally, Section 5.9 summarizes the chapter.

5.2 Basic Definitions of Stochastic Stability

In this section, some of the basic concepts of geometric ergodicity as well as the existing results on the stochastic stability of NARX models are reviewed. Consider an integer "t" and let X_t be a Markov chain with the state space (R^p, B), where B is the Borel σ-algebra. The t-step-ahead transition probability of X_t is denoted by $P^t(x, A)$, i.e. :

$$P^t(x, A) = P(X_t \in A | X_0 = x),\ x \in R^p,\ A \in B. \tag{5.2}$$

Now, the concept of geometric ergodicity is defined as follows.

Definition 5.2.1 *X_t is geometrically ergodic if there exists a probability measure π on (R^p, B), a positive constant $\rho < 1$, and a π-integrable non-negative measurable function ϖ such that for any t:*

$$\|P^t(x, .) - \pi(.)\|_{Var} \leq \rho^t \varpi(x),\ x \in R^p \tag{5.3}$$

where $\|.\|_{Var}$ denotes the total variation norm.

Definition (5.2.1) shows that geometric ergodicity is closely related to stability. According to (5.3), in a geometrically ergodic process, the transition probability approaches a (possibly unknown) well-behaved probability measure π geometrically fast.

Here the definition of stochastic stability is given:

Definition 5.2.2 *Consider a Markov chain X_t as described above. The process is called stochastically stable iff there exists a non-negative and measurable function V (called a Lyapounov function), and constants $c > 0$ and $0 < \rho < 1$ such that:*

$$E(V(X_{t+1})|X_t = x) \leq \rho V(x) - c\,. \tag{5.4}$$

The concept of stochastic stability is also referred to as "stochastic Lyapounov stability". The Lyapounov function V and the way the stability condition is defined show why the name stochastic Lyapounov stability seems to be a more appropriate name for this concept.

The following theorem by Mokkadem [5] is known to be the most general result on geometric ergodicity of Markov processes and the way this property is related to stochastic stability.

Theorem 5.2.1 *(Mokkadem [5]) Suppose a Markov chain X_t is stochastically stable as described in Definition (5.2.2). Then X_t is a geometrically ergodic process.*

Theorem (5.2.1) shows how geometric ergodicity relates to the concept of stochastic stability. Now, the notion of "α-mixing" (also known as strong mixing) is defined. This concept describes a type of stationary random process with exponentially weakening dependency.

Definition 5.2.3 *Let $\{y_t\}_{t=-\infty}^{\infty}$ be a stationary process. For $-\infty < t < \infty$, let $\mathcal{Y}_t^{\infty}$ and $\mathcal{Y}_{-\infty}^{t}$ denote the σ-algebras of events generated by random variables $\{y_i, t \geq i\}$ and $\{y_i, t \leq i\}$, respectively. Define the strong mixing coefficient $\alpha_y(t)$ as:*

$$\alpha_y(t) = \sup_{A \in \mathcal{Y}_{-\infty}^{0},\, B \in \mathcal{Y}_{t}^{\infty}} |\Pr(A \cap B) - \Pr(A)\Pr(B)| \,. \tag{5.5}$$

Then $\{y_t\}_{t=-\infty}^{\infty}$ *is called* α*-mixing (strongly mixing), if:*

$$\lim_{t \to \infty} \alpha_y(t) = 0 \,. \tag{5.6}$$

Moreover, suppose $\alpha_y(t)$ *approaches zero geometrically fast in* t*, i.e., there exist:*

$$k_1,\ k_2,\ k_3 > 0$$

such that:

$$\alpha_y(t) = k_1 e^{-k_3 t^{k_2}} \tag{5.7}$$

Then, the process is called geometrically α*-mixing.*

Next, the focus is given to the existing results on the Markov process of form (5.1). Here, a theorem by Doukham [43], which presents a set of sufficient conditions for geometric ergodicity of the process (5.1) is reviewed.

Theorem 5.2.2 *[Doukham [43]] Consider the process (5.1). Let:*

$$X_t = (y_{t-k},\ y_{t-k+1},\ \cdots\ y_{t-1},\ u_{t-q+1},\ u_{t-q+2},\ \cdots\ u_{t-d}) \,. \tag{5.8}$$

Assume that $X_{t,i}$ *indicates the* i*th element of* X_t*. Also, assume the followings.*

1. *There exist a number* $x_0 > 0$ *and non-negative constants* $\psi_1,\ \ldots,\ \psi_k$*, a locally bounded measurable function* $h : R \to R^+$*, and a positive constant* c *such that:*

$sup_{\|X_t\|\leq x_0}|f(X_t)| < \infty$ (where $\|X_t\|$ is the Euclidean norm of X_t), and:

$$|f(X_t)| \leq \sum_{j=1}^{k} \psi_j |X_{t,j}| + \sum_{j=k+1}^{q-d+k} h(X_{t,j}) - c \tag{5.9}$$

if $\|X_t\| > x_0$.

2. $E[|\zeta_1|] + (q-d)E[h(u_1)] < c < \infty$.

Then, if the unique non-negative real zero of the "characteristic polynomial" $P(z) = z^k - \psi_1 z^{k-1} - \ldots - \psi_k$ is smaller than one, the process (5.1) is geometrically ergodic. Moreover, if the process X is stationary, the process "y" (i.e. $\{y_t\}_{t=-\infty}^{\infty}$) is geometrically α-mixing.

Although the details of the long proof presented for this theorem (by Doukham) are not repeated here, a brief description of the general scheme of the proof will be given. The proof starts with introducing a Lyapounov function of the form:

$$V(X_t) = \sum_{j=1}^{k} \alpha_j |X_{t,j}| + \sum_{j=k+1}^{q-d+k} \beta_j h(X_{t,j}) \ . \tag{5.10}$$

Then the appropriate choices of α_j's and β_j's to satisfy (5.4) are investigated. It is then proved that if all the assumptions made in the theorem hold, the condition set on the zeros of the characteristic polynomial $P(z)$ guarantees geometric ergodicity. As can be seen, the sufficient conditions set in Theorem (5.2.2) guarantee the stochastic Lyapounov stability as well as the geometric erogicity of the model.

5.3 Geometric Ergodicity of Sigmoid Neural Networks

This section starts with the following lemma about the atan sigmoid neural networks.

Lemma 5.3.1 *Suppose $x \in R^p$. Consider a family of sigmoid neural networks $\mathcal{F}$ with members as follows:*

$$f(x) = \sum_{i=1}^{l} a_i \sigma(b_i x)$$

where: $\sigma(.) = \frac{2}{\pi}\tan^{-1}(.)$ *is a smooth sigmoid activation function,* l *indicates the number of neurons,* a_i*'s* ($a_i \in R$) *are the weights of the output layer and the* p*-dimensional vectors* b_i*'s defined as:* $b_i = (b_{i1}, \ldots, b_{ip})$ *represent the weights of the hidden layer. Then:*

$$|f(x)| \leq \sum_{j=1}^{p} \left(\sum_{i=1}^{l} \frac{2}{\pi} |a_i||b_{ij}| \right) |x_j| \,. \tag{5.11}$$

Proof: From the definition of atan sigmoid neural networks:

$$\begin{aligned}
|f(x)| &= \frac{2}{\pi} |\sum_{i=1}^{l} a_i \tan^{-1}(b_i x)| \\
&\leq \frac{2}{\pi} \sum_{i=1}^{l} |a_i| |\tan^{-1}(b_i x)| \\
&\leq \frac{2}{\pi} \sum_{i=1}^{l} |a_i||b_i x| \\
&= \frac{2}{\pi} \sum_{i=1}^{l} |a_i| |\sum_{j=1}^{p} b_{ij} x_j| \\
&\leq \frac{2}{\pi} \sum_{i=1}^{l} |a_i| \sum_{j=1}^{p} |b_{ij}||x_j| \\
&= \sum_{j=1}^{p} \left(\sum_{i=1}^{l} \frac{2}{\pi} |a_i||b_{ij}| \right) |x_j|
\end{aligned}$$

which concludes the proof. □ □ □

The next lemma gives a similar bound for the bipolar exponential sigmoid networks:

Lemma 5.3.2 *Suppose* $x \in R^p$*. Consider a family of sigmoid neural networks* $\mathcal{F}$ *with members as follows:*

$$f(x) = \sum_{i=1}^{l} a_i \sigma(b_i x)$$

where: $\sigma(.) = \frac{1-e^{-(.)}}{1+e^{-(.)}}$ *is a smooth sigmoid activation function,* l *indicates the number of neurons,* a_i*'s* ($a_i \in R$) *are the weights of the output layer and the* p*-dimensional vectors* b_i*'s defined as:* $b_i = (b_{i1}, \ldots, b_{ip})$ *represent the weights of the hidden layer. Then:*

$$|f(x)| \leq \sum_{j=1}^{p} \left(\sum_{i=1}^{l} |a_i||b_{ij}| \right) |x_j| \tag{5.12}$$

Proof: From the definition of bipolar exponential neural networks:

$$\begin{aligned}
|f(x)| &= \left| \sum_{i=1}^{l} a_i \frac{1-e^{-b_i x}}{1+e^{-b_i x}} \right| \\
&\leq \sum_{i=1}^{l} |a_i| \left| \frac{1-e^{-b_i x}}{1+e^{-b_i x}} \right| \\
&\leq \sum_{i=1}^{l} |a_i||b_i x| \\
&= \sum_{i=1}^{l} |a_i| | \sum_{j=1}^{p} b_{ij} x_j | \\
&\leq \sum_{i=1}^{l} |a_i| \sum_{j=1}^{p} |b_{ij}||x_j| \\
&= \sum_{j=1}^{p} \left(\sum_{i=1}^{l} |a_i||b_{ij}| \right) |x_j|
\end{aligned}$$

which concludes the proof. □ □ □

Now, the following theorems present a set of sufficient conditions for stochastic stability and geometric ergodicity of the families of sigmoid neural network discussed above. These conditions involve the known parameters of the network, and as a result can be easily tested during a practical modeling task. A family of atan sigmoid networks is to be addressed first:

Theorem 5.3.1 *Let* $X_t = (y_{t-k},\ y_{t-k+1},\ \cdots\ y_{t-1},\ u_{t-q+1},\ u_{t-q+2},\ \cdots\ u_{t-d},)$. *Take* y_t, ζ_t *and* u_t *as defined in (5.1). Also assume that* f *is a sigmoid neural network as*

defined in Lemma (5.3.1) with $x = X_t$ *where:* $p = q - d + k$. *Further assume that* $E[|\zeta_t|] \leq M_\zeta$ *and* $E[|u_t|] \leq M_u$. *Define:*

$$\omega_j = \sum_{i=1}^{l} \frac{2}{\pi} |a_i||b_{ij}| \tag{5.13}$$

where $j = 1, \ldots, k$. *Let:* $M_\omega = \max_j \omega$. *Also define the following characteristic polynomial:* $P(z) = z^k - \omega_1 z^{k-1} - \ldots - \omega_k$. *Then the sequence* X_t *is geometrically ergodic if the unique non-negative real zero of* $P(z)$ *is smaller than one. Also, if* X_t *is stationary then "y" is geometrically* α*-mixing.*

Proof: In order to apply the results of Theorem(5.2.2), the existence of a real number x_0 such that the conditions of the theorem are satisfied has to be investigated. Lemma (5.3.1) shows that for any x_0, $\sup_{|X_t| \leq x_0} |f(X_t)| < \infty$. Therefore the case where $\|X_t\| > x_0$ is investigated. Assuming $\|X_t\| > x_0$, there exists at least one index τ such that:

$$|X_{t,\tau}| > \frac{x_0}{\sqrt{q-d+k}}, \; 1 \leq \tau \leq q - d + k \; . \tag{5.14}$$

Next, from Lemma (5.3.1):

$$|f(X_t)| \leq \sum_{j=1}^{k} \omega_j |X_{t,j}| + \sum_{j=k+1}^{q-d+k} \omega_j |X_{t,j}| \; . \tag{5.15}$$

Now, taking an arbitrary positive real number $\rho > 0$:

$$|f(X_t)| \;\; \leq \;\; \sum_{j=1}^{k} (\omega_j + \rho)|X_{t,j}| + \sum_{j=k+1}^{q-d+k} (\omega_j + \rho)|X_{t,j}| - \rho \sum_{j=1}^{q-d+k} |X_{t,j}| \; . \tag{5.16}$$

Now observe that:

$$\rho \sum_{j=1}^{q-d+k} |X_{t,j}| \geq \rho \frac{x_0}{\sqrt{q-d+k}} . \tag{5.17}$$

Therefore:

$$|f(X_t)| \leq \sum_{j=1}^{k} (\omega_j + \rho)|X_{t,j}| + \sum_{j=k+1}^{q-d+k} (\omega_j + \rho)|X_{t,j}| - \rho \frac{x_0}{\sqrt{q-d+k}} \tag{5.18}$$

Defining: $\psi_j = \omega_j + \rho,\ 1 \leq j \leq k,\ h(X_{t,j}) = (M_\omega + \rho)|X_{t,j}|,\ k+1 \leq j \leq q-d+k$ and $c = \rho \frac{x_0}{\sqrt{q-d+k}}$, the next step of the proof would be checking the second condition of Theorem(5.2.2). It suffices to have:

$$M_\zeta + (q-d)E[(M_\omega + \rho)|X_{1,q-d+k}|] < \rho \frac{x_0}{\sqrt{q-d+k}}$$

This means that it suffices to have:

$$M_\zeta + (q-d)M_u(M_\omega + \rho) < \rho \frac{x_0}{\sqrt{q-d+k}} \tag{5.19}$$

Now it can be seen that in order to satisfy Inequality (5.19), x_0 and ρ need only be chosen such that ρx_0 is large enough to satisfy the inequality. Choose ρ sufficiently small (but non-zero) such that if the positive real root of $P(z)$ with $\psi_j = \omega_j,\ (j = 1, \ldots, k)$ is less than 1, the positive real root of $P(z)$ with $\psi_j = \omega_j + \rho,\ (j = 1, \ldots, k)$ is also less than 1. Then, choose x_0 sufficiently larger so that Inequality (5.19) holds. Under such choices of x_0 and ρ, if the unique positive real root of $P(z)$ with $\psi_j = \omega_j,\ (j = 1, \ldots, k)$ is less than 1, all the conditions for geometric ergodicity are satisfied. Moreover, if X_t is stationary then "y" is geometrically α-mixing. □ □□

In the above theorem it is assumed that if the positive real root of $P(z)$ with $\psi_j = \omega_j,\ (j = 1, \ldots, k)$ is less than 1, there exists ρ such that the positive real root of $P(z)$ with $\psi_j = \omega_j + \rho,\ (j = 1, \ldots, k)$ is also less than 1. This assumption requires that a very small

change in the coefficients of $P(z)$ does not change the location of the poles significantly, because ρ can be made arbitrarily small (but not equal to zero) to avoid such a change. A similar result can be obtained on a family of bipolar exponential networks, as follows.

