

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

NOTE TO USERS

The original manuscript received by UMI contains pages with indistinct, light, broken, and/or slanted print. Pages were microfilmed as received.

This reproduction is the best copy available

UMI

University of Alberta

**Aerodynamic Parameter Classification and Estimation
by Neural Networks**

By

Tak Keung (Simon) Wong



A thesis submitted to the Faculty of Graduate Studies and Research in
partial fulfillment of the requirements for the degree of Master of Science

In

**Applied Mathematics
Department of Mathematical Science**

Edmonton, Alberta

Fall 1998



National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-34436-3

Canada

University of Alberta

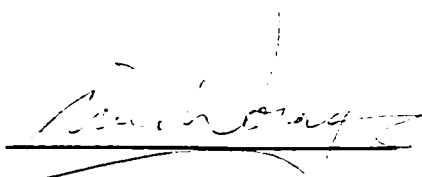
Library Release Form

Name of Author: Tak Keung (Simon) Wong
Title of Thesis: Aerodynamic Parameter Classification
and Estimation by Neural Networks
Degree: Master of Science
Year this degree Granted: 1998

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as herein before provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

Signature:



Permanent Address:

4/F, 28 Junction Road,
Kowloon, Hong Kong.


(852)-2716-2288


Date: 30/7/98.

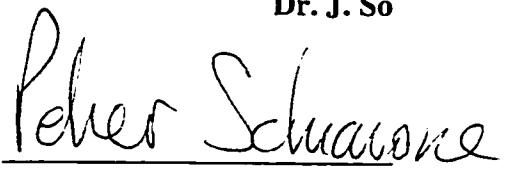
University of Alberta

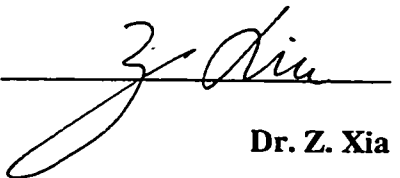
Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled Aerodynamic Parameter Classification and Estimation by Neural Networks submitted by Tak Keung (Simon) Wong in partial fulfillment of the requirements for the degree of Master of Science in Applied Mathematics.


Dr. Y. S. Wong


Dr. J. So


Dr. P. Schiavone


Dr. Z. Xia

Date: 30/7/98

Abstract

In this study, applications of neural networks to the modern flight flutter test are considered. These applications are using neural networks to classify and to extract parameters for a given data. Several suggestions involving wavelet transformation are given to improve the performance of these neural networks. The wavelet transformation tool is discussed. The ability to use the parallel computer system with neural networks is shown. Some basic neural network architectures are summarized and the training algorithms are discussed.

Acknowledgments

I am deeply grateful for encouragement and financial support for this thesis from my supervisor, Dr. Y. S. Wong, and the Mathematical Science Department of the University of Alberta.

I would like to thank Dr. Y. S. Wong for suggesting the topic for this thesis and for helping me all through these years.

I am especially indebted to Dr. B. H. K. Lee of the National Research Council of Canada for his financial support and for his interest in this research work.

Finally, I wish to thank God and my family for giving me this chance to study at the University of Alberta.

Table of Content

Abstract

Acknowledgements

List of Tables

List of Figures

Table of Content

Chapter 1:	Introduction	1
Chapter 2:	Neural Networks	4
	2.1 Neuron Model	4
	2.2 Network Architectures	8
	2.3 Learning rules	10
	2.4 Backpropagation	18
Chapter 3:	Wavelets and the Multi-resolution Method	28
	3.1 Wavelets	28
	3.2 Multi-resolution Analysis	30
Chapter 4:	Software Programs	42
	4.1 Sinon	42
	4.2 MATLAB toolbox	46
Chapter 5:	Computer Simulations	48
	5.1 Classification problem	48
	5.2 Parameter Extraction Problem I	58
	5.3 Parameter Extraction Problem II	66
	5.4 Comparison of two Simulating Programs	78
Chapter 6:	Conclusion	83
	References	87

List of tables

Table 2.1	Iterations of training	14
Table 3.1	Maximum error of the reconstruction with different numbers of coefficients	33
Table 3.2	Minimum Δf for $t = 0$ to 2 with different sampling rates	41
Table 3.3	Minimum Δf for $t = 0$ to 0.5 with different sampling rates	41
Table 5.1.1	Result of the classification problem	53
Table 5.1.2	Result of the classification problem	54
Table 5.1.3	Successful rate (in %) for 1000 testing samples	55
Table 5.1.4	Successful rate (in %) for 1000 testing samples	56
Table 5.1.5	Successful rate (in %) for 1000 testing samples	57

List of figures:

Figure 2.1 Linear function $f(n) = n$	4
Figure 2.2 Hard limit function $f(n) = 0$ if $n < 0$, $f(n) = 1$ if $n \geq 0$	5
Figure 2.3 Log-sigmoid function $f(n) = 1 / (1 + \exp(-n))$	5
Figure 2.4 Single-input neuron	6
Figure 2.51 Multiple-input neuron	7
Figure 2.52 Multiple-input neuron	7
Figure 2.6 Single-layer neural network with S neurons	8
Figure 2.7 Single-layer neural network with 3 neurons and 2 inputs	9
Figure 2.8 Three-layer feedforward network	10
Figure 2.9 Decision boundary	12
Figure 2.10 Relation of vectors and their visual patterns	16
Figure 2.11 Neural network for approximating function $g(x)$	25
Figure 3.1 Time-frequency plane	29
Figure 3.2 Example of an input signal using 128 points	32
Figure 3.3 Output of multi-resolution	32
Figure 3.4 Example of a signal using 512 points	33
Figure 3.5 Reconstruction using 45 wavelet coefficients	34
Figure 3.6 Reconstruction using 12 wavelet coefficients	34
Figure 3.7 Reconstruction using 25 wavelet coefficients	35
Figure 3.8 Noisy signal of the signal in figure 3.4	35
Figure 3.9 Signal reconstructed by 10 wavelet coefficients	35
Figure 3.10 Signal reconstructed by 20 wavelet coefficients	36
Figure 3.11 Signal reconstructed by 30 wavelet coefficients	36
Figure 3.12 Signal reconstructed by 40 wavelet coefficients	36
Figure 3.13 Signal reconstructed by 50 wavelet coefficients	37
Figure 3.14 Multi-resolution result of $x(t)$	37
Figure 3.15 First 50 coefficients in figure 3.12	38
Figure 3.16 Reconstruction using the first 25 coefficients and the error	38
Figure 3.17 Reconstruction using the next 25 coefficients and the error	38
Figure 3.18 Example signal using 512 points	39
Figure 3.19 Wavelet coefficients of the signal in figure 3.18	39
Figure 3.20 First 50 coefficients of the signal $x_1(t)$ when $\Delta f = 0.4$	40
Figure 3.21 First 100 coefficients of the signal $x_2(t)$ when $\Delta f = 5$	40
Figure 4.1 Notation of file calling	43
Figure 4.2 System of <code>sinon.exe</code>	43
Figure 5.1.1 Example of the function in C_1 for problem I	49
Figure 5.1.2 Example of the function in C_0 for problem I	49
Figure 5.1.3 Example of the function in C_1 with 10% noise level	50
Figure 5.1.4 Example of the function in C_0 with 10% noise level	50
Figure 5.1.5 Example of the function in C_1 with 20% noise level	50
Figure 5.1.6 Example of the function in C_0 with 20% noise level	51
Figure 5.1.7 Example of the function in C_1 with 30% noise level	51
Figure 5.1.8 Example of the function in C_0 with 30% noise level	51
Figure 5.1.9 Example of the function in C_1 with 40% noise level	52

Figure 5.1.10 Example of the function in C_0 with 40% noise level	52
Figure 5.1.11 Example of the function in C_1 with 50% noise level	52
Figure 5.1.12 Example of the function in C_0 with 50% noise level	53
Figure 5.1.13 Example of the function in C_1	57
Figure 5.1.14 Example of the function in C_0	57
Figure 5.2.1 Method 5.2.1	58
Figure 5.2.2 Example of the input of the neural network using 512 points	59
Figure 5.2.3 Relative error (in %) of the first output (i.e. a)	59
Figure 5.2.4 Relative error (in %) of the second output (i.e. b)	60
Figure 5.2.5 Method 5.2.2	60
Figure 5.2.6 Example of the input of the neural network	61
Figure 5.2.7 Relative error (in %) of the first output (i.e. a)	61
Figure 5.2.8 Relative error (in %) of the second output (i.e. b)	61
Figure 5.2.9 Method 5.2.3	62
Figure 5.2.10 First 30 points of wavecoeff	63
Figure 5.2.11 Next 30 points of wavecoeff	63
Figure 5.2.12 256 wavelet coefficients of $y(t)$	63
Figure 5.2.13 Relative error (in %) of the first output (i.e. a)	64
Figure 5.2.14 Sample input of network 2	64
Figure 5.2.15 Relative error (in %) of the second output (i.e. b)	65
Figure 5.3.1 System of method 5.3.1	66
Figure 5.3.2 Sample input of the neural network	67
Figure 5.3.3 Relative error (in %) of the first output (i.e. a_1)	67
Figure 5.3.4 Relative error (in %) of the second output (i.e. b_1)	68
Figure 5.3.5 Relative error (in %) of the third output (i.e. a_2)	68
Figure 5.3.6 Relative error (in %) of the fourth output (i.e. b_2)	68
Figure 5.3.7 System of improving performance, as mention in remarks 2	69
Figure 5.3.8 System of method 5.3.2	70
Figure 5.3.9 Relative error (in %) of the first output (i.e. a_1)	71
Figure 5.3.10 Relative error (in %) of the second output (i.e. b_1)	71
Figure 5.3.11 Relative error (in %) of the third output (i.e. a_2)	71
Figure 5.3.12 Relative error (in %) of the fourth output (i.e. b_2)	72
Figure 5.3.13 Relative error (in %) of the first output (i.e. a_1)	72
Figure 5.3.14 Relative error (in %) of the second output (i.e. b_1)	73
Figure 5.3.15 Relative error (in %) of the third output (i.e. a_2)	73
Figure 5.3.16 Relative error (in %) of the fourth output (i.e. b_2)	73
Figure 5.3.17 Relative error (in %) of the first output (i.e. a_1)	74
Figure 5.3.18 Relative error (in %) of the second output (i.e. b_1)	74
Figure 5.3.19 Relative error (in %) of the third output (i.e. a_2)	74
Figure 5.3.20 Relative error (in %) of the fourth output (i.e. b_2)	75
Figure 5.3.21 System of method 5.3.3	76
Figure 5.3.22 Relative error (in %) of the first output (i.e. a_1)	76
Figure 5.3.23 Relative error (in %) of the second output (i.e. b_1)	77
Figure 5.3.24 Relative error (in %) of the third output (i.e. a_2)	77
Figure 5.3.25 Relative error (in %) of the fourth output (i.e. b_2)	77