Theorem 5.3.2 *Let* $X_t = (y_{t-k},\ y_{t-k+1},\ \cdots\ y_{t-1},\ u_{t-q+1},\ u_{t-q+2},\ \cdots\ u_{t-d},)$. *Take* y_t, ζ_t *and* u_t *as defined in (5.1). Also assume that* f *is a sigmoid neural network as defined in Lemma (5.3.2) with* $x = X_t$ *where:* $p = q - d + k$. *Further assume that* $E[|\zeta_t|] < M_\zeta$ *and* $E[|u_t|] < M_u$. *Define:*

$$\omega_j = \sum_{i=1}^{l} |a_i||b_{ij}| \tag{5.20}$$

where $j = 1, \ldots, k$. *Also define the following characteristic polynomial :* $P(z) = z^k - \omega_1 z^{k-1} - \ldots - \omega_k$. *Then the sequence* X_t *is geometrically ergodic if the unique non-negative real zero of* $P(z)$ *is smaller than one. Also, if* X_t *is stationary then the process "y" is geometrically* α*-mixing.*

Proof: Defining ω_j's as in (5.20), the rest of the proof is the same as Theorem (5.3.1), and is not repeated here. □ □ □

Theorems (5.3.1) and (5.3.2) give sufficient conditions for the stability of the corresponding sigmoid networks, which can be easily tested. Having addressed the important properties of stochastic stability, geometric ergodicity and geometric α-mixing for a neural ARX model, a learning scheme for dynamic neural modeling is now introduced. A learning theory which can be applied to ARX models provides a useful framework to assess properties such as sample complexity, overfitting and complexity of a model.

5.4 Geometrically α-Mixing PAC Learning

Having described the conditions for stochastic stability and α-mixing of SNN's, the next step is to define a PAC learning scheme for learning with geometrically α-mixing data. One can easily omit "geometrically" and define the learning scheme for a general family

of learning tasks where data are α-mixing but not necessarily "geometrically α-mixing". In order to specialize the results towards the geometric case, only geometrically mixing cases are addressed here.

Definition 5.4.1 *Suppose that z_n is a set of input-output data where $(x_1, \ldots, x_n)$ is a sequence of geometrically α-mixing r.v.s , marginally distributed according to the probability measure $P \in \mathcal{P}$. Then, a function set $\mathcal{F}$ is said to be "PAC learnable with geometrically α-mixing data according to the distance measure d_P" iff an algorithm A can be found based on which for any ϵ and δ, there exists n such that:*

$$\sup_{f \in \mathcal{F}} \Pr\{d_P(f, h) \leq \epsilon\} \geq (1 - \delta) \tag{5.21}$$

Hereafter, referring to the above property, where the meaning is clear,"according to the distance measure d_P" will be dropped. The first step in obtaining some practical results involves bounds on the summation of a sequence of geometrically α-mixing random variables. The results presented by Modha in [16] can provide useful (somewhat conservative) bounds, as given below:

Theorem 5.4.1 *(Modha [16]) Suppose $\{\Lambda_i\}_{i=1}^n$ is a sequence of stationary and geometrically α-mixing zero-mean r.v.s (with the k_i's defined in (5.7)) such that $|\Lambda_i| \leq M$ and $E(|\Lambda_1|^2) \leq Q$. Also, define:*

$$\bar{n} = \lfloor n \lceil (8n/k_3)^{1/(k_2+1)} \rceil^{-1} \rfloor \tag{5.22}$$

where the unary operations $\lfloor . \rfloor$ and $\lceil . \rceil$ refer to "the largest integer smaller than" and "the smallest integer greater than" a given number, respectively.

Then:

$$\Pr\left\{\frac{1}{n}\sum_{i=1}^{n}\Lambda_i \geq \epsilon\right\} \leq (1+4e^{-2}k_1)\exp\left[\frac{-\epsilon^2\bar{n}}{2(Q+\frac{\epsilon M}{3})}\right] \tag{5.23}$$

Next, a lemma is proved that applies Theorem 5.4.1 to evaluate the closeness of the mean value of a function to its empirical mean where the input is a sequence of geometrically α-mixing r.v.s.

Lemma 5.4.1 *Let $x \in [-1\ 1]^d$. Also assume a function $\Upsilon : X \to [0,1]$. Suppose that a set of training data has been generated as: $\{(x_i, \Upsilon(x_i))\}_{i=1}^{n}$, where output data is a sequence of stationary and geometrically α-mixing r.v.s. If the mean value and the empirical mean of Υ are defined as follows:*

$$E_P(\Upsilon) = \int_X \Upsilon(x)\, P(dx)\ ,\ \hat{E}(\Upsilon) = \frac{1}{n}\sum_{i=1}^{n}\Upsilon(x_i) \tag{5.24}$$

then:

$$\Pr\{\hat{E}(\Upsilon) - E_P(\Upsilon) \geq \epsilon\} \leq (1+4e^{-2}k_1)\exp\left[\frac{-\epsilon^2\bar{n}}{4(2+\frac{\epsilon}{3})}\right] \tag{5.25}$$

and:

$$\Pr\{E_P(\Upsilon) - \hat{E}(\Upsilon) \geq \epsilon\} \leq (1+4e^{-2}k_1)\exp\left[\frac{-\epsilon^2\bar{n}}{4(2+\frac{\epsilon}{3})}\right]\ . \tag{5.26}$$

Proof: First define: $\Lambda_i = \Upsilon(x_i) - E_P(\Upsilon)$. Notice that the Λ_i's form a sequence of zero-mean geometrically α-mixing r.v.s to which Inequality (5.23) can be applied with $M = 2$ and $Q = 4$. This will result in Inequality (5.25). Inequality (5.26) can be obtained in the same fashion, assuming $\Lambda_i = E_P(\Upsilon) - \Upsilon(x_i)$. □ □ □

Now, consider an extension of the empirical risk minimization algorithm to geometrically α-mixing random inputs. Since the definition of such an algorithm is straightforward, without writing the formal definition, PAC learning evaluation of the empirical risk minimization algorithm is next evaluated. The following theorem is similar to the theorem proved for m-dependent learning.

Theorem 5.4.2 *Let $x \in [-1\ 1]^d$. Also assume a function Υ as defined in Definition 5.4.1. Suppose that a set of training data has been generated as: $\{(x_i, \Upsilon(x_i))\}_{i=1}^n$, where output data is a sequence of stationary and geometrically α-mixing r.v.s. Also assume that there exists a finite set $\{g_i\}_{i=1}^{\varrho}$ which $\epsilon/2$-covers $\mathcal{F}$. Then the minimum empirical risk results in PAC learning of $\mathcal{F}$ with geometrically α-mixing training data to the accuracy of ϵ. In particular:*

$$\sup_{f \in \mathcal{F}} \Pr\{d_P(f, h) \leq \epsilon\} \geq (1 - \delta) \tag{5.27}$$

whenever:

$$\delta \geq \varrho((1 + 4e^{-2}k_1)\exp\left[\frac{-\epsilon^2 \bar{n}}{64(2 + \frac{\epsilon}{12})}\right] . \tag{5.28}$$

Proof: If $\{g_i\}_{i=1}^{\varrho}$ is an $\epsilon/2$-cover for $\mathcal{F}$, there exists an index t such that $d_P(f, g_t) \leq \epsilon/2$. Without loss of generality, suppose that $d_P(f, g_\varrho) \leq \epsilon/2$. Again, without loss of generality, suppose that the g_i's are renumbered such that: $d_P(f, g_i) > \epsilon$ for $i = 1, \ldots, k$ and $d_P(f, g_i) \leq \epsilon$ for $i = k + 1, \ldots, \varrho$. Note that: $k \leq \varrho - 1$. Notice that the error involved in the minimum empirical risk algorithm would be introducing one of the g_i's,

where $i = 1, \ldots, k$, as the model h. Let the event E be as follows:

$$E = \{\hat{J}_\varrho \leq 3\epsilon/4 \; and \; \hat{J}_i > 3\epsilon/4 \; , \; for \; all \; i = 1, \ldots, k\} \tag{5.29}$$

From the definition of E, it is known that: $E \subseteq \{d_P(f, h) \leq \epsilon/2\}$. Also:

$$\Pr\{E^c\} \leq \Pr\{\hat{J}_\varrho > 3\epsilon/4\} + \sum_{i=1}^{k} \Pr\{\hat{J}_i \leq 3\epsilon/4\} \tag{5.30}$$

Now observe that, for each index i, the cost $\hat{J}_i$ is the empirical mean of the function $|f(.) - g_i(.)|$ based on the available training set. Therefore, Lemma 5.4.1 can be used to evaluate the distance between $\hat{J}_i$, the empirical mean, and $d_P(f, g_i)$, the true mean of the function $|f(.) - g_i(.)|$. Now:

$$\Pr\left\{\hat{J}_\varrho - E_P(|f - g_\varrho|) \geq \epsilon/4\right\} \leq (1 + 4e^{-2}k_1)\exp\left[\frac{-\epsilon^2\bar{n}}{64(2 + \frac{\epsilon}{12})}\right] \tag{5.31}$$

where $\bar{n}$ and k_1 are as defined before. Now, since $E_P(|f - g_\varrho|) = d_P(f, g_\varrho) \leq \epsilon/2$, then:

$$\Pr\left\{\hat{J}_\varrho > 3\epsilon/4\right\} \leq (1 + 4e^{-2}k_1)\exp\left[\frac{-\epsilon^2\bar{n}}{64(2 + \frac{\epsilon}{12})}\right] \; . \tag{5.32}$$

For $\hat{J}_i \leq 3\epsilon/4$ where $i = 1, \ldots, k$, following a similar procedure, the following inequality is obtained:

$$\Pr\left\{\hat{J}_i \leq 3\epsilon/4\right\} \leq (1 + 4e^{-2}k_1)\exp\left[\frac{-\epsilon^2\bar{n}}{64(2 + \frac{\epsilon}{12})}\right] \; . \tag{5.33}$$

Now, since $k \leq \varrho - 1$, the maximum error involved in the overall learning process would be: $\varrho(1+4e^{-2}k_1)\exp\left[\frac{-\epsilon^2\bar{n}}{64(2+\frac{\epsilon}{12})}\right]$. In other words:

$$\sup_{f\in\mathcal{F}} \Pr\{d_P(f,h) \leq \epsilon\} \geq \left(1 - \varrho(1+4e^{-2}k_1)\exp\left[\frac{-\epsilon^2\bar{n}}{64(2+\frac{\epsilon}{12})}\right]\right) \tag{5.34}$$

which results in:

$$\delta \geq \varrho(1+4e^{-2}k_1)\exp\left[\frac{-\epsilon^2\bar{n}}{64(2+\frac{\epsilon}{12})}\right] . \tag{5.35}$$

□ □ □

In an approach similar to that of Chapter 1, as the next step, ϱ is bounded. However, this time no assumptions on the distribution of the data can be made, i.e. the complexity measure ϱ indicated in this chapter is a distribution-free one. This calls for a more careful analysis and results in more conservative bounds.

5.5 Distribution-Free Complexity of SNN's Neural Models

In this section, "ϱ" (the size of the $\epsilon/2$-cover set of $\mathcal{F}$) for SNN's is bounded to construct a practical algorithm that searches for a minimum-complexity neural model. First, some fundamental properties of sigmoid neural networks are evaluated. The following analysis follows the methodology introduced by Barron [7].

Consider a sigmoid neural network with l neurons of the following general form:

$$f_l(x) = \sum_{i=1}^{l} a_i\sigma(b_i x) .$$

Also, define:

$$|w|_1 = \sum_{j=1}^{d} |w_j| .$$

Now, let $\theta = (a_1, \ldots, a_l, b_{11}, \ldots, b_{ld}, \ldots, b_{l1}, \ldots, b_{ld})$. For $\tau_i > 0$, $i = 1, \ldots, l$, a continuous parameter space $\Theta_{l,\tau_1,\ldots,\tau_l}$ is taken to be the set of all such θ for which $|b_i|_1 \leq \tau_i$. For any $C > 0$, let $\Theta_{l,\tau_1,\ldots,\tau_l,C} \subset \Theta_{l,\tau_1,\ldots,\tau_l}$ be the subset of parameters with $\sum_{i=1}^{l} |a_i| \leq C$.

Next, the precision of the above functions is controlled by controlling the resolution of the parameter space. For each $\acute{\epsilon} > 0$ and $C > 0$, let $\Theta_{l,\acute{\epsilon},\tau_1,\ldots,\tau_l,C}$ be a set of parameter points that $\acute{\epsilon}$-covers $\Theta_{l,\tau_1,\ldots,\tau_l,C}$, i.e. for every θ in $\Theta_{l,\tau_1,\ldots,\tau_l,C}$, there exists a θ^* in $\Theta_{l,\acute{\epsilon},\tau_1,\ldots,\tau_l,C}$ such that for $i = 1, \ldots, l$:

$$\begin{aligned} |b_i - b_i^*|_1 &\leq \acute{\epsilon},\ i = 1, \ldots, l \\ \sum_{k=1}^{l} |a_k - a_k^*| &\leq \acute{\epsilon} C \ . \end{aligned} \tag{5.36}$$

Thus, the parameter points in:

$$\Theta_{l,\acute{\epsilon},\tau_1,\ldots,\tau_l,C}$$

can be used to create a covering set:

$$\{f_l(.,\theta^*) : \theta^* \in \Theta_{l,\acute{\epsilon},\tau_1,\ldots,\tau_l,C}\}$$

for the family of functions:

$$\mathcal{F}_l = \{f_l(.,\theta) :\ \theta \in \Theta_{l,\tau_1,\ldots,\tau_l,C}\} \ . \tag{5.37}$$

Now, the Lipschitz constants of specific types of sigmoid functions used in the networks are bounded, and then the precision of such function sets is assessed.

Lemma 5.5.1 *Suppose v_0 is the Lipschitz constant of a sigmoid function $\sigma(.)$. Assume that the conditions of (5.36) hold. Then for any θ in the continuous parameter set $\Theta_{l,\tau_1,\ldots,\tau_l,C}$, there is a θ^* in the discrete set $\Theta_{l,\acute{\epsilon},\tau_1,\ldots,\tau_l,C}$ such that for any $x \in X$:*

$$|f_l(x,\theta) - f_l(x,\theta^*)| \leq (v_0 + 1)\acute{\epsilon} C \tag{5.38}$$

and therefore:

$$d_P(f_l(.,\theta), f_l(.,\theta^*)) \leq (v_0+1)\epsilon C \tag{5.39}$$

where $f_l(x,\theta) \in \mathcal{F}_l$, *and:*

$$d_P(f_l(.,\theta), f_l(.,\theta^*)) = \int_X |f_l(x,\theta) - f_l(x,\theta^*)| P(dx) \; . \tag{5.40}$$

More specifically the bound on the left hand-side of the above inequality can be replaced by $(1+2/\pi)\epsilon C$ *for* $\sigma(u) = \frac{2}{\pi}\tan^{-1}(u)$, *and* $(3/2)\epsilon C$ *for* $\sigma(u) = \frac{1-e^{-u}}{1+e^{-u}}$.

Proof: Take an arbitrary θ in $\Theta_{l,\tau_1,\ldots,\tau_l,C}$ and choose $\theta^* \in \Theta_{l,\epsilon,\tau_1,\ldots,\tau_l,C}$ such that (5.36) holds. Consider the difference between a function $f_l(.,\theta)$ and its corresponding function in $\Theta_{l,\epsilon,\tau_1,\ldots,\tau_l,C}$, i.e. $f_l(.,\theta^*)$. Now:

$$\begin{aligned}
|f_l(x,\theta) - f_l(x,\theta^*)| &= \sum_{k=1}^{l} a_k\sigma(b_k x) - a_k^*\sigma(b_k^* x) \\
&= \sum_{k=1}^{l} a_k(\sigma(b_k x) - \sigma(b_k^* x)) + \sum_{k=1}^{l}(a_k - a_k^*)\sigma(b_k^* x) \\
&\leq \sum_{k=1}^{l} |a_k||\sigma(b_k x) - \sigma(b_k^* x)| + \sum_{k=1}^{l} |a_k - a_k^*| \\
&\leq v_0 \sum_{k=1}^{l} |a_k||b_k x - b_k^* x| + \epsilon C
\end{aligned} \tag{5.41}$$

Now, notice that for all $x \in [-1\ 1]^d$ and all k:

$$\begin{aligned}
|b_k x - b_k^* x| &= |(b_{k1}x_1 + \ldots + b_{kd}x_d) - (b_{k1}^* x_1 + \ldots + b_{kd}^* x_d)| \\
&= |(b_{k1} - b_{k1}^*)x_1 + \ldots + (b_{kd} - b_{kd}^*)x_d| \\
&\leq |b_{k1} - b_{k1}^*||x_1| + \ldots + |b_{kd} - b_{kd}^*||x_d|
\end{aligned}$$

$$\begin{aligned} &\leq |b_{k1} - b^*_{k1}| + \ldots + |b_{kd} - b^*_{kd}| \\ &= |b_k - b^*_k|_1 \\ &\leq \acute{\epsilon} \end{aligned}$$

(5.42)

From (5.41) and (5.42):

$$\begin{aligned} |f_l(x,\theta) - f_l(x,\theta^*)| &\leq v_0 \acute{\epsilon} \sum_{k=1}^{l} |a_k| + \acute{\epsilon} C \\ &\leq v_0 \acute{\epsilon} C + \acute{\epsilon} C \\ &\leq (v_0 + 1)\acute{\epsilon} C \end{aligned}$$

As for the more specific results, note that from (2.5.1), for atan sigmoid functions: $v_0 \leq \frac{2}{\pi}$, and therefore the above function's accuracy can be further specified as: $(\frac{2}{\pi}+1)\acute{\epsilon}C$ for this family of SNN's.

For the bipolar exponential sigmoid network, from(2.5.2), $v_0 \leq \frac{1}{2}$ which results in a more specific accuracy bound of $\frac{3}{2}\acute{\epsilon}C$ for the function set. □ □ □.

In order to see the results of this theorem more clearly and with a notation similar to our formulation of ϵ-covers for function sets, define $\epsilon = (v_0 + 1)\acute{\epsilon}C$. Then the above theorem asserts that an ϵ-cover of $\mathcal{F}_l$ can be formed as:

$$\mathcal{F}^*_l = \left\{ f_l(.,\theta^*) : \theta^* \in \Theta_{l,\epsilon/(v_0+1)C,r_1,\ldots,r_l,C} \right\} . \tag{5.43}$$

This indicates that the cardinality of the function covering set can be bounded by bounding the cardinality of the parameter covering set. Next, a bound on the cardinality of $\Theta_{l,\acute{\epsilon},r_1,\ldots,r_l,C}$ is found.

Lemma 5.5.1.1 *Consider $\Theta_{l,\acute{\epsilon},\tau_1,\ldots,\tau_l,C}$ as defined above. The cardinality of this set can be bounded as follows:*

$$\mathrm{Card}(\Theta_{l,\acute{\epsilon},\tau_1,\ldots,\tau_l,C}) \leq (2e(1+\acute{\epsilon})/\acute{\epsilon})^l \prod_{k=1}^{l} (2e(\tau_k+\acute{\epsilon})/\acute{\epsilon})^d \tag{5.44}$$

Equivalently, for the set $\mathcal{F}_l^*$:

$$\mathrm{Card}\,(\mathcal{F}_l^*) \leq (2e((v_0+1)C+\epsilon)/\epsilon)^l \prod_{k=1}^{l} \left((2e((v_0+1)\tau_k C+\epsilon)/\epsilon)^d\right) . \tag{5.45}$$

Proof: Consider a rectangular grid at width $\acute{\epsilon}/d$ for the coordinates of b_k, and width $\acute{\epsilon}C/l$ for a_k, for $k = 1,\ldots,l$. Intersecting the grid with $\Theta_{l,\tau_1,\ldots,\tau_l,C}$ yields the desired covering set satisfying the requirements of (5.36). Now the cardinality of this set is calculated. Notice that:

$$\Theta_{l,\tau_1,\ldots,\tau_l,C} = \left\{\theta : |b_k|_1 \leq \tau_k,\ \sum_{k=1}^{l} |a_k| \leq C\right\} \tag{5.46}$$

is a Cartesian product of the constraint sets for the a's and b's. Therefore, the desired cardinality can be obtained as the products of the corresponding counts. First, the number of the grid points is bounded:

$$S_{\tau_k} = \{b \in R_d :\ |b_k|_1 \leq \tau_k\} \tag{5.47}$$

where the grid points are spaced at width $\acute{\epsilon}/d$ in each coordinate. Here it is claimed that the union of the small hypercubes that intersect S_{τ_k} is contained in $S_{\tau_k+\acute{\epsilon}}$. This is

because any point b_k in this union has a distance less than $\acute{\epsilon}$ from a point $\acute{b_k}$ in S_{τ_k}, where $|b_k|_1 \leq |\acute{b_k}|_1 + |b_k - \acute{b_k}|_1 \leq \tau_k + \acute{\epsilon}$, which means $b_k \in S_{\tau_k+\acute{\epsilon}}$. Since the volume of the union of the hypercubes is contained in $S_{\tau_k+\acute{\epsilon}}$, the volume of the union can be bounded by the volume of $S_{\tau_k+\acute{\epsilon}}$. From [7], it can be seen that the volume $S_{\tau_k+\acute{\epsilon}}$ is $(2(\tau_k+\acute{\epsilon}))^d/d!$.

Now, the volume of this union of hypercubes is the product of the number of the hypercubes and $(\acute{\epsilon}/d)^d$. Therefore, the number of hypercubes that intersect S_{τ_k} is not greater than $(2d(\tau_k+\acute{\epsilon})/\acute{\epsilon})^d/d! \leq (2e(\tau_k+\acute{\epsilon})/\acute{\epsilon})^d$. For l such parameter vectors b_k, the total count is bounded by $\prod_{k=1}^{l}(2e(\tau_k+\acute{\epsilon})/\acute{\epsilon})^d$. In the same way, the count for the a_k's is not larger than $(2e(1+\acute{\epsilon})/\acute{\epsilon})^l$. Taking the product of the counts results in:

$$\text{Card}(\Theta_{l,\acute{\epsilon},\tau_1,\ldots,\tau_l,C}) \leq (2e(1+\acute{\epsilon})/\acute{\epsilon})^l \prod_{k=1}^{l}(2e(\tau_k+\acute{\epsilon})/\acute{\epsilon})^d \tag{5.48}$$

which concludes the proof for $\Theta_{l,\acute{\epsilon},\tau_1,\ldots,\tau_l,C}$. Considering that $\acute{\epsilon} = \frac{\epsilon}{(v_0+1)C}$, the bound for the function covering set is a direct result of (5.48). □ □ □

Now, the problem of PAC learning of a function set is further investigated. It can be seen that for $\mathcal{F}_l$ (as defined above), the covering set required by the empirical risk minimization algorithm can be generated by the grid described above. This shows that the complexity measure ϱ can be bounded by $\text{Card}(\Theta_{l,\acute{\epsilon}/2,\tau_1,\ldots,\tau_l,C})$. This observation leads to the following theorem.

Theorem 5.5.1 *Let $\mathcal{F}_l$ be as defined in 5.37. Suppose all the assumptions made in Theorem 5.4.2 hold. Also, assume that the $\epsilon/2$-cover required in Theorem 5.4.2 is generated by a grid described in Lemma 5.5.1.1. Then the empirical risk minimization algorithm provides PAC learning with geometrically α-mixing, i.e. for any ϵ and δ there exist n such that:*

$$\sup_{f\in\mathcal{F}_l} \Pr\{d_P(f,h) \leq \epsilon\} \;\geq\; 1 - (2e(2(v_0+1)C+\epsilon)/\epsilon)^l \left[\prod_{k=1}^{l}(2e(6\tau_k C+\epsilon)/\epsilon)^d\right] \times$$

$$(1+4e^{-2}k_1)\exp\left[\frac{-\epsilon^2\bar{n}}{64(2+\frac{\epsilon}{12})}\right]$$

or equivalently:

$$\delta \geq (2e(2(v_0+1)C+\epsilon)/\epsilon)^l\left[\prod_{k=1}^{l}(2e(2(v_0+1)\tau_k C+\epsilon)/\epsilon)^d\right]\times (1+4e^{-2}k_1)\exp\left[\frac{-\epsilon^2\bar{n}}{64(2+\frac{\epsilon}{12})}\cdot\right]$$

Proof: The proof comes as a direct substitution of the bound calculated in Lemma 5.5.1.1 for Card($\mathcal{F}_l^*$) in Inequality 5.34. □□□

A brief glance at the results of the above theorem reveals that the introduced bounds are highly conservative. Also, in many real applications, the values of k_1, k_2 and k_3 may not be known. The direct use of the above bounds may fit applications where huge training data sets, along with some information on the statistical nature of the data, are available. In some applications, one may even want to first use the data to estimate k_1, k_2 and k_3 and then apply the above bounds. However, in many applications, the available training sets are small, and performing another estimation task, merely to find k_1, k_2 and k_3 may not be desirable, as it requires another nonlinear estimation process besides the main function modeling. Also, even if the values of k_i's are estimated, the resulting bounds would not hold exactly due to the estimation error.