Figure 5.4.1 Sample input of the neural networks	78
Figure 5.4.2 Result using only 4 discrete points	78
Figure 5.4.3 Relative error of the output after 500 loops	79
Figure 5.4.4 Relative error of the output after 1000 loops	79
Figure 5.4.5 Relative error of the output after 2000 loops	80
Figure 5.4.6 Relative error of the output after 3500 loops	80
Figure 5.4.7 Relative error of the output after 1 loop	81
Figure 5.4.8 Relative error of the output after 10 loops	81
Figure 5.4.9 Relative error of the output after 20 loops	81
Figure 5.4.10 Relative error of the output after 40 loops	82

CHAPTER 1

Introduction

The neural networks discussed in this thesis are related to their biological counterparts. They consist of a finite number of highly connected elements called neurons. Neuron consists of weights, biases and transfer functions. A neural network is a way to connect every neuron to perform a useful task.

Neural networks have been trained to perform complex functions in various fields of application including aerospace, automotive, medicine, speech, securities and telecommunications. Researchers found that neural networks can be considered as potential tools for providing solutions to many practical problems. The focus upon this area has been substantial and subsequently so has the amount of investment in this field.

In engineering, Chu and Sze [1] discuss the potential applications of neural network to the Nava Theater Ballistic Missile Defense System. Barton and Himmelblau [2] discuss how internally recurrent neural networks can predict a key polymer product quality variable from an industrial polymerization reactor. An adaptive decision feedback recurrent neural equalizer for high-speed channel equalization has been introduced by Shin et al.[3]. In Medicine and Biology, an article [4] on the ALOPEX process relates the neural network approach to studies of animal and human visual systems and discusses applications in image processing and pattern recognition. Iwate et al. [4] describe a method of data compression for electrocardiograms for Holter monitors that could find applications in replacing a 24-h cassette recording with a memory card. Cios et al. [5] report on efforts to detect cardiac diseases from two-dimensional echocardiographic images. Holdaway et al. [6] report

on the classification of evoked somatosensory potentials from patients with severe head injuries. Hiraiwa et al. [7] report the case of neural networks for prearticulation in EEG signals. Kauffman et al. [8] use neural network tools for the analysis of bone fracture healing. In Mathematics and Statistics, Pao [12] shows that ordinary multiple regression could be viewed as a neural network. Specht [13] proposes that a probabilistic neural network could be used as a powerful statistical technique. Without neural network, the statistical technique [14] was not utilized because its memory and processing requirements are large [10]. More applications can be found in the Handbook of Neural Computing Applications [15], IEEE Transactions on Neural Networks and the IEEE International Conference on Neural Networks, etc.

In the modern flight flutter test, the aircraft is equipped with excitation systems. Using computers, the value of frequency and damping coefficients can be estimated from the data obtained from the systems. It should be pointed out that the success of a flight flutter test depends on accurately determining the frequency and damping from the given data [9]. In this thesis, a simple neural network is developed that will be used for classification and parameter extraction from the simulated flutter data. First, in order to determine the correct property for a given signal, we must classify whether the signal is increasing or decreasing. Secondly, we want to extract the frequency and damping coefficients from a class of simple signals. Thirdly, we want to extract the same coefficients from a complex signal (i.e. sum of two or more simple signals). Details for defining these signals and the neural network will be discussed later.

In Chapter 2, neural models, network architectures, learning rules and optimization methods in training are discussed. In Chapter 3, basic wavelet

transformation and some of its properties are shown. In Chapter 4, computer software programs developed and used in this project are reported. In Chapter 5, applications and results using neural networks are shown. Finally, the conclusion will be given in Chapter 6.

CHAPTER 2

Neural Networks

In this Chapter, we will first introduce several basic components of neural networks. Secondly, transfer functions, neurons, and single-layer and multiple-layer feedforward neural networks will be discussed. Lastly, learning rules, along with some examples will be given.

2.1 Neuron Model

A neuron is a basic unit of a neural network. It consists of input(s), weight(s), bias, transfer function and output. Weight w is a real number, which is multiplied by the input p . In a neuron, if the input p is a vector, then the weight w is also a vector. Bias b in a neuron is a real number. A transfer function is a function mapping the real numbers to the real numbers. As examples of transfer functions, we will list three of the most commonly used transfer functions. Let n be the input value, $\text{output} = f(n) = n$, is called the linear function.

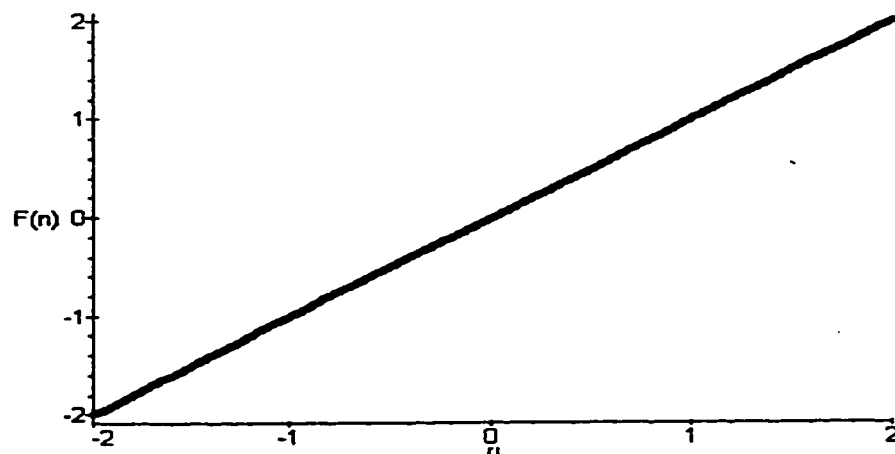


Figure 2.1 Linear function $f(n) = n$.

Output = $f(n) = 0$ if $n < 0$, Output = $f(n) = 1$ if $n \geq 0$, which is called the hard limit function.

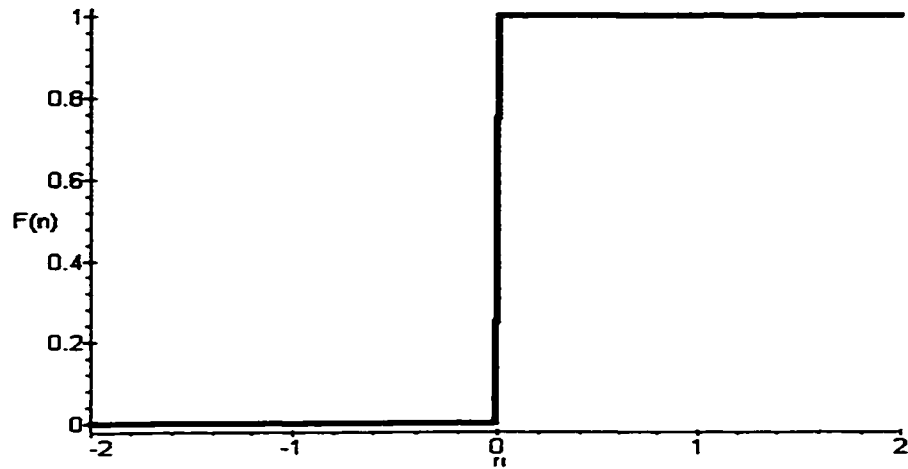


Figure 2.2 Hard limit function $f(n) = 0$ if $n < 0$, $f(n) = 1$ if $n \geq 0$.

Output = $f(n) = 1/(1+\exp(-n))$, which is called the log-sigmoid function.

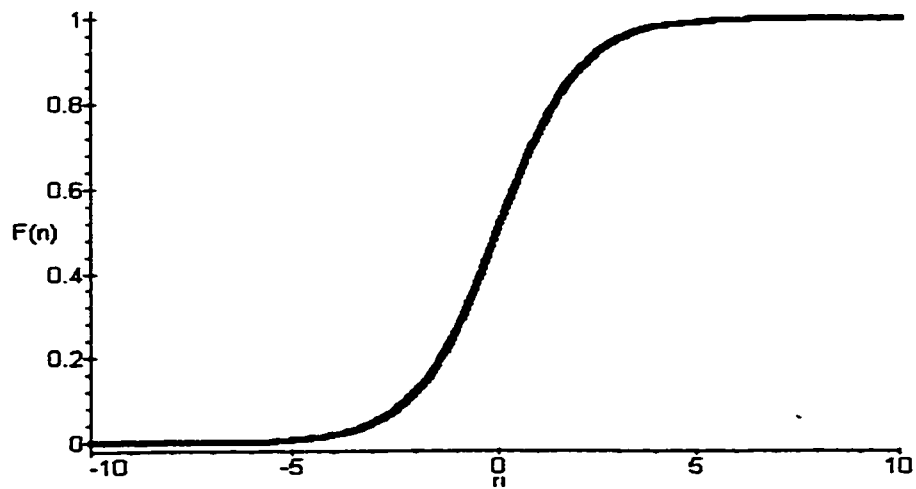


Figure 2.3 Log-sigmoid function $f(n) = 1/(1+\exp(-n))$.