As in the case of FIR modeling, one can easily extend the results of the above learning scheme to a more general paradigm of model-free learning. This enables the algorithm to deal with cases where data are noisy or the system that generates the data is not a neural network of known structure. Practically, any real application is a case of model-free learning and that is why this issue was addressed in Chapter 2 in details. Here, due to the significant similarities between m-dependent learning and α-mixing learning schemes, the model-free version of α-mixing learning is not formulated and merely used. In this chapter, the loss function "$l(.,.)$" used corresponds to the distance d_P, i.e. $l(\zeta_1,\zeta_2)$ is

defined as:

$$l(\zeta_1, \zeta_2) = |\zeta_1 - \zeta_2| \ . \tag{5.49}$$

As shown in Chapter 2, the overall effect of using other norms in the formulation is the appearance of a constant μ (as defined in the model-free learning scheme of Chapter 2) in the results. The main reason for the popularity of the L_2 norm is the existence of algorithms such as Least Mean Square (LMS) method for the linearly parameterized optimization tasks. However, for nonlinear optimization processes (including our task), there may not be a substantial reason to prefer this norm over the others, as all gradient-based algorithms may get trapped in local minima when used against nonlinear functions. As a result, here, d_P (which corresponds to L_1 norm) is used in simulations, and it is re-emphasized that any other distance measures that satisfy a uniform Lipschitz condition (as described in Chapter 2) can also be used.

5.6 Minimum Complexity ARX Neural Modeling

Similar to our approach for FIR modeling, new complexity terms based on the functional dependencies of the available bound for δ (or more specifically $\ln(\delta)$) are defined. A brief look at Inequality (5.49) shows that since k_1, k_2 and k_3 are normally unknown (as described above), one can not define a complexity term that encompasses all the statistical aspects of the modeling procedure accurately. This means that, unlike FIR modeling, the complexity measure created here will not be completely supported by the learning results. However, in order to create some reasonably accurate learning-based complexity terms, one can start from the bound on $\ln(\delta)$. Using (5.49):

$$\begin{aligned} \ln(\delta) \ &\geq \ l \ln\left(2e(2(v_0+1)C+\epsilon)/\epsilon\right) + d\sum_{k=1}^{l} \ln\left((2e(2(v_0+1)\tau_k C+\epsilon)/\epsilon)\right) \\ &+ \ \ln(1+4e^{-2}k_1) - \frac{\epsilon^2 \overline{n}}{64(2+\frac{\epsilon}{12})} \ . \end{aligned}$$

Since the value of k_1 is often not available, one may exclude the term: $\ln(1+4e^{-2}k_1)$ from the complexity term. Although by omitting this term, some statistical characteristics of the data are disregarded, making assumptions on a variable that is unknown to us is also avoided. As to the value of $\bar{n}$ things are more complicated, as described below.

Here, some choices for k_2 and k_3 are assumed that seem to be reasonable for a typical modeling application; however, these choices are by no means meant to be the best possible selections and can be replaced by any better estimated or assumed values. It has to be stressed again that the direct estimation of these values from the training data might give a better set of values for k_2 and k_3, but this would involve another estimation problem. In choosing the value of k_2, set $k_2 = 1$. This is because in most applications, the decrease in correlation of data is not too fast. As to k_3, this value is chosen to be: $k_2 = 1/k$ where k is the degree of the system (as defined in (5.1)). This way, it is ensured that the correlation decreases more slowly when the degree of the system is higher. These choices of k_2 and k_3, along with (5.50) suggest the following complexity term:

$$\begin{aligned} C_{snn,np} &= l \ln\left(2e(2(v_0+1)C+\epsilon)/\epsilon\right) \\ &+ d\sum_{k=1}^{l} \ln\left(2e(2(v_0+1)\tau_k C+\epsilon)/\epsilon\right) - \frac{\epsilon^2\sqrt{n}}{\sqrt{512k}(2+\frac{\epsilon}{12})} \end{aligned}$$

Replacing v_0 with the Lipschitz bounds given in this chapter can give specific complexity terms for "atan" and bipolar exponential neural networks. With a discussion similar to the one given in Chapter 3 for FIR modeling, it is preferred to use a more practical complexity measure by replacing C and τ_k with $\sum_{i=1}^{l}|a_i|$ and $|b_k|_1$, respectively. This gives the following practical complexity measure:

$$\begin{aligned} C_{snn} &= l \ln\left(2e(2(v_0+1)(\sum_{i=1}^{l}|a_i|)+\epsilon)/\epsilon\right) \\ &+ d\sum_{k=1}^{l} \ln\left((2e(6|b_k|_1(\sum_{i=1}^{l}|a_i|)+\epsilon)/\epsilon)\right) - \frac{\epsilon^2\sqrt{n}}{\sqrt{512k}(2+\frac{\epsilon}{12})}\,. \end{aligned}$$

Based on the above complexity term, a cost function is defined as follows:

$$J_{snn} = \frac{1}{n}\left[\sum_{i=1}^{n} l(f(x_i), y_i)\right] + \lambda C_{snn} .$$

(5.50)

As mentioned in previous chapters, higher values of λ may give similar testing and training errors that are both too large, while the smaller values of λ may result in small training errors and large testing ones. The choice of λ should be made according to the objectives of the specific application in hand along with the suggestions made in Chapter 3.

In forming the above cost function, it is assumed that the process is geometrically α-mixing. However, as mentioned before, this property can not be assumed easily and must be checked. If the output process is known to be stationary, then from the results given in this chapter, if the positive real root of the characteristic equation is less than one, the process is guaranteed to be geometrically ergodic, stochastically stable and geometrically α-mixing. In order to make sure that the positive real root is indeed less than one, the optimization process can be extended to minimize the magnitude of this root throughout the training process. In order to do so, it is suggested here that a new term be added to the cost function, i.e. assuming that PRR represents this root and $\gamma \geq 0$, the cost function becomes:

$$J_{snn} = \frac{1}{n}\left[\sum_{i=1}^{n} l(f(x_i), y_i)\right] + \lambda C_{atan} + \gamma PRR .$$

(5.51)

The value γ describes how important it is for us to ensure that the model is indeed

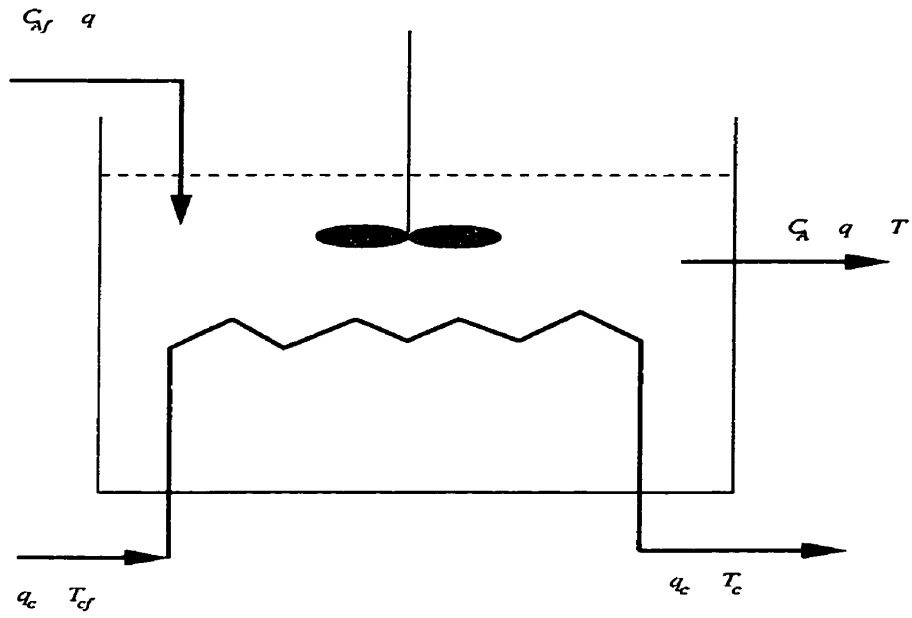

Figure 5.1: Schematic diagram of Continuously-Stirred Tank Reactor (CSTR)

stable and geometrically α-mixing. In many applications, the geometrically α-mixing condition may be assumed without checking. However, in many applications, the system to be modeled is known to be stable and it is often desirable to obtain "stable models" for "stable systems". Then, one has to choose γ large enough to make sure that PRR is actually less than one. Having a stable neural network that accurately models a stable system is considered to be a major objective in dynamic neural modeling.

As in the FIR case, in order to minimize the complex cost function presented above, an optimization algorithm that can handle nonlinear and non-smooth cost functions must be used. The algorithms used here are again variable-structure and fixed-structure systematic evolutionary algorithms, as introduced in Chapter 2. In the next section, some simulation results obtained from the modeling of a simulated system are presented.

5.7 Simulation Results

In this section, an "atan sigmoid neural network" is used for modeling of a simulated Continuously-Stirred Tank Reactor (CSTR) system under an ideal chemical-mixing assumption. A schematic of CSTR is shown in Figure (5.1).

A single irreversible, exothermic reaction, $A \rightarrow B$, is assumed to occur in the reactor,

where A and B are two chemical species. The reaction takes place in a container of fixed volume and the product flow rate, input concentration, temperature and output flow rate are assumed constant at their nominal values. Since in this section, an identification procedure is performed on the simulated CSTR system, the physical model of the process is given here. The process model consists of the following nonlinear ordinary differential equations [38]:

$$\begin{aligned} \dot{C}_A &= \frac{q}{V}(C_{Af} - C_A) - k_0 C_A \exp(-\frac{E}{RT}) \\ \dot{T} &= \frac{q}{V}(T_f - T) + \frac{(-\Delta H)k_0 C_A}{\rho C_p}\exp(-\frac{E}{RT}) + \frac{\rho_c C_{pc}}{\rho C_p V} q_c[1 - \exp(-\frac{h_A}{q_c \rho_c C_{pc}})](T_c - T) \end{aligned} \tag{5.52}$$

where C_A is the effluent concentration of species A, T is the reactor temperature, V is tank volume, q is feed flow rate, C_{Af} is feed concentration, T_f is feed temperature, q_c is coolant flow, T_c is coolant temperature, ρ and ρ_c are densities, C_p and C_{pc} are special heats, k_0 is the pre-exponential factor, E/R is the exponential factor, $-\Delta H$ is the heat of reaction, and h_A is heat transfer characteristics. During the simulation, the sampling time Δt is assumed to be 1 minute. The nominal values of the above parameters are taken from [38], and shown in Table (5.1).

In the identification (or modeling) procedure, CSTR is treated as a system with one input variable q_c and one output variable C_A, i.e. $u = q_c$ and $y = C_A$. This selection of input-output is made since such a model can be used to control the system. During the control phase, introduction of a coolant flow, q_c, allows the manipulation of the reaction temperature and hence of the product concentration, C_A. Other parameters of the modeling task are as follows: $k = 2$, $d = 2$, $q = 4$. Such assumptions (taken from [38]) create a discrete input-output model of the system as follows:

$$C_A(t) = f(C_A(t-2),\ C_A(t-1),\ q_c(t-3),\ q_c(t-2))\ . \tag{5.53}$$

Variable	Symbol	Nominal value
Tank volume	V	$100\ l$
Feed flow rate	q	$100\ l\ min^{-1}$
Feed concentration	C_{Af}	$1\ mol\ l^{-1}$
Feed temperature	T_f	$350\ K$
Coolant flow rate	q_c	$100\ l\ min^{-1}$
Coolant temperature	T_c	$350\ K$
Densities	$\rho,\ \rho_c$	$1000\ g\ l^{-1}$
Specific heat	$C_p,\ C_{pc}$	$1\ cal\ g^{-1}\ K^{-1}$
Pre-exponential factor	k_0	$7.2 \times 10^{10}\ min^{-1}$
Exponential Factor	E/R	$9.98 \times 10^3\ K$
Heat of reaction	$-\Delta H$	$2.0 \times 10^5\ cal\ mol^{-1}$
Heat transfer characteristics	h_A	$7 \times 10^5\ min^{-1}\ K^{-1}$
Sampling period	Δt	$0.1\ min$

Table 5.1: Nominal parameters of a simulated CSTR system.