Single-Input Neuron

The output of the single-input neuron consists of the scalar input p , the scalar weight w , the bias b and the transfer function f . The scalar input p is multiplied by the scalar weight w and the product is then added to the bias b , the result of which n is put into the transfer function f . So, the neuron output is calculated as $a = f(wp + b)$. In a particular problem, p is given, and the appropriate transfer function must be chosen; moreover, w and b must be adjusted by some learning rules so that the relation between input and output meets our specific goal.

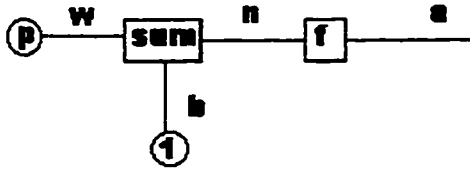


Figure 2.4 Single-input neuron.

Multiple-Input Neuron

The output of the R -input neuron consists of inputs $p_1, p_2, p_3, \dots, p_R$, weights w_1, w_2, \dots, w_R , the bias b and the transfer function f . The individual inputs $p_1, p_2, p_3, \dots, p_R$ are multiplied by the corresponding weights and are then added to the bias b , and the sum n is put into the transfer function f . So, the neuron output is calculated as $a = f(p_1w_1 + p_2w_2 + \dots + p_Rw_R + b)$. As in the case of the single-input neuron, weights and bias will be adjusted by some learning rules. The appropriate transfer function must be chosen.

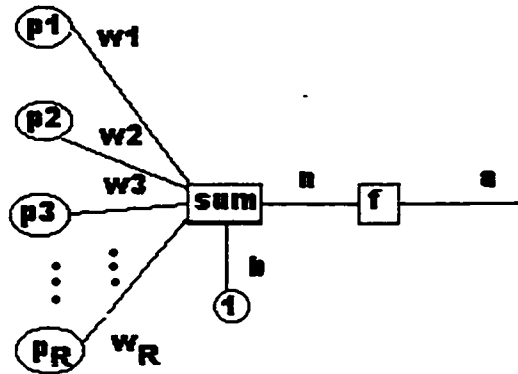


Figure 2.5.1 Multiple-input neuron.

To simplify the notation, let \mathbf{p} be the input vector and \mathbf{w} be the weight matrix; then the above diagram can be simplified as:

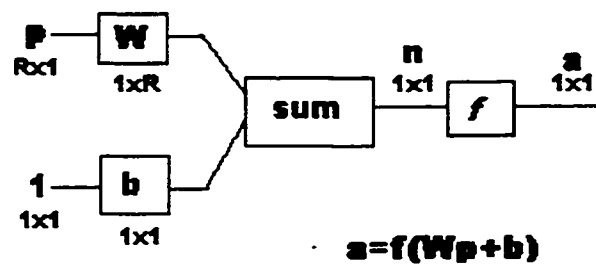


Figure 2.5.2 Multiple-input neuron

Since one neuron may not be sufficient for a particular problem, we will need two or more neurons operating in parallel to reach our goal. This arrangement is called a “layer”.

2.2 Network Architectures

Single-Layer Network

A single-layer network of S neurons and R inputs consists of inputs $p_1, p_2, p_3, \dots, p_R$, weights $w_{11}, w_{12}, \dots, w_{1R}, w_{21}, w_{22}, \dots, w_{2R}, \dots, w_{S1}, w_{S2}, \dots, w_{SR}$, biases b_1, b_2, \dots, b_S and transfer functions f_1, f_2, \dots, f_S . We can use the matrix notation such that the input vector $\mathbf{p} = [p_1 \ p_2 \ p_3 \ \dots \ p_R]^T$, bias vector $\mathbf{b} = [b_1 \ b_2 \ b_3 \ \dots \ b_S]^T$, transfer function vector $\mathbf{f} = [f_1 \ f_2 \ f_3 \ \dots \ f_S]^T$ and weight matrix \mathbf{W} , where:

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1R} \\ w_{21} & w_{22} & \dots & w_{2R} \\ \vdots & \vdots & \dots & \vdots \\ w_{S1} & w_{S2} & \dots & w_{SR} \end{bmatrix}$$

Multiply the weight matrix \mathbf{W} by the input vector \mathbf{p} , add the product to the bias vector \mathbf{b} , then take the result \mathbf{n} to the transfer function \mathbf{f} to get the output vector \mathbf{a} . So, the output vector $\mathbf{a} = \mathbf{f}(\mathbf{W}\mathbf{p} + \mathbf{b})$ is the output of the S -neuron, R -input, and one-layer network.

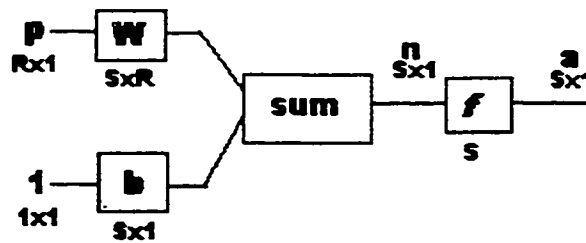


Figure 2.6 Single layer neural network with S neurons.

Let $S=3$ and $R=2$, which means that this network has two inputs and three neurons.

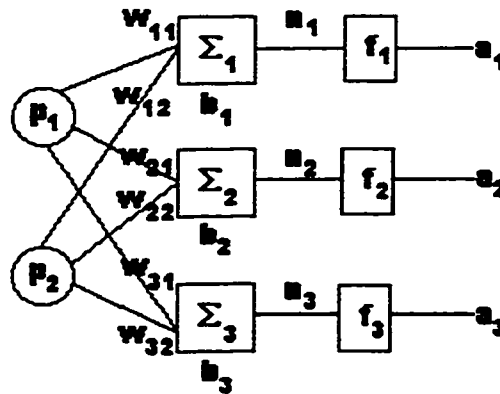


Figure 2.7 Single-layer neural network with 3 neurons and 2 inputs.

Multiple Layer of Neurons

A multiple layer of neurons is a neural network with several layers. Each layer has its own weight matrix \mathbf{W} , bias vector \mathbf{b} , input vector \mathbf{p} , transfer function f and output vector \mathbf{a} . Let \mathbf{W}^j be the weight matrix of the j -th layer, \mathbf{b}^j be the bias vector of the j -th layer, \mathbf{p}^j be the input vector of the j -th layer, f^j be the transfer function of the j -th layer and \mathbf{a}^j be the output vector of the j -th layer. Note that the input of the second layer is the output of the first layer.

As in the single-layer case, $\mathbf{a}^1 = f^1 (\mathbf{W}^1 \mathbf{p} + \mathbf{b}^1)$ for the first layer, but for the second layer, $\mathbf{a}^2 = f^2 (\mathbf{W}^2 \mathbf{a}^1 + \mathbf{b}^2)$ and so on. So, for the three-layer network $\mathbf{a}^3 = f^3 (\mathbf{W}^3 \mathbf{a}^2 + \mathbf{b}^3)$ or $\mathbf{a}^3 = f^3 (\mathbf{W}^3 f^2 (\mathbf{W}^2 f^1 (\mathbf{W}^1 \mathbf{p} + \mathbf{b}^1) + \mathbf{b}^2) + \mathbf{b}^3)$.

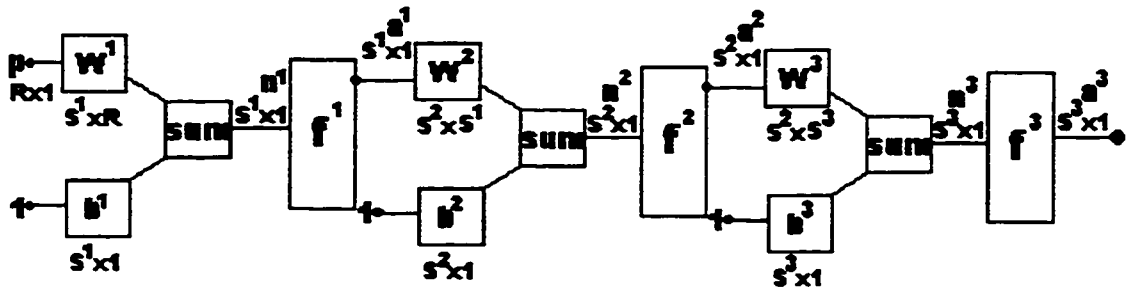


Figure 2.8 Three-layer feedforward network.

A layer whose output is the network output is called an output layer. The other layers are called hidden layers. In this output equation, the only given information is p (input vector), so we have to choose the weight matrices and bias vectors. To determine W and b , we need to use some training rules. Note that the network presented in figure 2.8 is called a feedforward network. Another type of neural network is the recurrent network, which will not be discussed in this chapter. The term “feedforward” means that information flows in one direction only. A recurrent network [17] is a network with feedback; some of its outputs are connected to its inputs. Recurrent networks are potentially more powerful than feedforward networks and they can exhibit temporal behavior [16].

2.3 Learning rules

A learning rule (training algorithm) [16] is a procedure for modifying the weights and biases for a network. The purpose of the learning rule is to train the network to perform certain tasks. There are three categories of neural network learning rules, namely supervised learning, unsupervised learning and reinforcement learning.