In the following simulations, in order to generate the input to the CSTR system, an independent random sequence identically distributed according to a normal distribution with mean of 100 liters/minute and variance of 10 liters/minute has been generated. The output sequence y_n at each sample time is formed as the response of CSTR system to the input plus a zero-mean normal random noise with variance of 0.1. In order to ease modeling calculations and follow the framework of the theoretical results, the input and output of the system have been normalized to the interval of $[-1\ 1]$ mol/liter, as in [38]. The normalization process is based on the "max-min" method, i.e. for each variable x (which includes the noise), the normalized variable x_{norm} is computed as:

$$x_{norm} = 2\frac{x - min(x)}{max(x) - min(x)} - 1\ . \tag{5.54}$$

This method of normalization violates our dependence assumptions in the strict sense as the values of $max(x)$ and $min(x)$ (found based on a long record of the output) may occur at any point in time and thus influence the statistical dependency of the variables. However, in practice, this method is used for modeling of CSTR (see [38] for example) to create a mapping of the output to the interval $[-1\ 1]$, and therefore used. A set of

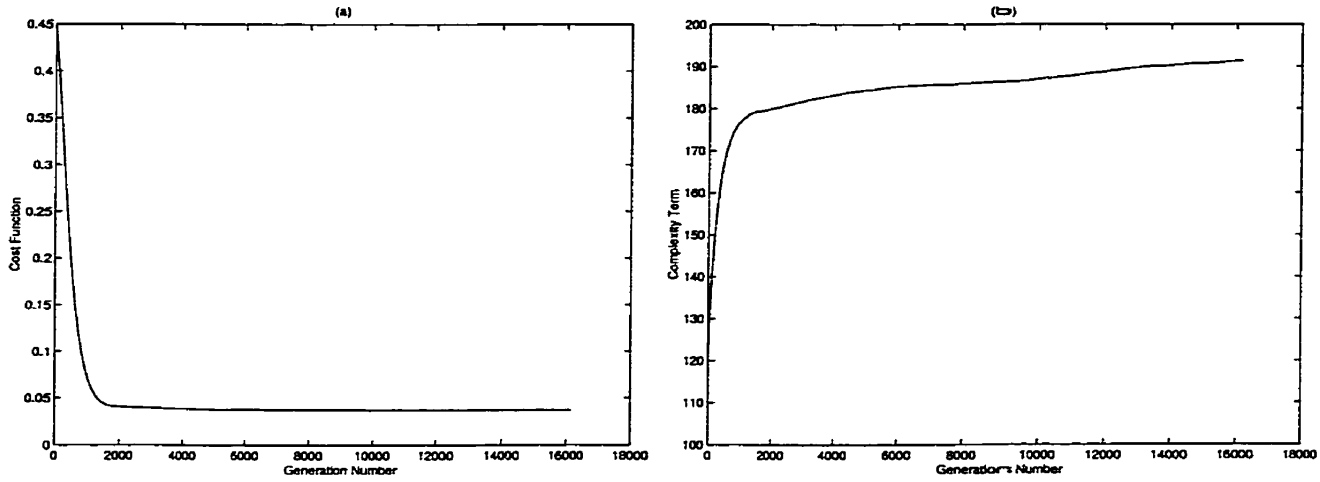

Figure 5.2: (a) Cost function (left), and (b) Complexity term (right) for Simulation 1

input-output data of length 2000 has been generated of which the first 650 data points were used for training and the rest of the points for testing the model.

Notice that all the learning results of this chapter are generated when the empirical risk minimization algorithm along with the grading method of generating an $\epsilon/2$-cover (introduced in this chapter) are used during the optimization process. However, when a very fast computer is not available, this optimization process may be too time consuming and is not applied here. It is necessary to relax the assumptions to use a more efficient optimization method. The optimization algorithms applied in the following simulations are those introduced in Chapter 3, using the new cost functions presented in this chapter.

5.7.1 Simulation 1

In the first simulation, using a family of atan SNN's with 5 neurons, the evolutionary fixed-structure neural modeling algorithm is applied to search for a neural network with three neurons, assuming that both λ and γ are set to zero. The settings of the algorithm are as follows: $MinSlope = -1 \times 10^{-8}$, $BackStep = 1000, w_a = w_b = 0.0005$, $\lambda = 0$, $\gamma = 0$, $N = 50$, $\epsilon = 0.05$. Figures (5.2.a) and (5.2.b) depict the evolution of the cost function and the complexity term, respectively.

The training cost function for this simulation (which is the same as the empirical

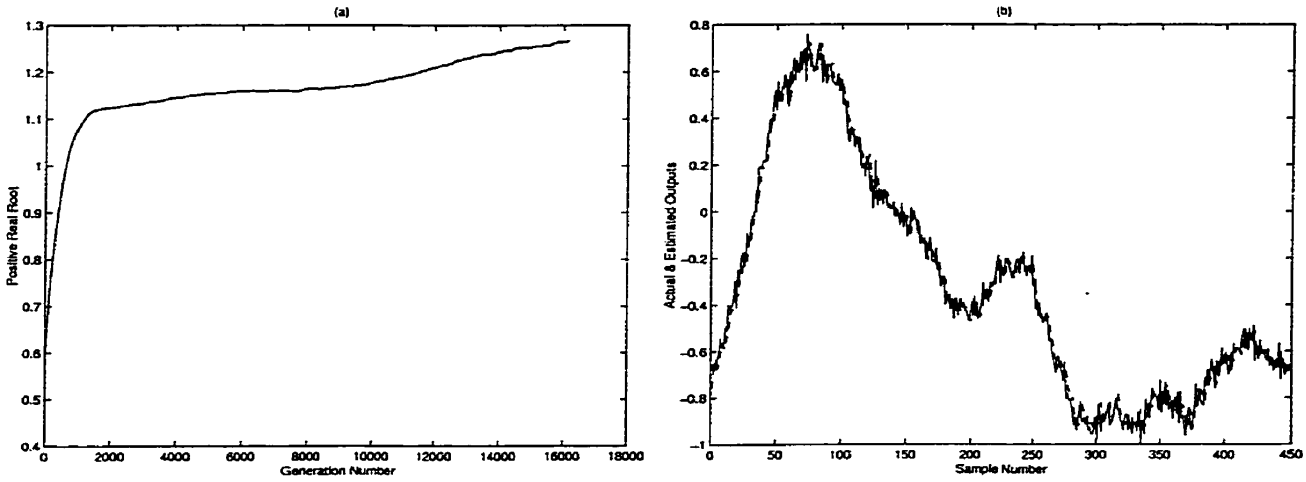

Figure 5.3: (a) Positive real root of the best network of each generation for Simulation 1 (left), and (b) Actual (solid) and estimated (dashed) outputs for the testing data for Simulation 1 (right)

error) is 0.0368. As can be seen, the value of the complexity term increases throughout the process of evolution and the final network has the complexity of 191.34. Figure (5.3.a) shows the evolution of the positive real root (PRR) of the characteristic polynomial. As can be seen, PRR of the final neural model is 1.2678 , which is larger than one and the stability of the model, therefore, can not be guaranteed. Then the performance of the model is assessed against the testing data set. The testing empirical error is 0.0431, which is still small but somewhat larger than the training error. The estimation graph for the first 450 points is shown in Figure (5.3.b). In this figure, the actual (solid line) and the estimated (dashed line) are compared to each other.

It is also important to notice that when PRR is larger than one (as in this simulation), none of the learning results used to define the cost function can be guaranteed. Desirable learning performance from such a simulation may not occur.

5.7.2 Simulation 2

In the second simulation, all the input-output points as well as the settings are the same as the those of Simulation 1, except that $\lambda = 2 \times 10^{-4}$ and $\gamma = 4 \times 10^{-2}$. The resulting curves of the cost function and complexity term are shown in Figure (5.4). The training

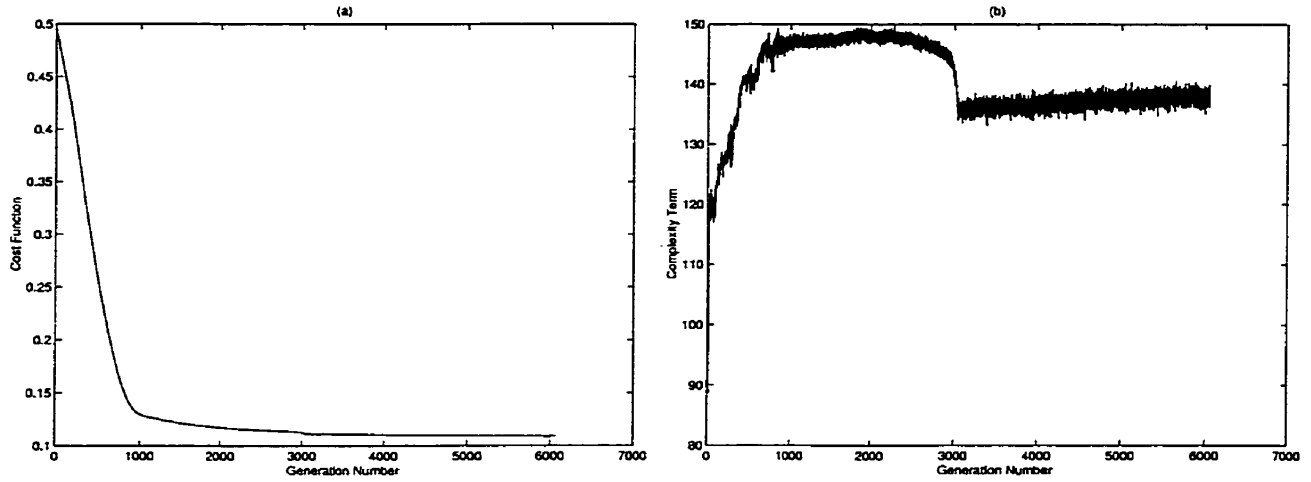

Figure 5.4: (a) Cost function (left), and (b) Complexity term (right) for Simulation 2

cost function, empirical error, and complexity term for the resulting network are 0.1092, 0.0401, and 138.5287, respectively. From Figure (5.4.b), it can be observed that the complexity term does not rise as fast as in Simulation 1, and reaches a value smaller than the previous simulation.

Figure (5.5.a) shows the location of the positive real root (PRR) of the characteristic polynomial. The PRR for the resulting network is 1.0361, which is still larger than one and can not guarantee the stability. This suggests that the value of γ may have to be further increased (which will be implemented in the next simulation). In the testing phase, the testing empirical error of 0.0442 is obtained. As can be seen, the difference between the testing and training errors in Simulation 2 is slightly smaller than that of Simulation 1. The estimation curve for the first 350 points is shown in Figure (5.5.b), which indicates that the estimation quality decreases as the signal approaches the extreme boundaries of its interval.

Although both testing and training empirical errors of Simulation 2 are larger than those of Simulation 1, the relative proximity of the testing and training values can be regarded as the significant characteristic of Simulation 2.

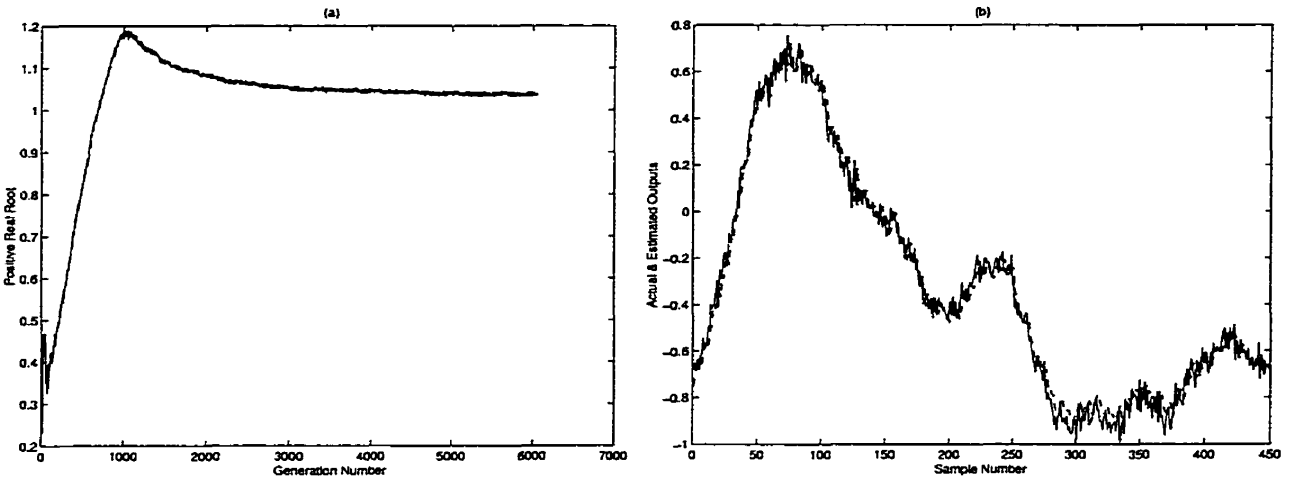

Figure 5.5: (a) Positive real root of the best network of each generation for Simulation 2 (left), and (b) Actual (solid) and estimated (dashed) outputs for the testing data for Simulation 2 (right)

5.7.3 Simulation 3

In this simulation, the evolutionary variable-structure neural modeling with the following settings is applied: $MinSlope = -1 \times 10^{-8}$, $BackStep = 1000, w_a = w_b = 0.0005$, $\lambda = 2 \times 10^{-4}$, $\gamma = 7 \times 10^{-2}$, $N = 50$, $l_{low} = 2$, $\kappa = 100$.

Figure (5.6) shows the curves of the cost function as well as the complexity term of this simulation. The algorithm stops at $l = 4$; however, since the cost function of the network with two neurons is lower than that of the one with three neurons, the algorithm introduces the final network with two neurons as the output function. The jumps in the complexity term when a new neuron is added play an important role in this selection (as expected). The jumps in the complexity curve correspond to the points where new neurons are added. Notice that the defined complexity term depends on the number of neurons directly and as a result adding a new neuron can make a significant change in the complexity. These sudden jumps in complexity term were not present in the case of FIR modeling. This is because the number of neurons l appeared in the complexity measures of FIR models only in the form of upper bounds of the summations over the weights of the model. Therefore, adding a new neuron with all its weights set to zero (or small values) would not cause a jump in the complexity term. However, as mentioned

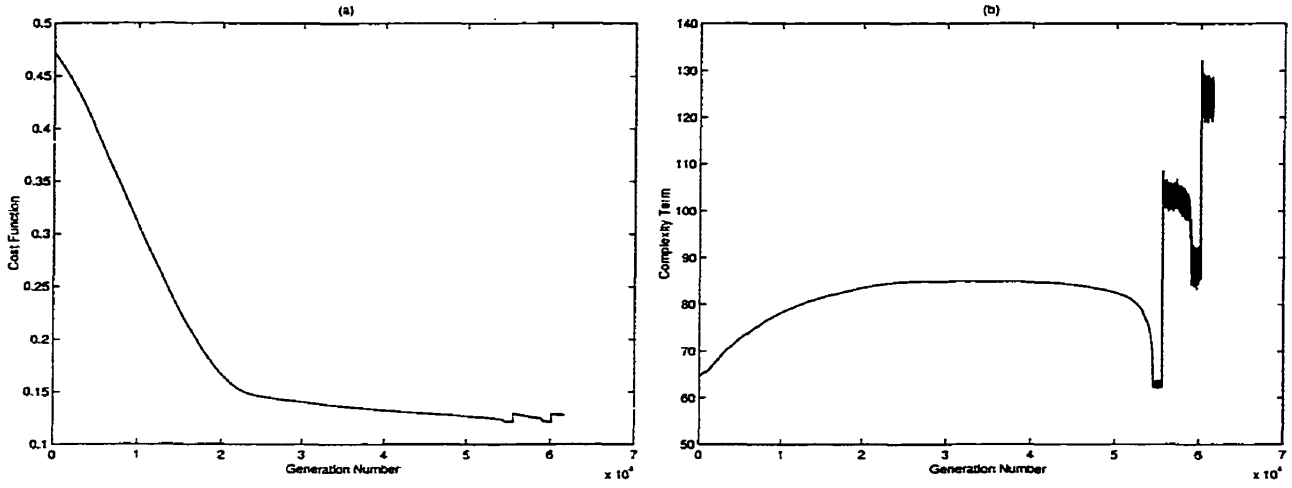

Figure 5.6: (a) Cost function (left), and (b) Complexity term (right) for Simulation 3

above, the complexity terms defined for ARX models contain multiplicative term of l and therefore create the jumps.