Supervised Learning

In this category, the learning rule is provided with a set of examples of proper network behavior; we call it a training set. Let $\{p_1, t_1\}, \{p_2, t_2\}, \dots, \{p_Q, t_Q\}$ be Q examples of proper network behavior. Note that p_i is an input to the network and t_i is the corresponding correct output. We apply the input to the network to get the network output and adjust the weights and biases after comparing the network output with the correct output. The perceptron learning rule and the supervised Hebbian rule are examples of supervised learning rules.

Unsupervised learning

In this category of learning rules, the weights and biases are modified in response to network inputs only. Normally this method is used when the correct outputs are not available, or in other words, when t_i is not available in the training set. One example is the unsupervised Hebbian rule that is used for pattern recognition problems.

Reinforcement learning

Reinforcement learning [20] is similar to supervised learning. The algorithm is given a grade, which is a measure of the network performance over some sequence of inputs. This kind of learning method is frequently applied to control system applications.

Examples of supervised learning

First we will introduce the perceptron learning rule [19]. Let $f(n)$ be the hard limit transfer function. The general perceptron network output is equal to $f(Wp + b)$, where W is the weight matrix, p is the input vector and b is the bias vector. Consider the single-neuron perceptron with two inputs and one output. We have output $= f(Wp + b) = f(w_{11}p_1 + w_{12}p_2 + b)$. From the behavior of function f , the boundary (decision boundary) is determined by $w_{11}p_1 + w_{12}p_2 + b = 0$. As an example, let $w_{11} = 2$, $w_{12} = 3$ and $b = 4$. We then have a line $2p_1 + 3p_2 + 4 = 0$ in the input space. Output = 0 when $2p_1 + 3p_2 + 4 < 0$, and output = 1 otherwise.

If $p_1 = 1$ and $p_2 = 1$, then output $= f(2+3+4) = f(9) = 1$.

If $p_1 = 0$ and $p_2 = -3$, then output $= f(2(0)+3(-3)+4) = f(-5) = 0$.

The grey area in figure 2.9 is provided for output = 1.

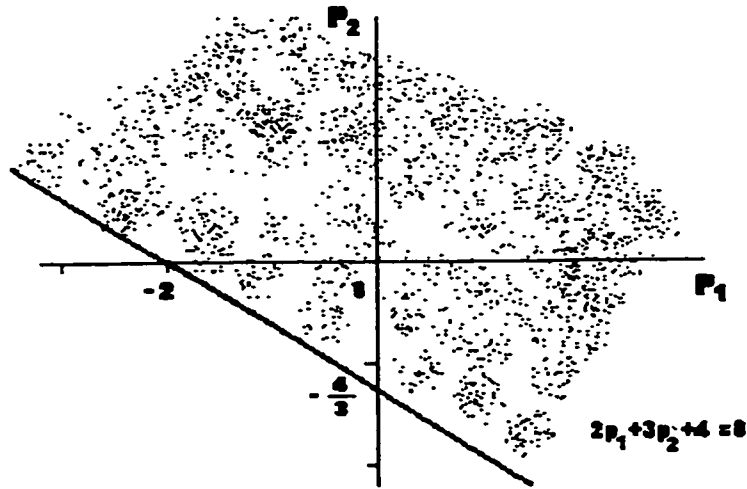


Figure 2.9 Decision boundary.

For the multiple-neuron perceptron, we will consider the S-neuron perceptron with R inputs, in which case the output $= f(Wp + b)$; however, now the weight matrix W is $S \times R$, the input vector is $R \times 1$, the bias vector is $S \times 1$ and the output vector is $S \times 1$. Note

that we can classify 2^S possible categories in this case. The perceptron learning rule is given by

$$\mathbf{W}^{\text{new}} = \mathbf{W}^{\text{old}} + \mathbf{e} \mathbf{p}^T \text{ and } \mathbf{b}^{\text{new}} = \mathbf{b}^{\text{old}} + \mathbf{e},$$

where \mathbf{e} = target output - network output [16]. Suppose we have a training set

$\{\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \dots, \{\mathbf{p}_Q, \mathbf{t}_Q\}\}$ with Q elements, where \mathbf{p}_i is the input vector and \mathbf{t}_i is the corresponding target output. Apply the learning rule for each \mathbf{p}_i repeatedly until all the inputs provide the correct output. The OR gate is an example to which we can apply the learning rule. Here $\{\mathbf{p}_1 = [0 \ 0]^T, \mathbf{t}_1 = 0\}$, $\{\mathbf{p}_2 = [1 \ 0]^T, \mathbf{t}_2 = 1\}$, $\{\mathbf{p}_3 = [0 \ 1]^T, \mathbf{t}_3 = 1\}$, $\{\mathbf{p}_4 = [1 \ 1]^T, \mathbf{t}_4 = 1\}$. In this case, two inputs are given and one output should be produced. We can use the single neuron perceptron. The output = $f(\mathbf{W}\mathbf{p} + \mathbf{b}) = f(w_{11}p_1 + w_{12}p_2 + b)$ where $f(n) = 1$ if $n \geq 0$ and $f(n) = 0$ if $n < 0$.

We apply a learning rule with the initial guess, say w_{11} , w_{12} , and b are first set to zero.

Iteration	input	Target output	Network output	Error	w_{11}	w_{12}	b	mod
0	—	—	—	—	0	0	0	on
1	$[0\ 0]^T$	$t_1 = 0$	$f(0(0)+0(0)+0)=1$	$0-1=-1$	0	0	-1	on
1	$[1\ 0]^T$	$t_2 = 1$	$f(0(1)+0(0)-1)=0$	$1-0=1$	1	0	0	on
1	$[0\ 1]^T$	$t_3 = 1$	$f(1(0)+0(1)+0)=1$	$1-1=0$	1	0	0	on
1	$[1\ 1]^T$	$t_4 = 1$	$f(1(1)+0(1)+0)=1$	$1-1=0$	1	0	0	on
2	$[0\ 0]^T$	$t_1 = 0$	$f(1(0)+0(0)+0)=1$	$0-1=-1$	1	0	-1	on
2	$[1\ 0]^T$	$t_2 = 1$	$f(1(1)+0(0)-1)=1$	$1-1=0$	1	0	-1	on
2	$[0\ 1]^T$	$t_3 = 1$	$f(1(0)+0(1)-1)=0$	$1-0=1$	1	1	0	on
2	$[1\ 1]^T$	$t_4 = 1$	$f(1(1)+1(1)+0)=1$	$1-1=0$	1	1	0	on
3	$[0\ 0]^T$	$t_1 = 0$	$f(1(0)+1(0)+0)=1$	$0-1=-1$	1	1	-1	on
3	$[1\ 0]^T$	$t_2 = 1$	$f(1(1)+1(0)-1)=1$	$1-1=0$	1	1	-1	on
3	$[0\ 1]^T$	$t_3 = 1$	$f(1(0)+1(1)-1)=1$	$1-1=0$	1	1	-1	on
3	$[1\ 1]^T$	$t_4 = 1$	$f(1(1)+1(1)-1)=1$	$1-1=0$	1	1	-1	on
4	$[0\ 0]^T$	$t_1 = 0$	$f(1(0)+1(0)-1)=0$	$0+0=0$	1	1	-1	off
4	$[1\ 0]^T$	$t_2 = 1$	$f(1(1)+1(0)-1)=1$	$1-1=0$	1	1	-1	off
4	$[0\ 1]^T$	$t_3 = 1$	$f(1(0)+1(1)-1)=1$	$1-1=0$	1	1	-1	off
4	$[1\ 1]^T$	$t_4 = 1$	$f(1(1)+1(1)-1)=1$	$1-1=0$	1	1	-1	off

Table 2.1 Iterations of the training

We can stop here, since there is no further modification in the fourth iteration, and the final result is $w_{11} = 1$, $w_{12} = 1$ and $b = -1$.

Another example of the supervised learning rule is the Hebbian learning rule [16]. Now consider the case in which the neural network output is equal to Wp , that is, when the bias is equal to zero, the transfer function of this network is a linear function. This network is called a linear associator, which is a type of associative memory. The task of an associative memory is to learn Q pairs of prototype input/output vectors. The Hebbian learning rule is given by $W^{new} = W^{old} + t_q p_q^T$. Note that if the input prototype vectors are orthonormal, the Hebbian learning rule will produce the correct output for any input (see below). If the input prototype vectors are not orthonormal, then an error is introduced. Let p_i be the input vectors and t_i be the target vectors, where $i = 1$ to Q . Also, let the initial weight matrix be a zero matrix. Applying each data pair to the rule, then $W^1 = 0 + t_1 p_1^T$, $W^2 = W^1 + t_2 p_2^T = t_1 p_1^T + t_2 p_2^T$, $W^3 = W^2 + t_3 p_3^T = t_1 p_1^T + t_2 p_2^T + t_3 p_3^T$, ..., $W^Q = W^{Q-1} + t_Q p_Q^T = t_1 p_1^T + t_2 p_2^T + \dots + t_Q p_Q^T = W$.

So, for any input p_k ,

$$output = \left(\sum_{i=1}^Q t_i p_i^T \right) p_k = \sum_{i=1}^Q t_i (p_i^T p_k) \quad (2.1)$$

Assume the norm of input p_i is 1, then,

$$output = \sum_{i=1}^Q t_i (p_i^T p_k) = t_k (p_k^T p_k) + \sum_{i \neq k} t_i (p_i^T p_k) = t_k + \sum_{i \neq k} t_i (p_i^T p_k) \quad (2.2)$$

Note that if $(p_i^T p_k) = 0$ for $i \neq k$ (orthonormal), then output = t_k .