Also, notice that after adding a new neuron there always exists a period of rapid oscillations in the complexity curve. Oscillations can be reduced by choosing smaller values of w_a and w_b at the expense of having a slower training procedure. In other words, these variations are due to high values of adaptation parameters w_a and w_b and reducing the values of these parameters would reduce the variations, but would make the entire training process undesirably slow.

The output network has the training cost function and empirical error of 0.1216 and 0.0436, respectively. The complexity term for this network is 88.4046 .

Figure (5.7.a) shows the positive real root of the best network of each generation. As can be seen, the location of the real positive root for the output function is 0.8289, which is less than one and therefore the resulting network is stable. Figure (5.7.b) depicts the performance of the network against the testing data. The empirical error for the testing set of data is 0.0458 . This value being close to the training value indicates that the model has avoided overfitting the data.

The direct comparison of the simulation results given here with those of the literature may not be possible as the cost function used in all existing simulations in the literature

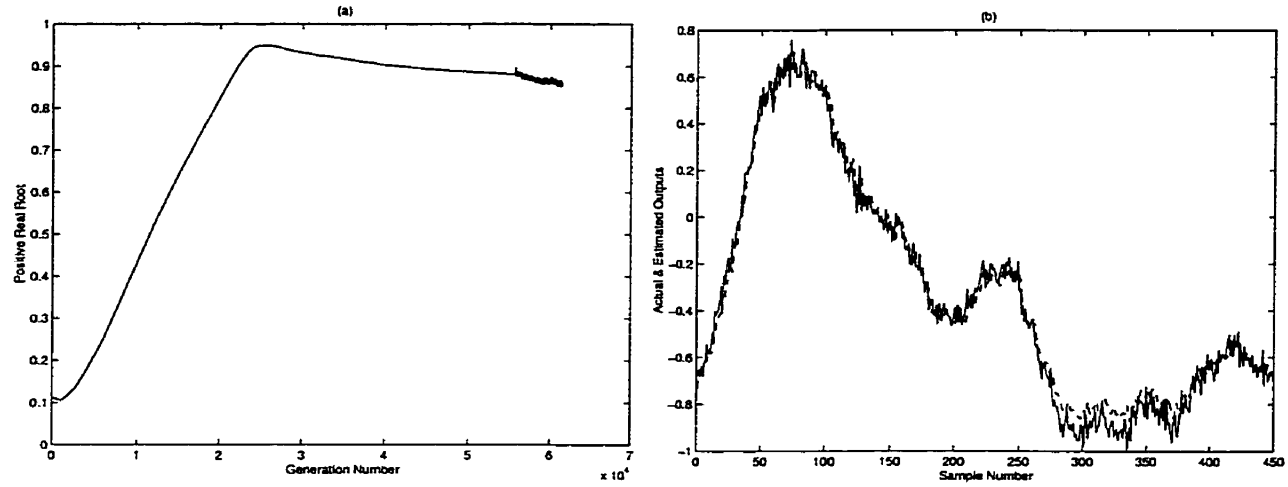

Figure 5.7: (a) Positive real root of the best network of each generation for Simulation 3 (left), and (b) Actual (solid) and estimated (dashed) outputs for the testing data for Simulation 3 (right)

is the empirical error, calculated as the sum of the squared error (which is significantly different from the one used here). Also, in some cases (such as [38]), it seems that the entire data set has been used for training and as a result the comparison between the testing-training balance may not be possible. Moreover, in none of the literature in CSTR modeling is the issue of stability mentioned. However, judging from the overall form of the estimated and actual curves, it seems that the tracking error in [38] may be slightly smaller than the graphs given here.

Each simulation in this section takes between 300 to 420 minutes on a computer with 400MHz Pentium II processor.

5.8 Discussion

Here, the results of the chapter are discussed:

1. The above simulations indicate that the proposed algorithm can provide ARX models that have relatively similar performances on the testing and training data.

2. As can be seen in Simulation 1, without including the value of the positive real root into the optimization process, the algorithm may results in models whose stability

is unknown. This encourages the use of non-zero γ. However, the condition for stability is a sufficient one, and even when the condition is violated the model may still be stable.

3. As in the FIR case, the resulting model depends on the algorithm settings, i.e. different values of N, w_a, w_b, λ, γ, *BackStep* and *MinSlope* would give different models of the training data. At the same time, one can use the above settings to implement the objectives of the modeling process. As an example, if a fast training process is desirable, smaller values of N, higher values of w_a and w_b, and smaller values of *Backstep* can be used. Repeating the training process to find the "appropriate settings" may be a more reliable strategy.

4. Since the number of hidden neurons l appears in the complexity term explicitly, one can expect to see a jump in the cost function when the variable-structure algorithm adds a new neuron to the structure. This jump seems to perform a vital role in evolutionary variable-structure neural modeling, as such a sudden jump in the cost function may easily stop the search.

5. The bounds on the sample complexity are distribution-free, and as a result are highly conservative. Tighter bounds can be found assuming particular probability distributions (or even families of probability distributions), as in the case of FIR modeling.

6. The idea of forming the covering set of a function family by grading the parameter space (introduced by Barron [7]) seems to be both feasible and practical. Although the process may be tedious and time-consuming, in some applications where reliability plays a more crucial role than the computation time, one may choose to use this process in order to gain confidence in the learning behaviour of the model.

7. Modeling of the CSTR system (which is used in literature as a benchmark of nonlinearity) shows that both evolutionary algorithms introduced here can model complex nonlinear dynamic systems.

5.9 Summary

The results of this chapter can be briefly reviewed as follows.

- A new sufficient condition for stochastic stability, geometrical ergodicity, and α-mixing properties of sigmoid neural networks is presented. This condition is defined over the parameters of the network and can be easily evaluated.
- With sufficiently large values of γ, stable neural models of CSTR system can be obtained.

Chapter 6

Modeling of Neuromuscular Blockade

6.1 Introduction

For a patient undergoing surgery, unconsciousness may not suffice, as some muscles may move due to involuntarily muscular activities. In order to avoid such unwanted movement, drugs (such as atracurium) are injected to create muscle relaxation or neuromuscular blockade. The amount and timing of such injections play critical roles in overall medical procedures as too small a dose of the drug may not block the motion completely and an excessive dose of such drugs may cause long term neuromuscular disorder. Accurate identification and control of the neuromuscular blockade system is therefore important.

Unlike the level of unconsciousness which cannot be measured easily, the level of muscle relaxation can be monitored on-line and non-incisively via evoked EMG responses. This has encouraged a number of researchers to develop experimental dynamic models for this nonlinear system. All such models are based on the average values of some parameters that vary significantly from one person to another. This variability, together with the nonlinear nature of the model, suggests that more sophisticated models such as neural networks might be used to provide more accurate estimation of the system.

In this chapter, neural networks are applied to obtain a nonlinear model of the response of the neuromuscular blockade system to drugs such as atricurium. Since real data were not available, a parametric model based on the chemistry of the problem is used to simulate the actual system. Section 6.2 describes the entire process of neuromuscular blockade and the model which will be used. In Section 6.3, a sigmoid neural model is developed for the process, and the performance of the developed model is assessed against a set of testing data. Section 6.4 discusses the results, and Section 6.5 concludes the chapter.

6.2 Muscle Relaxation and Neuromuscular Blockade

Define $u(t)$ as the input dose of the drug and $c_p(t)$ as the plasma concentration of the drug. In Pharmacokinetics, it is shown (see [15] for example) that after a dose of drug, the plasma concentration of atracurium declines rapidly in two exponential phases corresponding to distribution and elimination. The elimination compartment models a phenomenon referred to as "Hofman elimination" [9]. As a result, combining the two compartments together, one can model the relation between $c_p(t)$ and $u(t)$ as a second order linear system. Based on the empirical data gathered from different patients, the transfer function of this system is reported as [15]:

$$H_p(s) = \frac{C_p(s)}{U(s)} = \frac{s+z}{(s+p_1)(s+p_2)} \tag{6.1}$$

where $U(s)$ and $C_p(s)$ represent the Laplace transforms of $u(t)$ and $c_p(t)$ respectively, and:

$$\begin{aligned} z &= 0.0940 \text{ min}^{-1} \\ p_1 &= 0.3247 \text{ min}^{-1} \\ p_2 &= 0.2079 \text{ min}^{-1} . \end{aligned}$$

Similarly, to characterize different aspects of drug effect, a third compartment known as the "effect compartment" is introduced. This compartment is connected to the central compartment and relates $c_p(t)$ to $c_e(t)$. The signal $c_e(t)$ determines how the effective concentration of the drug in blood varies through time. The effect compartment which describes the dynamics of effective concentration of the drug can be expressed as follows:

$$H_e(s) = \frac{C_e(s)}{C_p(s)} = \frac{k}{s+p_0} \tag{6.2}$$

where $C_e(s)$ represent the Laplace transform of $c_e(t)$, $p_0 = 0.9910 \text{ min}^{-1}$ and $k =$

$0.0120\ \text{min}^{-1}$.

Including the delay between the central and effective compartments, the overall transfer function from $u(t)$ to $c_e(t)$ can be expressed as follows:

$$H(s) = \frac{C_e(s)}{U(s)} = \frac{k(s+z)e^{-s}}{(s+p_0)(s+p_1)(s+p_2)} \tag{6.3}$$

where the values of z, p_0, p_1 and p_2 are given above.

Now, based on $c_e(t)$, one can calculate the desired output level of the neuromuscular blockade, named as $r(t)$. This variable, which is normally expressed as a number between 0 and 100 at each time t, indicates the level of blockade, i.e. 100 is regarded as full muscular activity and 0 as full paralysis. The relation between express $r(t)$ and $c_e(t)$ reveals the nonlinear behaviour of the system. Experimentally, it has been observed that the functional dependency between $r(t)$ and $c_e(t)$ is mainly of saturation type. In Pharmodynamics, this saturation effect for atracurium is normally modeled by the Hill equation [9]:

$$r(t) = \frac{100 c_p(t)^\nu}{c_{e,50}^\nu + c_p(t)^\nu} \tag{6.4}$$

where $c_{e,50} = 0.404\text{ml}^{-1}\ \mu\text{g}$ is the drug concentration at 50 percent effect. The reported value for ν in Hill's equation is 2.98 .

The existence of this saturation element makes the entire process a nonlinear system and calls for the use of a nonlinear model to describe the system. In the next section, this system is modeled using SNN's.

6.3 Neural Modeling of Neuromuscular Blockade

Since real data from the neuromuscular blockade system were not available, simulated data representing the actual system are formed using the model described above. Then,

the training and testing of performance of our neural nets will be assessed using the resulting simulated data.

The discrete model is assumed to have the following form:

$$r(t) = f(r(t-2),\ r(t-1),\ u(t-4),\ u(t-3)) + \chi(t) \tag{6.5}$$

where the sampling time is assumed to be 20 seconds. This is due to the fact that in the actual system the input rate as well as the output measurements are updated every 20 seconds. The delay of one minute between the input and the output translates to roughly a delay of 3 samples in the discrete model.

In order to emulate the noisy nature of actual EMG measurement systems, a sequence of normally distributed random numbers ($\chi(t)$) with mean of zero and variance of 3.5 has been added to the output. The training and testing data sets are then normalized such that the variables r and u fall in the interval of [-1 1]. The training and testing data set consist of 450 and 550 input-output samples, respectively. Since it is known that the neuromuscular blockade is a stable system, it is desirable to ensure that the resulting neural model is stable (geometrically ergodic). As a result, in the following modeling task, a non-zero γ is used.

The evolutionary variable-structure neural modeling is used for the training. The settings of the algorithm are the following: $N = 50$, $w_a = w_b = 0.0005$, $\lambda = 0.0002$, $\gamma = 0.08$, $\epsilon = 0.05$, $MinSlop = -1 \times 10^9$, $BackStep = 1000$, $l_{low} = 2$, $\kappa = 100$.

The algorithm stops after generating a network with 3 neurons; however, due to the lower cost function of the last network with two neurons, the latter network is used as the output of the algorithm. This can be seen in Figure (6.1.a), which shows the cost function evaluated for the best network of each generation. Due to the jump in the value of the complexity term, there exists a jump in the overall cost function that makes the three-neuron-networks too costly. The evolution of the complexity term is shown in Figure (6.1.b). The jump in the complexity term can be seen in this graph. The cost function, the empirical error and the complexity term for the resulting two-neuron network are

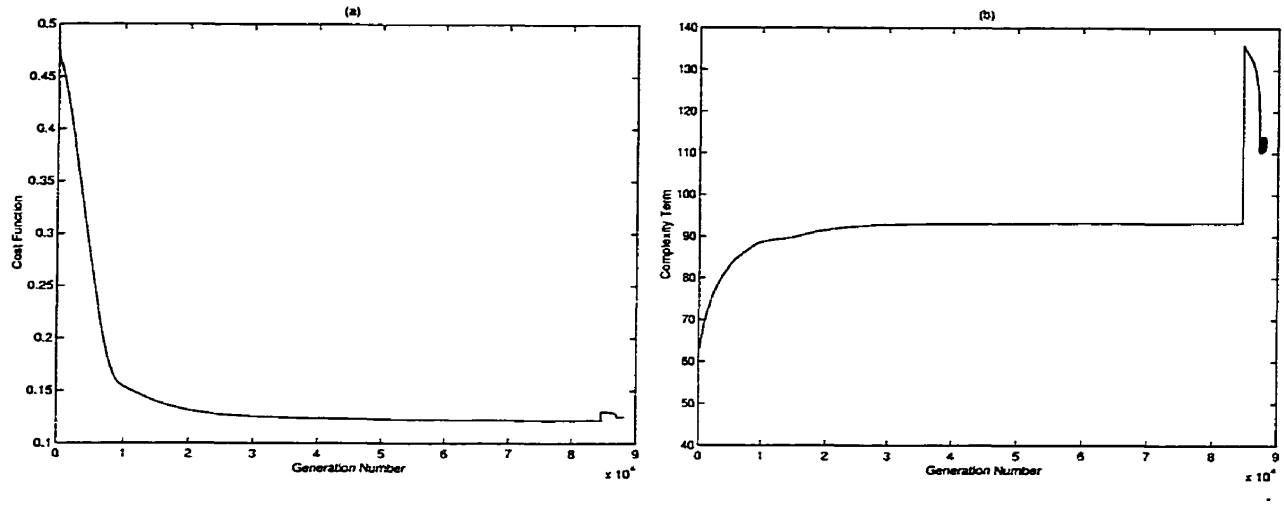

Figure 6.1: (a) Cost function (left), and (b) Complexity term (right)

0.1218, 0.0382, and 93.2590, respectively.

Figure (6.2.a) shows the location of the positive real root of the characteristic polynomial of the best network of each generation. The value of the positive real root (PRR) for the resulting network is 0.8111, which indicates that the resulting network is stable. In Figure (6.2.b), the performance of the resulting network against the testing data set is depicted.

The empirical error on the testing data is 0.0385, which is very close to the testing one. Also, from Figure (6.2.b) one can see that the estimated curve closely tracks the actual one.

In order to see the prediction performance of the neural model more clearly, in Figure (6.3), the auto-correlation of the prediction error is depicted. As can be seen, the only point with high correlation corresponds to zero shift of the signal. This shows that the error signal is merely a noise and contains little information.

This can be further seen in Figures (6.4.a) and (6.4.b), where "the power spectral density" and "the cumulative integrated power spectral" of the error are depicted, respectively. As can be seen in Figure (6.4.a), the spectrum of the error is relatively flat which resembles that of a white noise. The small increase in the power of the signal at high frequency indicates that the model is less successful in prediction of high fre-

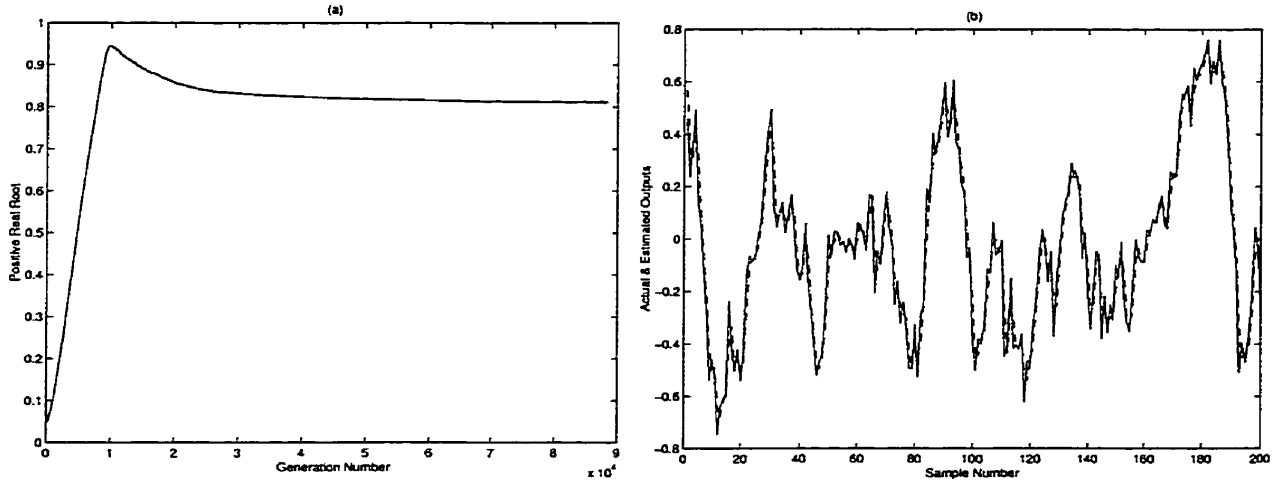

Figure 6.2: (a) Positive real root of the best network of each generation (left), and (b) Actual (solid) and estimated outputs for the testing data (right)

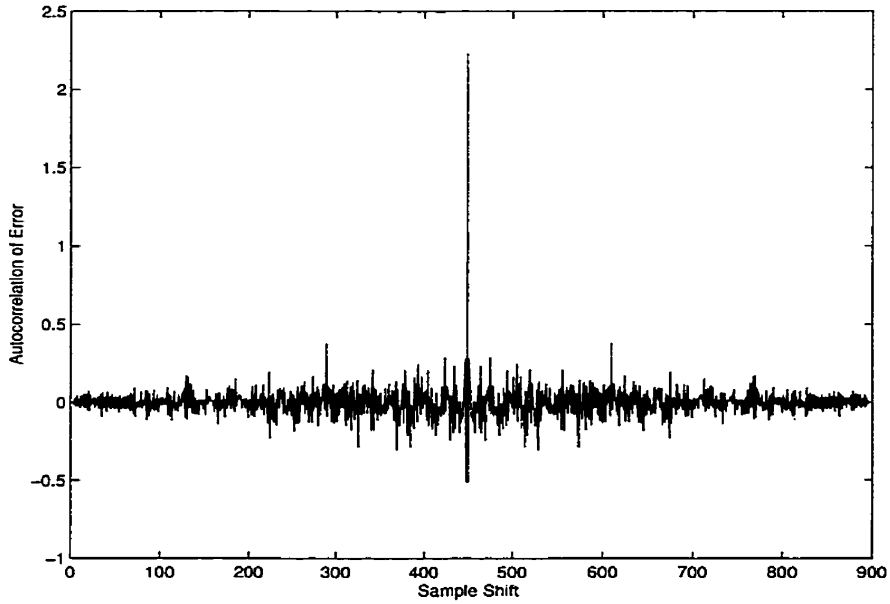

Figure 6.3: Auto-correlation of the empirical prediction error

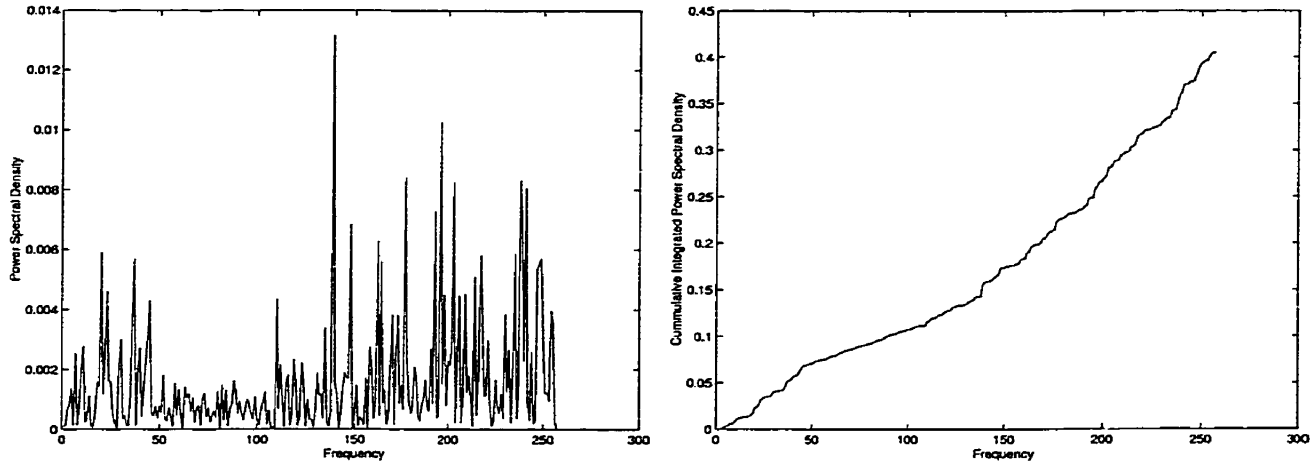

Figure 6.4: (a) Power spectral density of the empirical prediction error (left), and (b) Cumulative integrated power spectral density of the empirical prediction error (right)

quency variations. However, since most of the high frequency contents of the signal can be attributed to the added noise, this may not be a considered as a disadvantage of the model.

Due to the noisy nature of signal and the jumpy form of the power spectral density, the curve of the cumulative integrated power spectral is sometimes considered as a more visually-appealing graph. The curve in Figure (6.4.b) shows the cumulative summation of the power spectral density for any given frequency. The closer this curve becomes to a straight line, the whiter the signal is. As can be seen, Figure (6.4.b) gives a curve which resembles a straight line which in turn indicates that the error signal is close to the white noise.

Performing this simulation takes about 300 minutes on a computer with 400MHz Pentium II processor.

6.4 Discussion

Considering the results obtained in this chapter, the following remarks can be made.

1. The higher values of γ result in neural models with higher stability margins. However, for such simulations, the empirical error of the resulting network might be too

large.

2. Since the neuromuscular blockade system is known to be a stable system, a non-zero γ was used to ensure that the resulting network is stable. However, if having a neural model with unknown stability status is still acceptable, one can set $\gamma = 0$, which may result in networks with smaller empirical errors and even closer testing and training errors.

3. The resulting network performs similarly on the testing and training data sets which further supports the idea of using learning-based cost functions. The fact that the testing and training errors are very close suggests that the value of λ can be further reduced to obtain even smaller empirical errors.

4. Since the neuromuscular blockade system is known to be time-variant (i.e. the parameters of the system change as the time evolves), the model can be designed to follow the changes of the system assertively. Once a model is developed off-line and throughout a batch training procedure, it can be updated after a few samples by including the new measured data points in the on-line training process and disregarding some of the old training points. Assuming that the computations involved in updating the neural model can be performed in a few minutes using a fast computer, the adaptive version of the above modeling procedure is a more suitable approach, but needs to be confirmed throughout the implementation of the model on the actual system.

5. A certain dose of atricurium causes different degrees of muscular paralysis on different people so that different model parameters may be needed for different patients. In order to deal with this problem, one can take an approach similar to the one described in the previous remark, i.e. using adaptive implementation of the algorithm. This way, the modeling process starts with the parameters obtained from the data taken from different patients and updates the model throughout the process. Also, characteristics such as weight, age and gender can be used to create models that work for a group of people with certain similar characteristics.

6. The neural model developed here can be use to design a nonlinear controller that generates the drug dose input so as to provide a certain level of muscle relaxation.

7. The performance of the neural modeling of the neuromuscular blockade can be best evaluated using actual data sets taken from the real systems used.

6.5 Summary

A summary of the chapter is as follows.

- The neuromuscular blockade system can be modeled using neural networks.
- Due to the fact that the neuromuscular blockade systems is known to be stable, the methodology used in this chapter is designed to give a stable model of the system.
- The similar performance over the testing and training data suggests that the obtained neural model is a reliable one

Chapter 7

Conclusions and Future Works

In this chapter, the main results of the thesis are briefly reviewed. The main contributions of the research can be listed as follows.

1. The new inequality on the summation of a sequence of m-dependent r.v.s, presented in this thesis helped extending the conventional PAC with i.i.d. to a more general framework of learning with m-dependent r.v.s. This new paradigm of m-dependent learning allows the quantitative evaluation of the learning properties of FIR modeling procedures. Using the results of m-dependent PAC learning theory, the learning properties of the following families of neural FIR models are assessed: Gaussian RBFN's, Reciprocal Multi-Quadratic RBFN's, atan SNN's, bipolar exponential SNN's, Volterra NN's as well as simple linear models. These results give bounds on the number of training data points which guarantees an accurate and reliable model and avoids overfitting the data. These bounds are shown to be most useful when the user has access to large training sets. Moreover, the learning properties of FIR modeling with the above families of NN's are compared with each other. Although the comparison is based on the sufficient bounds of the sample complexities, the results can still be used in selecting a neural structure. The comparison shows that from the standpoint of learning theory, bipolar exponential networks are the most desirable neural models.