If $(p_i^T p_k) \neq 0$ for $i \neq k$, then the error = target output - network output. So,

$$error = \sum_{i \neq k} t_i (p_i^T p_k) \quad (2.3)$$

In this case, we need to find a method to minimize the error. One of the methods is the pseudoinverse method. Recall that output $\mathbf{a}_q = \mathbf{W}\mathbf{p}_q$, in which case we have $(error \mathbf{r}_q)^2 = \|\mathbf{t}_q - \mathbf{W}\mathbf{p}_q\|^2$ for each $q=1,2,...,Q$. Hence, we want to minimize the sum of $(error \mathbf{r}_q)^2$ for all q . Let

$$F(W) = \sum_{q=1}^Q \|\mathbf{t}_q - \mathbf{W}\mathbf{p}_q\|^2 \quad (2.4)$$

The pseudoinverse method suggests picking $\mathbf{W} = \mathbf{T}\mathbf{P}^+$ [16], where $\mathbf{T} = [\mathbf{t}_1 \mathbf{t}_2 \mathbf{t}_3 \dots \mathbf{t}_Q]$, $\mathbf{P} = [\mathbf{p}_1 \mathbf{p}_2 \mathbf{p}_3 \dots \mathbf{p}_Q]$ and $\mathbf{P}^+ = (\mathbf{P}^T \mathbf{P})^{-1} \mathbf{P}^T$. For simplicity, assume that the input patterns are orthonormal. Using the Hebbian rule to find the weight matrix will produce no error. For example, suppose we want to recognize two patterns,

$\mathbf{p}_1 = \sqrt{6}/6 [1 -1 -1 1 1 -1]^T$, $t_1 = 1$ and $\mathbf{p}_2 = \sqrt{6}/6 [1 1 -1 1 -1 1]^T$, $t_2 = -1$.

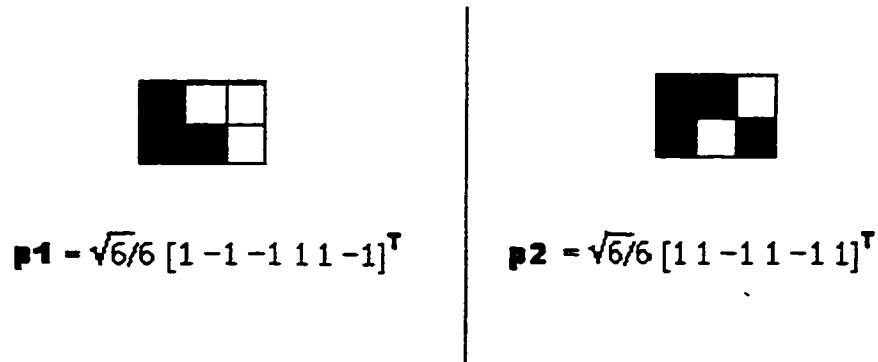


Figure 2.10 Relation of vectors and their visual patterns.

These two patterns are orthonormal since $\mathbf{p}_1^T \mathbf{p}_2 = 0$, $\mathbf{p}_1^T \mathbf{p}_1 = 1$ and $\mathbf{p}_2^T \mathbf{p}_2 = 1$. The Hebbian rule gives:

$$\begin{aligned}
W &= TP^+ = TP = \frac{1}{\sqrt{6}} \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & -1 & -1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix} \\
&= \frac{1}{\sqrt{6}} \begin{bmatrix} 0 & -2 & 0 & 0 & 2 & -2 \end{bmatrix} \quad (2.5)
\end{aligned}$$

Testing the network with p_1 we have:

$$Wp_1 = TPp_1 = \frac{1}{6} \begin{bmatrix} 0 & -2 & 0 & 0 & 2 & -2 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \\ 1 \\ -1 \end{bmatrix} = \frac{1}{6} (0 + 2 + 0 + 0 + 2 + 2) = 1 = t_1 \quad (2.6)$$

Testing the network with p_2 we have:

$$\begin{aligned}
Wp_2 &= TPp_2 = \frac{1}{6} \begin{bmatrix} 0 & -2 & 0 & 0 & 2 & -2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \\ -1 \\ 1 \end{bmatrix} \\
&= \frac{1}{6} (0 - 2 + 0 + 0 - 2 - 2) = -1 = t_2 \quad (2.7)
\end{aligned}$$

Now the network recognizes these two patterns.

2.4 Backpropagation

Backpropagation [18] is one of the supervised training methods commonly used for multiple-layer networks. We consider the M-layer network. Let M be the number of layers, W^i be the weighted matrix for the i th layer, a^i be the output vector of the i th layer, b^i be the bias vector of the i th layer and f^i be the transfer function of the i th layer. Then the network output $a = a^{m+1} = f^{m+1}(W^{m+1}a^m + b^{m+1})$ for $m = 0, 1, 2, \dots, M-1$, and a^0 is equal to the input vector p . To determine the performance of the network, we have to find a quantitative measure of network performance; we call it the performance index. Consider the function $F(W) = \sum \|t_q - Wp_q\|$ that was used in the pseudoinverse method. This is an example of a performance index, which is the sum of squares of the errors for all inputs. When the performance index is large, then the network performs poorly; similarly, when the performance index is small, then the network performs well. Now, let's denote the performance index by $F(x)$, which is assumed to be an analytic function, that is to say, all of its derivatives exist and it can be expressed as a convergent power series. To make it easier to handle, we will use a finite number from terms of the Taylor series expansion to approximate the performance index. In this case, the neural network performance index will be a function of all the network parameters (weights and biases). Recall that the Taylor series expansion for functions of n variables in matrix form at the point x' is given by:

$$F(x) = F(x') + \nabla F(x')^T |_{x=x'} (x - x') + \frac{1}{2} (x - x')^T \nabla^2 F(x) |_{x=x'} (x - x') + \dots \quad (2.9)$$

where $\nabla F(x) = \left[\frac{\partial}{\partial x_1} F(x) \quad \frac{\partial}{\partial x_2} F(x) \quad \dots \quad \frac{\partial}{\partial x_n} F(x) \right]^T$ is called the gradient

$$\text{and } \nabla^2 F(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2}{\partial x_1^2} F(\mathbf{x}) & \frac{\partial^2}{\partial x_1 \partial x_2} F(\mathbf{x}) & \cdots & \frac{\partial^2}{\partial x_1 \partial x_n} F(\mathbf{x}) \\ \frac{\partial^2}{\partial x_2 \partial x_1} F(\mathbf{x}) & \frac{\partial^2}{\partial x_2^2} F(\mathbf{x}) & \cdots & \frac{\partial^2}{\partial x_2 \partial x_n} F(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial x_n \partial x_1} F(\mathbf{x}) & \frac{\partial^2}{\partial x_n \partial x_2} F(\mathbf{x}) & \cdots & \frac{\partial^2}{\partial x_n^2} F(\mathbf{x}) \end{bmatrix} \text{ is the Hessian Matrix.}$$

Since we want to find the minimum (optimum) point of the performance index, let's define the strong minimum, weak minimum and the global minimum as follows.

Definition 1: \mathbf{x}' is a strong minimum of $F(\mathbf{x})$ if $\exists \delta > 0$, such that $F(\mathbf{x}) < F(\mathbf{x} + \Delta \mathbf{x}) \forall \Delta \mathbf{x}$ with $\delta > \|\Delta \mathbf{x}\| > 0$.

Definition 2: \mathbf{x}' is a weak minimum of $F(\mathbf{x})$ if \mathbf{x}' is not a strong minimum and $\exists \delta > 0$, such that $F(\mathbf{x}) \leq F(\mathbf{x} + \Delta \mathbf{x}) \forall \Delta \mathbf{x}$ with $\delta > \|\Delta \mathbf{x}\| > 0$.

Definition 3: \mathbf{x}' is a global minimum of $F(\mathbf{x})$ (unique) if $F(\mathbf{x}) < F(\mathbf{x} + \Delta \mathbf{x}) \forall \Delta \mathbf{x} \neq 0$.

Note that the first order necessary condition for \mathbf{x}' to be a minimum point is that the gradient must equal to zero, that is:

$$\nabla F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}'} = \left[\frac{\partial}{\partial x_1} F(\mathbf{x}) \quad \frac{\partial}{\partial x_2} F(\mathbf{x}) \quad \cdots \quad \frac{\partial}{\partial x_n} F(\mathbf{x}) \right]^T \bigg|_{\mathbf{x}=\mathbf{x}'} = 0 \quad (2.10)$$

Points that satisfy this condition are called stationary points. If we have a stationary point \mathbf{x}' of $F(\mathbf{x})$, then $\nabla F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}'} = 0$ and

$$F(\mathbf{x}) = F(\mathbf{x}' + \Delta \mathbf{x}) = F(\mathbf{x}') + \frac{1}{2}(\mathbf{x} - \mathbf{x}')^T \nabla^2 F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}'} (\mathbf{x} - \mathbf{x}') + \cdots \quad (2.11)$$

Taking the first two terms to approximate $F(\mathbf{x})$, we have:

$$F(\mathbf{x}) = F(\mathbf{x}' + \Delta \mathbf{x}) = F(\mathbf{x}') + \frac{1}{2} \Delta \mathbf{x}^T \nabla^2 F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}'} \Delta \mathbf{x} \quad (2.12)$$

So, \mathbf{x}' is a strong minimum if:

$$\Delta \mathbf{x}^T \nabla^2 F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}'} \Delta \mathbf{x} > 0 \quad \forall \Delta \mathbf{x} \neq 0 \quad (2.13)$$

This means that $\nabla^2 F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}'}$ is positive definite. Similarly, if \mathbf{x}' is a weak minimum then

$$\Delta \mathbf{x}^T \nabla^2 F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}'} \Delta \mathbf{x} \geq 0 \quad \forall \Delta \mathbf{x} \neq 0 \quad (2.14)$$

According to linear algebra, a real symmetric matrix A is positive definite if all its eigenvalues are positive, and A is a positive semi-definite if all its eigenvalues are nonnegative. To summarize these results, the necessary conditions for \mathbf{x}' to be a strong minimum point of $F(\mathbf{x})$ are $\nabla F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}'} = 0$ and $\nabla^2 F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}'}$ is positive semi-definite. The sufficient conditions for \mathbf{x}' to be a strong minimum point of $F(\mathbf{x})$ are $\nabla F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}'} = 0$ and $\nabla^2 F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}'}$ is positive definite. To optimize $F(\mathbf{x})$, a number of methods are commonly used. First, consider the steepest descent method. Let \mathbf{x} be the vector that includes all the parameters in the network (weights and biases). So, $\mathbf{x} = [\mathbf{w}_{11} \ \mathbf{w}_{12} \ \dots \ \mathbf{w}_{1R} \ \dots \ \mathbf{w}_{S1} \ \mathbf{w}_{S2} \ \dots \ \mathbf{w}_{SR} \ \mathbf{b}_1 \ \mathbf{b}_2 \ \dots \ \mathbf{b}_S]^T$, where w_{ij} is the entry of the weight matrix \mathbf{W} at i th row and j th column and b_i is the entry of the bias vector \mathbf{b} at i th row. The steepest descent algorithm gives $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_k}$, the gradient of F at \mathbf{x}_k can be calculated from $F(\mathbf{x})$, but we have to choose a suitable learning rate α_k such that the algorithm is stable. The maximum learning rate is, in general, not possible to predict for arbitrary functions. However, for some special functions such as quadratic functions, we can set an upper limit. The quadratic function is a function F of vector \mathbf{x} such that it can be expressed in the form

$F(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{d}^T \mathbf{x} + c$, where \mathbf{A} is symmetric. For example, let $F(\mathbf{x}) = 7x_1^2 - x_1x_2 + 7x_2^2 + 5x_1 + 5x_2$, then

$$F(x) = \frac{1}{2} x^T \begin{bmatrix} 14 & -8 \\ -8 & 14 \end{bmatrix} x + [5 \ 5]x \quad \text{where } x = [x_1 \ x_2]^T \quad (2.15)$$

$$\nabla F(x) = \begin{bmatrix} 14 & -8 \\ -8 & 14 \end{bmatrix} x + \begin{bmatrix} 5 \\ 5 \end{bmatrix} \quad \text{and} \quad \nabla^2 F(x) = \begin{bmatrix} 14 & -8 \\ -8 & 14 \end{bmatrix} \quad (2.16)$$

This kind of function has a number of special properties. First, the gradient $\nabla F(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{d}$; second, the Hessian matrix $\nabla^2 F(\mathbf{x}) = \mathbf{A}$; and third, the directional derivatives are between the minimum eigenvalue of \mathbf{A} and the maximum eigenvalue of \mathbf{A} . With these properties, the constant stable learning rate α is given by $\alpha < 2/\lambda_{\max}$, where λ_{\max} is the maximum eigenvalue of \mathbf{A} . Using the constant stable learning rate is easy and it is guaranteed to converge. If there is a great difference in magnitude between the largest and smallest eigenvalue, the constant stable learning rate results in a slow convergent rate of the algorithm. Select the learning rate to minimize $F(\mathbf{x})$ with respect to α_k such that $F(\mathbf{x}_k + \alpha_k \mathbf{p})$ is minimized. For a quadratic function $F(\mathbf{x})$,

$$\frac{\partial}{\partial \alpha_k} F(x_k + \alpha_k p_k) = \nabla F(x)^T \Big|_{x=x_k} p_k + \alpha_k p_k^T \nabla^2 F(x)^T \Big|_{x=x_k} p_k \quad (2.17)$$

Because of the first derivative condition, we have to set it equal to zero. Hence

$$\alpha_k = \frac{\nabla F(x)^T \big|_{x=x_k} p_k}{p_k^T \nabla^2 F(x) \big|_{x=x_k} p_k} \quad (2.18)$$

Note that for quadratic functions, the Hessian matrix is independent of k . The second optimization method commonly used is the Newton's method; as in the case of the steepest descent algorithm, Newton's method is based on Taylor series expansion. But this time $F(\mathbf{x})$ is approximated by the first three terms of the series, that is

$$F(x_{k+1}) = F(x_k + \Delta x) \approx F(x_k) + \nabla F(x)^T \big|_{x=x_k} \Delta x_k + \frac{1}{2} \Delta x_k^T \nabla^2 F(x) \big|_{x=x_k} \Delta x_k \quad (2.19)$$

Newton's method suggests $\mathbf{x}_{k+1} = \mathbf{x}_k - \nabla^2 F(\mathbf{x})^{-1} \big|_{\mathbf{x}=\mathbf{x}_k} \nabla F(\mathbf{x}) \big|_{\mathbf{x}=\mathbf{x}_k}$. From basic numerical analysis or calculus, we know that Newton's method usually produces faster convergence (quadratic rate) than the steepest descent method (linear rate). However, with Newton's method, the method is not guaranteed to converge. Recall that convergence in Newton's method depends on the function and the initial guess. To compute and store the Hessian matrix and its inverse requires significant computing time and memory. Note that if $F(\mathbf{x})$ is a quadratic function, then Newton's method uses a finite number of iterations to get to the minimum point exactly. This property is called quadratic termination. To handle the divergence problem, we can use steepest descent steps when divergence begins to occur. A powerful method, which has the quadratic termination property but does not require use of the second derivative is the method of conjugate gradient. As before, let $F(\mathbf{x})$ be a quadratic function. We have $F(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{d}^T \mathbf{x} + c$, $\nabla F(\mathbf{x}) = \mathbf{A} \mathbf{x} + \mathbf{d}$ and $\nabla^2 F(\mathbf{x}) = \mathbf{A}$. Note that $\nabla F(\mathbf{x}) \big|_{\mathbf{x}=\mathbf{x}_{k+1}} - \nabla F(\mathbf{x}) \big|_{\mathbf{x}=\mathbf{x}_k} = \mathbf{A} \mathbf{x}_{k+1} + \mathbf{d} - (\mathbf{A} \mathbf{x}_k + \mathbf{d}) = \mathbf{A}(\mathbf{x}_{k+1} - \mathbf{x}_k)$.

Let \mathbf{p}_k be the search direction and let α_k be the learning rate that will be chosen to minimize $F(\mathbf{x})$ in direction \mathbf{p}_k , so that $\mathbf{x}_{k+1} - \mathbf{x}_k = \alpha_k \mathbf{p}_k$. Let $\{ \mathbf{p}_k \}$ be a set of vectors. We say that $\{ \mathbf{p}_k \}$ is mutually conjugate with respect to a positive definite Hessian matrix \mathbf{A} if and only if $\mathbf{p}_k^T \mathbf{A} \mathbf{p}_j = 0$ for $k \neq j$. Then $0 = \alpha_k \mathbf{p}_k^T \mathbf{A} \mathbf{p}_j = (\mathbf{x}_{k+1} - \mathbf{x}_k)^T \mathbf{A} \mathbf{p}_j = [\mathbf{A} (\mathbf{x}_{k+1} - \mathbf{x}_k)]^T \mathbf{p}_j = (\nabla F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_{k+1}} - \nabla F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_k})^T \mathbf{p}_j = 0$. If we let \mathbf{x}_0 be the initial guess, then the conjugate gradient method suggests:

$$\mathbf{q}_0 = \nabla F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_0}, \mathbf{p}_0 = -\mathbf{q}_0,$$

$$\alpha_k = \frac{\nabla F(\mathbf{x})^T|_{\mathbf{x}=\mathbf{x}_k} \mathbf{p}_k}{\mathbf{p}_k^T \nabla^2 F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_k} \mathbf{p}_k}, \quad \mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k,$$

$$\beta_k = \frac{\nabla F(\mathbf{x})^T|_{\mathbf{x}=\mathbf{x}_k} \nabla F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_k}}{\nabla F(\mathbf{x})^T|_{\mathbf{x}=\mathbf{x}_{k-1}} \nabla F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_{k-1}}}, \quad \mathbf{p}_k = -\nabla F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_k} + \beta_k \mathbf{p}_{k-1} \quad (2.20)$$

For a general $F(\mathbf{x})$, the Hessian matrix is not required for the conjugate gradient method.

Since computing α requires the Hessian matrix, we use the golden section search method [24] to find α . Consider the function $F(\mathbf{x})$. Let the stepsize be k , which is a small number, says 0.075, and let \mathbf{x}_0 be the initial guess and \mathbf{p}_0 be the direction vector.