2. Based on the learning properties of neural FIR models and using the functional dependencies between the learning parameters, complexity terms are introduced that reflect the complexity of learning with such neural structures. Based on the resulting complexity terms for neural FIR models, a set of cost functions is constructed that creates a balance between the empirical error and the complexity of reliable learning of the model. This cost function can be used to avoid overfitting when the size of

the training data is small. For a typical task of FIR modeling with small training sets, evolutionary neural modeling algorithms are proposed that consider the learning properties of the model. This is done by minimizing the above mentioned cost functions and searching for appropriate structure as well as set of parameters that avoid overfitting. Simulation results testify to the suitable performance of the proposed algorithms.

3. The evolutionary variable-structure neural modeling algorithm is applied to FIR modeling of the paper machine's next-scan-estimation. This shows that when the sensors are off-line, for the first few scan lines, neural networks can use the previous information to estimate the future scans (rows). This shows that neural FIR models can be successfully used to approximate the industrial systems without overfitting the small training data, provided that an appropriate learning-based complexity measure is included in the cost function.

4. A set of sufficient conditions for the important properties of stochastic stability, geometric ergodicity, and geometric α-mixing properties of two families of SNN's (atan and bipolar exponential) are presented. These conditions not only evaluate the important issue of stability for some important neural models, but also allow the extension of PAC learning to a more general learning scheme. Using these conditions, a new extension of the conventional PAC learning framework is presented that includes learning with geometrically α-mixing data. This provides a framework to assess the learning properties of a group of important neural ARX models. In order to obtain specific results, the PAC learning with geometrically α-mixing data is applied to evaluate the learning properties of atan and bipolar exponential SNN's. The results bound the number of training data points that guarantees an accurate and reliable ARX model and avoids overfitting. These results are mainly applicable to the modeling tasks where the user has access to large data bases.

5. Based on the learning properties of sigmoid neural ARX models, complexity terms are introduced that reflect the complexity of learning with such neural structures. Then, using the resulting complexity terms for sigmoid neural ARX models, new cost functions are constructed that create a balance between the empirical error and

the complexity of reliable learning of the model. For a neural ARX modeling task (using the above mentioned SNN's) with small training data sets, the evolutionary neural modeling (similar to that of FIR modeling) is introduced. By minimizing the above-mentioned cost functions, the algorithm searches for suitable structure as well as parameters sets that avoid overfitting. Simulation results obtained from the modeling of a Continuously-Stirred Tank Reactor (CSTR) testify to the successful performance of the proposed algorithms.

6. The evolutionary ARX modeling algorithm is applied to the neuromuscular blockade system, which results in a neural model that learns the system without overfitting the data. This shows that neural ARX models are capable of approximating complex systems without overfitting the small training data, provided that an appropriate learning-based complexity measure is included in the cost function.

In continuation of our research the following future works may further improve our results.

- As mentioned in Chapter 2, in FIR learning with RBFN's, one can assume a fixed set of locations for the centers and based on that obtain tighter bounds for the sample of complexity. This will limit the results to the particular choice of centers but give more specific and less conservative bounds.

- A new algorithm based on the empirical risk minimization algorithm that is introduced in [33] and [34] has resulted in more practical search methods for learning with i.i.d. data. This algorithm, called "canonical smooth estimation", is an extension of the empirical risk minimization. The algorithm forms an empirical ϵ-cover set rather than the probability based one. The sample complexity of this algorithm has been directly calculated based on the sample complexity of the empirical risk minimization. This suggests that it might be possible to use the results of this thesis to calculate the sample complexity of the canonical smooth estimation, which is more practical than the empirical risk minimization algorithm.

- In order to further evaluate the performance of the introduced cost functions in

dealing with the overfitting problem, one can perform many modeling simulations with different training and testing sets.

- More efficient non-smooth optimization algorithms might be used to minimize the cost functions used in the thesis. However, SIMPLEX and gradient-based algorithms have already been applied to the problem. The results indicated that the above methods are not as successful as the proposed EP methods are.

- From the results of Chapter 4, it seems that the prediction of future rows can be extended to more than five rows ahead. A set of simulations on predictions of more rows can determine how many more rows can be accurately predicted.

- It may be possible to use the results of Tong [22] on the geometric ergodicity and geometric strong mixing of nonlinear systems, to create results similar to the ones introduced for SNN's. A careful formulation of the problem may result in a set of sufficient conditions on the weights of a RBFN that guarantees the above properties.

- As mentioned in Chapter 6, the use of actual neuromuscular blockade data taken from real patients in ARX neural modeling of the system can further assess the proposed algorithms.

Bibliography

[1] L.G. Valiant. A theory of learnable. *Comm. ACM*, pages pp. 1134–1142, 1984.

[2] M. Vidyasagar. *A Theory of Learning and Generalization.* Springer, 1997.

[3] A. Blum and R. Kannan. Training a 3-node neural metwork is NP-complete. *Proc. 1st workshop on Computational Learning Theory, San Mateo, CA*, pages 9–18, 1988.

[4] A. Blumer, A. Ehrenfeucht, D. Hussler, and M. Warmuth. Learnability and Vapnik-Chervonenkis dimension. *J. AMC*, 4, no. 36:929–965, 1989.

[5] A. Mokaddem. Mixing properties of polynomial autoregressive processes. *Ann. Inst. H. Poincare Probab. Statist.*, 26, no. 2:219–260, 1990.

[6] A.N. Kolmogorov and V.M. Tikhomirov. ϵ-Entropy and ϵ-capacity of sets in functional spaces. *Amer. Math. Soc. Transl.*, 17:pp. 227–364, 1961.

[7] A.R. Barron. Approximation and estimation bounds for artificial neural networks. *Machine Learning*, 14:115–133, 1994.

[8] B. Dasgupta, H. T. Siegelmann and E. D. Sontag. On the complexity of training neural networks with continuous activation functions. *IEEE Trans. Neural Networks*, 6:1490–1504, 1995.

[9] B. Whithing and A.W. Kelman. The modeling of drug response. *Clin. Sci.*, 59:311, 1980.

[10] C.G. Looney. *Pattern recognition using neural networks: theory and algorithms for engineers and scientists.* Oxford University Press, 1997.

[11] D. Aldous and U. Vazirani. A Markovian extension of Valiant's learning model. *Proc. 31th Annual IEEE Symp. on the Foundations of Comp. Sci.*, pages 392–396, 1990.

[12] D. Angluin. Computational learning theory: Survey and selected bibliography. *Proc. 24th ACM Symp. on Thy. of Computing*, pages 351–369, 1992.

[13] D. Anthony and N. Biggs. *Comutaional learning theory.* University Press, Cambridge, U.K., 1992.

[14] D. Anthony, N. Biggs and J. Shaw-Taylor. The learnability of formal concepts. *Proc. Third Workshop on Computational Learning Theory*, pages 246–257, 1990.

[15] D.A. Linkens, M. Mahfouf, M. Abood. Self-adaptive and self-organising control applied to nonlinear multivariable anesthesia: a comparative model-based study. *IEE Proceedings-D*, 139. no. 4:381–394, July 1992.

[16] D.S. Modha and E. Masry. Minimum complexity regression estimation with weakly dependent observation. *IEEE Trans. Information Theory*, 42, no. 6:2133–2145, Nov. 1996.

[17] E. Baum and D. Haussler. What size net gives valid generalization? *Neural Computation*, 1, no. 1:151–160, 1989.

[18] E. D. Sontag. Feedforward nets for interpolation and classification. *J. Comp. Sys. Sci.*, 45, no. 1:20–48, 1992.

[19] G. M. Benedek and A. Itai. Dominating distributions and learnability. *Proc. Fifth Workshop on Computational Learning Theory, ACM*, pages 253–264, 1991.

[20] G. M. Benedek and A. Itai. Learnability with respect to fixed distributions. *Theoretical Computer Sys.*, 86(2):377–390, 1991.

[21] H. E. Warren. Lower bounds for approximation by nonlinear manifolds. *Trans. AMS*, 133:167–178, 1968.

[22] H. Tong. *Non-linear time series.* Oxford Science Publications, 1990.

[23] H.J. Kushner. On the stability of processes defined by stochastic difference-differential equations. *J. Differential Equations*, 4, no. 3:424–443, 1968.

[24] H.J. Kushner. *Stochastic Stability, in Lecture Notes in Math.* Springer, New York, 1972.

[25] J. Mason and C. Kambhampati. Predictive control of a mixing tank using radial basis function networks. *Proc. 35th Conf. Decision and Cont.*, pages 478–479, 1996.

[26] J.P. La Salle. Stability theory for difference equations. *MAA Studies in Mathematics, American Math. Assoc.*, pages 1–31, 1977.

[27] K. Najarian, G.A. Dumont and M.S. Davies. A learning-theory-based training algorithm for variable-structure dynamic neural modeling. *Proc. Inter. Joint Conf. Neural Networks (IJCNN99)*, 1999.

[28] K. Najarian, G.A. Dumont, M.S. Davies, I. Motabar. Complexity Control of Neural Networks Using Learning Theory, Part I: Theory. *IASTED Conf. on Neural Networks (NN'2000)*, May, 2000.

[29] K. Najarian, G.A. Dumont, M.S. Davies, I. Motabar. Complexity Control of Neural Networks Using Learning Theory, Part II: Application in Minimum-Complexity Neural Modeling of Two-Dimensional Scanning System. *IASTED Conf. on Neural Networks (NN'2000)*, May, 2000.

[30] K. Najarian, G.A. Dumont, M.S. Davies, N. E. Heckman. Neural ARX Models and PAC Learning. *Springer's Lecture Notes in Artificial Intelligence Series*, 2000.

[31] K. Najarian, G.A. Dumont, M.S. Davies, N.E. Heckman. Learning of FIR Models Under Uniform Distribution. *Proc. The American Control Conference, San Dieo, U.S.A. (ACC1999)*, pages 864–869, June 1999.

[32] K. Najarian, Guy A. Dumont, and Michael S. Davies. PAC learning in Nonlinear FIR Models. *submitted to: Journal of Adaptive Control and Signal Processing*, 2000, (To appear).

[33] K.L. Bueschner and P.R. Kumar. Learning by canonical smooth estimation, Part I: Simultaneous estimation. *IEEE Trans on Auto. Control*, 42(4):545–556, April 1996.

[34] K.L. Bueschner and P.R. Kumar. Learning by canonical smooth estimation, Part II: Learning and choice od model complexity. *IEEE Trans on Auto. Control*, 42(4):557–569, April 1996.

[35] M. Iosifesco and R. Theodorescu. *Random processes and learning.* Springer-Verlog, 1969.

[36] M. Karpinski and A. J. Macintyre. Polynomial bounds for VC-dimension of sigmoidal and general Pfaffian neural networks. *J. Comp. Sys. Sci.*, 1996.

[37] M. Kearns and R. E. Shapire. Efficient distribution-free learning of probablistic concepts. *J. Comp. Sys.*, 48:464–497, 1994.

[38] M. Pottmann and D.E. Seborg. Identification of non-linear processes using reciprocal multiquadratic functions. *J. Proc. Cont.*, 2, no. 4:189–202, 1992.

[39] M.C. Campi and P.R. Kumar. Learning dynamical systems in a stationary environment. *Proc. 31th IEEE Conf. Decision and Control*, 16, no. 2:2308–2311, 1996.

[40] N.S.V. Rao. Nearest neighbor rules PAC-approximates feedforward networks. *IEEE International Conf. Neural Networks*, pages 108–113, June 1996.

[41] P. Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *Amer. Statistical Assoc. Math. Soc. Transactions*, 17:277–364, 1996.

[42] P. Bartlett, Fischer, Hoeffgen. Exploiting random walks for learning. *Proc. 7th ACM COLT*, pages 318–327, 1994.

[43] P. Doukham. *Mixing, properties and examples.* Springer-Verlog, 1985.

[44] P. Goldberg and M. Jerrum. Bounding VC-dimension of concept classes parametrized by real numbers. *Machine Learning*, 18:131–148, 1995.

[45] P. Koiran and E. D. Sontag. *Neural networks with quadratic VC-dimension, summary in Advances ion Neural Information Processing, 8.* MIT Press, Cambridge, MA, 1996.

[46] S. B. Holden and P. J. W. Rayner. Fast gradient based off-line training of multilayer perceptron. *IEEE Trans. Neural Networks*, 6, no. 2:368–380, March 1995.

[47] S. Chen, S.A. Billings and P.M. Grant. Recursive hybrid algorithm for non-linear system identification using radial basis function networks. *Int. J. Cont.*, 55, no. 5:1051–1070, 1992.

[48] T. M. Cover. *Capacity problem for linear machines, in "Pattern Recognition".* by , L. Kanak (Ed.), Thomson Book Co., 1952.

[49] V. N. Vapnik. *Statistical Learning Theory.* Wiley, New York, 1996.

[50] V. N. Vapnik and A. Ya. Chervonenkis. The necessary and sufficient conditions of the method of empirical risk minimization. *Pattern Recognition and Image Analysis*, 1(3):284–305, 1991.

[51] V.N. Vapnik and A.Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16, no. 2:264–280, 1971.

[52] W. Hoeffding. Probability inequalities for sum of bounded random variables. *Amer. Statistical Assoc. Math. Soc. Transactions*, 17:277–364, 1961.

[53] Z. Tang and G.J. Kohler. Deterministic global optimal FNN training algorithm. *Neural Networks*, 7, no. 2:301–311, 1994.