Calculate

$$F_0 = F(\mathbf{x}_0), F_1 = F(\mathbf{x}_0 + k\mathbf{p}_0), F_2 = F(\mathbf{x}_0 + 2k\mathbf{p}_0), F_3 = F(\mathbf{x}_0 + 4k\mathbf{p}_0),$$

$$F_4 = F(\mathbf{x}_0 + 8k\mathbf{p}_0), \dots, F(\mathbf{x}_0 + 2^m k\mathbf{p}_0) = F_{1+m}, \dots, \text{ until } F_{j-1} > F_j \text{ and } F_{j+1} > F_j. \text{ Let the constant}$$

$$\text{in the golden section search be } c = 0.618; \text{ fix } \mathbf{a}_1 = \mathbf{x}_0 + 2^{j-1} k\mathbf{p}_0 \text{ and } \mathbf{b}_1 = \mathbf{x}_0 + 2^{j+1} k\mathbf{p}_0;$$

$$\text{calculate } \mathbf{c}_1 = \mathbf{a}_1 + (1-c) * (\mathbf{b}_1 - \mathbf{a}_1), F_c = F(\mathbf{c}_1), \mathbf{d}_1 = \mathbf{a}_1 - (1-c) * (\mathbf{b}_1 - \mathbf{a}_1) \text{ and } F_d = F(\mathbf{d}_1).$$

Set $j=1$;

Repeat

If $F_c < F_d$, then

$$\text{Set } \mathbf{a}_{j+1} = \mathbf{a}_j, \mathbf{b}_{j+1} = \mathbf{d}_j, \mathbf{d}_{j+1} = \mathbf{c}_j, \mathbf{c}_{j+1} = \mathbf{a}_{j+1} + (1-c) (\mathbf{b}_{j+1} - \mathbf{a}_{j+1}), F_d = F_c,$$

$$F_c = F(\mathbf{c}_{j+1})$$

$$\text{else set } \mathbf{a}_{j+1} = \mathbf{c}_j, \mathbf{b}_{j+1} = \mathbf{b}_j, \mathbf{c}_{j+1} = \mathbf{d}_{j+1}, \mathbf{d}_{j+1} = \mathbf{b}_{j+1} - (1-c) (\mathbf{b}_{j+1} - \mathbf{a}_{j+1}), F_c = F_d,$$

$$F_d = F(\mathbf{d}_{j+1})$$

end;

until $(\mathbf{b}_{j+1} - \mathbf{a}_{j+1}) < \alpha$ a fixed small number. Choose $\mathbf{x}_1 = \mathbf{a}_{j+1}$ to be the next point and

calculate α by $\mathbf{x}_1 = \mathbf{x}_0 + \alpha \mathbf{p}_0$. Next, let consider a multi-layer network. Let M be the

number of layers in the network, \mathbf{n} be the vector just before passing through the transfer

function and \mathbf{t} be the target output vector. We want to find the output \mathbf{a} , for each layer.

The initial output \mathbf{a}^0 is equal to \mathbf{p} the input vector; the first layer output

$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1 \mathbf{a}^0 + \mathbf{b}^1). \text{ In general, } \mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1} \mathbf{a}^m + \mathbf{b}^{m+1}) \text{ for } m=0,1,2,\dots,M-1, \text{ and } \mathbf{a}^M$$

is the network output. Then propagate the sensitivities backward through the network.

$$\text{Initial sensitivity } \mathbf{s}^M = -2\mathbf{F}^M(\mathbf{n}^M)(\mathbf{t} - \mathbf{a}^M), \mathbf{s}^m = \mathbf{F}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T \mathbf{s}^{m+1}, \text{ for } m = M-1, \dots,$$

2,1, where

$$F^m(n^m) = \begin{bmatrix} f^m(n_1^m) & 0 & \dots & \dots & 0 \\ 0 & f^m(n_2^m) & 0 & \dots & 0 \\ \vdots & 0 & \ddots & 0 & \vdots \\ \vdots & \vdots & 0 & \ddots & 0 \\ 0 & 0 & \dots & 0 & f^m(n_S^m) \end{bmatrix}, \quad f^m(n_j^m) = \frac{\partial f^m(n_j^m)}{\partial n_j^m}$$

and n^m is the net input at m -th layer. This recurrence relation for the sensitivity could

be written out by using the chain rule in a matrix form. Update W s and b s as

$W^m(k+1) = W^m(k) - \alpha s^m (a^{m-1})^T$ and $b^m(k+1) = b^m(k) - \alpha s^m$, where α is the chosen

learning rate (for example golden section search).

Example

Suppose that we want to approximate the function $g(x) = \cos(\pi x/2)$, $-1 \leq x \leq 1$.

We have to pick some values of x to obtain the training set and choose some small random values to be the initial guess for the weights and biases. Note that this initial guess can be done by a number of intelligent methods. Use the network as below:

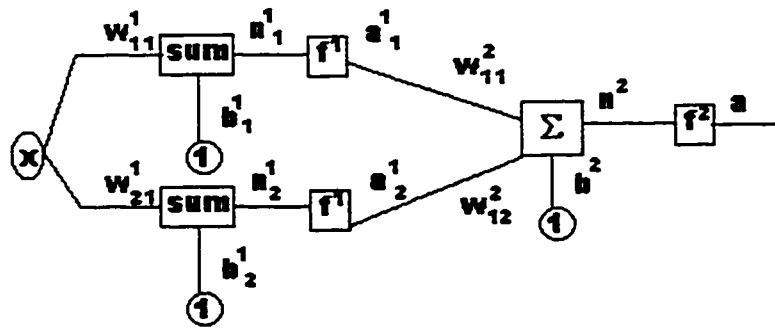


Figure 2.11 Neural network for approximating function $g(x)$.

Here, f^1 is the log-sigmoid transfer function, and f^2 is the linear transfer function.

Initial guesses are $W^1(0) = [-0.12 \ -0.37]^T$, $b^1(0) = [-0.25 \ -0.007]$, $W^2(0) = [0.19 \ -0.23]$

and $b^2(0) = [0.35]$; also for simplicity, set the learning rate at $\alpha = 0.1$. Choose $x=0.5$ as

an example for the training set. Therefore, $\mathbf{a}^0 = \mathbf{x} = 0.5$. Note that the performance index is the mean squared error, $F(\mathbf{x}) = \frac{1}{2}[(\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})]$. Calculate \mathbf{a}^1 and \mathbf{a}^2 as follows:

$$\mathbf{a}^1 = f^1([-0.12 \ -0.37]^T [0.5] + [-0.25 \ -0.07]^T) = f^1([-0.31 \ -0.255]^T) = [0.423 \ 0.437]^T$$

$$\mathbf{a}^2 = f^2([0.19 \ -0.23] [0.423 \ 0.437]^T + [0.35]) = f^2(0.32986) = 0.32986.$$

$$\mathbf{t}-\mathbf{a} = \cos(\pi(0.5)/2) - 0.32986 = 0.707106 - 0.32986 = 0.37725.$$

$$f^1(n) = \frac{d}{dn} \left(\frac{1}{1+e^{-n}} \right) = \left(1 - \frac{1}{1+e^{-n}} \right) \left(\frac{1}{1+e^{-n}} \right) = (1-a^1)a^1, \quad f^2(n) = \frac{d}{dn}(n) = 1$$

$$s^2 = -2F^2(n^2)(t-a) = -2(0.37725) = -0.7545.$$

$$s^1 = F^1(n^1)(W^2)s^2 = \begin{bmatrix} (1-a_1^1)a_1^1 & 0 \\ 0 & (1-a_2^1)a_2^1 \end{bmatrix} \begin{bmatrix} 0.19 \\ -0.23 \end{bmatrix} [-0.7545] = \begin{bmatrix} -0.34443 \\ 0.042695 \end{bmatrix}$$

$$\begin{aligned} \mathbf{W}^2(1) &= \mathbf{W}^2(0) - \alpha s^2 (\mathbf{a}^1)^T = [0.19 \ -0.23] - 0.1[-0.7545][0.423 \ 0.437] \\ &= [0.222 \ -0.197]. \end{aligned}$$

$$\mathbf{b}^2(1) = \mathbf{b}^2(0) - \alpha s^2 = [0.35] - 0.1[-0.7545] = [0.52545]$$

$$\begin{aligned} \mathbf{W}^1(1) &= \mathbf{W}^1(0) - \alpha s^1 (\mathbf{a}^0)^T = [-0.12 \ -0.37]^T - 0.1[-0.34443 \ 0.042695]^T [0.5] \\ &= [-0.10278 \ -0.37214]^T. \end{aligned}$$

$$\begin{aligned} \mathbf{b}^1(1) &= \mathbf{b}^1(0) - \alpha s^1 = [-0.25 \ -0.07]^T - 0.1[-0.34443 \ 0.042695]^T \\ &= [-0.215557 \ -0.0742695]. \end{aligned}$$

This is the end of the first iteration; pick another value of x to perform another iteration of the algorithm until the error is acceptable. Many people modify this basic backpropagation to get a faster algorithm to train the network. Some of the methods include backpropagation with momentum, variable learning rate backpropagation and

Leverberg-Marquardt backpropagation [23]. Some of methods pick a better learning rate [21], and others use a faster method to minimize the performance index [22].

Remarks:

1. The Leverberg-Marquardt backpropagation learning rule is given by:

$$\nabla W = (J^T J + \alpha I)^{-1} J^T e$$

where J is the Jacobian matrix of derivatives of each error for each weight, α is a scalar, and e is an error vector defined by $e = (\text{target output} - \text{network output})$. If the scalar α is small, the above expression becomes the Gauss-Newton method. Near an error minimum, the Gauss-Newton method is faster than the gradient descent method. This method performs a faster learning rate than the gradient descent, but it requires more memory (storage of the approximation of Hessian Matrix $J^T J$).

2. The gradient descent is a technique where the weights and biases are moved in the opposite direction to the error gradient.

3. A two-layer feedforward neural network with R inputs, S neurons in the first layer and T outputs requires storing two weight matrices W^1 , W^2 and two bias vectors b^1 , b^2 . W^1 is a $S \times R$ matrix, W^2 is a $T \times S$ matrix, b^1 is a $S \times 1$ vector and b^2 is a $T \times 1$ vector. The total storage requirement for this network is $S(R+1)+T(S+1)$.

CHAPTER 3

Wavelets and the Multi-resolution Method

In this chapter, we will discuss the basic idea of the wavelet transformation, examples of multi-resolution and some properties of wavelets will be presented.

3.1 Wavelets

The wavelet transformation is a tool that divides a given data into different frequency components, and each component is then studied with a resolution method to its scale. In the last ten years, interest in wavelets has grown at an explosive rate, partly because, wavelets are a mathematical tool with a great variety of possible applications [25, 26, 27, 28]. One of the reasons why we switched from traditional transformation to wavelet transformation is that a wavelet transformation provides a time-frequency description. In Figure 3.1, every box corresponds to a value of the wavelet transform in the time-frequency plane. Boxes have a constant area, although the widths and heights change. Each box represents an equal portion of the time-frequency plane, but gives different proportions to time and frequency. When the heights of the boxes are shorter and the widths are longer, a better frequency but a poorer time resolution is expected. On the other hand, when the widths of these boxes decrease and the heights increase, a better time but a poorer frequency resolution will be given. This explains how wavelets match longer time with lower frequencies and shorter time with higher frequencies.

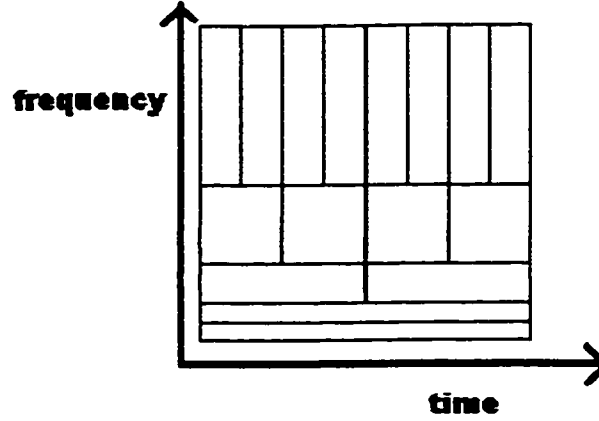


Figure 3.1 Time-frequency plane.

Several useful properties associated with wavelets, such as data compression, signal decomposition and de-noising, will be discussed. Wavelets, which are basis functions in continuous time, satisfy certain properties and cooperate with the scaling function $\phi(t)$. The way to construct the scaling function will not be discussed here, as they are described in detail in reference [11]. A basis is a set of linearly independent functions in which all admissible functions can be written as a combination of basis functions.

Wavelet basis functions $w_{jk}(t)$ are constructed from a mother wavelet $w(t)$, which is a pulse that starts at time $t = 0$ and ends at $t = N$. A typical wavelet $w_{jk}(t)$ is equal to $w(2^j t - k)$. Wavelet basis functions w_{jk} and w_{lm} are orthogonal if

$$\int_{-\infty}^{\infty} w_{jk}(t) w_{lm}(t) dt = 0 \quad (3.1)$$

If, in addition, the inner product of orthogonal wavelet basis functions w_{jk} and w_{jk} equal to 1,

$$\int_{-\infty}^{\infty} w_{jk}(t) w_{jk}(t) dt = 1 \quad (3.2)$$

then it is called orthonormal.

Given a basis $w_{jk}(t)$ and a real function $f(t)$, then $f(t)$ can be expressed in the form

$$f(t) = \sum_{j,k} b_{jk} w_{jk}(t) \quad (3.3)$$

where b_{jk} are constants.

3.2 Multi-resolution Analysis

The main tool in wavelets is called a multi-resolution analysis. Let the scaling functions $\phi(2^j t - k)$ be the basis for a set of signals at level j in which time steps are 2^{-j} .

The new details can then be represented by the wavelet $w(2^j t - k)$. The information y obtained from the scaling functions and the new details z combine into a multi-resolution of a signal. For example, suppose the input vector is $\mathbf{x} = (1,2,3,4,5,6,7,8)$, $|\mathbf{x}| = 8 = 2^3$; therefore it has three levels. Suppose the scaling basis is $(1/2)^{(1/2)}\{(1, 1, 0, \dots, 0), (0, 1, 1, 0, \dots, 0), (0, 0, 1, 1, 0, \dots, 0), \dots, (0, \dots, 0, 1, 1), (0, \dots, 0, 1)\}$ and the wavelet basis is $(1/2)^{(1/2)}\{(-1, 1, 0, \dots, 0), (0, -1, 1, 0, \dots, 0), (0, 0, -1, 1, 0, \dots, 0), \dots, (0, \dots, 0, -1, 1), (0, \dots, 0, -1)\}$. At level $J = 3$, input vector $\mathbf{x}^3 = (1,2,3,4,5,6,7,8)$

$$y^3 = \frac{1}{\sqrt{2}} \begin{bmatrix} 3 \\ 5 \\ 7 \\ 9 \\ 11 \\ 13 \\ 15 \\ 8 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{bmatrix}$$

$$z^3 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ -8 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{bmatrix}$$

For $k = 0$ to $|x^3|/2 - 1$, by picking $y^3(1+2k)$ and $z^3(1+2k)$ as the first part of the output and the input of the next level, the first part of the output is given by

$(1/2)^{(1/2)}(3, 7, 11, 15)$ and the next input is given by $(1/2)^{(1/2)}(1, 1, 1, 1)$.

At level $J = 2$, input vector $x^2 = (1/2)^{(1/2)}(1, 1, 1, 1)$.

$$y^2 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1/2 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$z^2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & -1 \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

For $k = 0$ to $|x^2|/2 - 1$, by picking $y^2(1+2k)$ and $z^2(1+2k)$ as the second part of the output and the input of the next level, the second part of the output is given by $(1, 1)$ and the next input is given by $(0, 0)$. At level $J = 1$, the input vector $x^1 = (0, 0)$ and the results y^1 and z^1 are given by $(0, 0)$. So, the third part of the output is (0) and the last part of the output is (0) . Hence,

$$output = \left[\frac{3}{\sqrt{2}} \quad \frac{7}{\sqrt{2}} \quad \frac{11}{\sqrt{2}} \quad \frac{15}{\sqrt{2}} \quad 1 \quad 1 \quad 0 \quad 0 \right]$$

Note

This is a simple example with complete calculation for the multi-resolution. One more example, without details, is shown below: consider input $x = 128$ points of $\exp(-0.3t)\sin(10\pi t)$, when $t=0$ to 1.

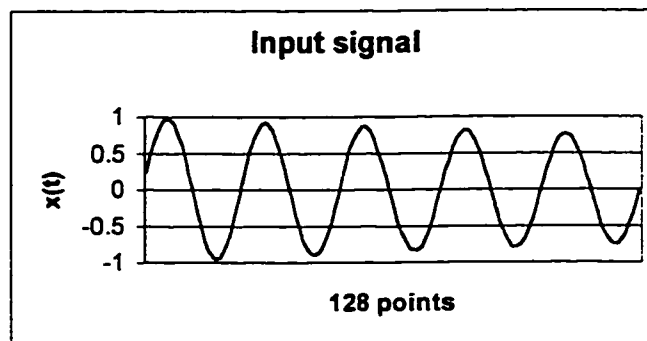


Figure 3.2 Example of an input signal using 128 points.

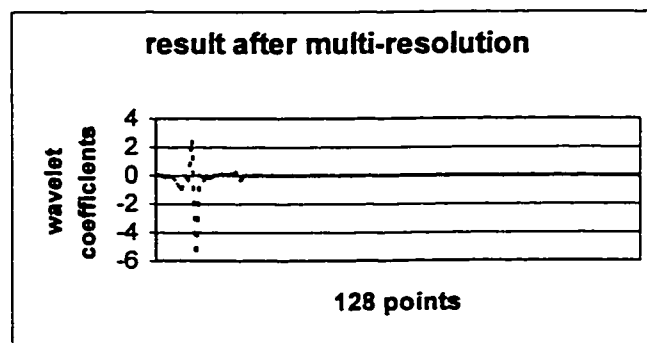


Figure 3.3 Output of multi-resolution.

Note that most of the wavelet coefficients are close to zero.

Remarks

1. Since most of the coefficients of the output are close to zero, it is sufficient to use k number of non-zero coefficients to reconstruct the approximation of the signal, where

$k \ll 128$. Using different numbers of coefficients in figure 3.3, the maximum absolute error of the reconstruction of the signal is shown below:

# of coefficients k	10	15	20	25
Maximum error	0.0713	0.0381	0.0333	0.0157

# of coefficients k	30	35	40	45
Maximum error	0.0081	0.0013	0.00078315	0.00035618

Table3.1 Maximum error of the reconstruction with different numbers of coefficients.

The results show that by using more coefficients, a better approximation can be constructed. Consider another example in data compression. Let the input vector $\mathbf{x}(t)$ with a length of 512 points which results from $\exp(-0.5t)\sin(8\pi t) + \exp(-0.7t)\sin(12\pi t)$, and $t = 0$ to 2, as shown in figure 3.4. In figure 3.5, the reconstruction signal using 45 coefficients is displayed, and it clearly indicated a good approximation of the signal.

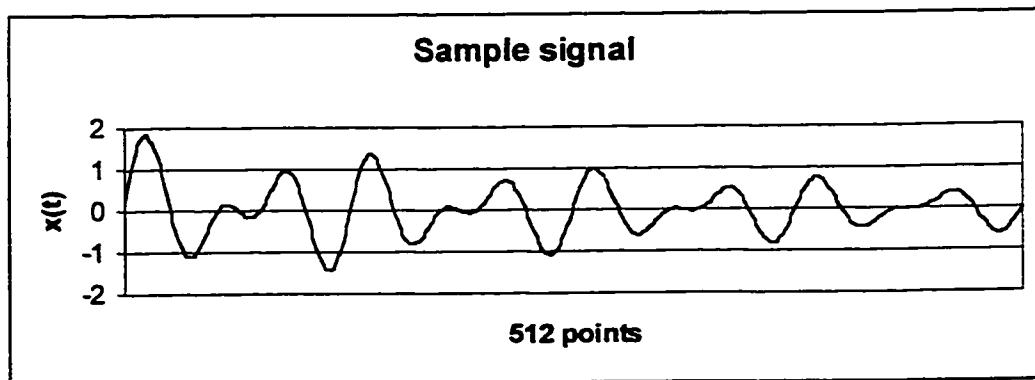


Figure 3.4 Example of a signal using 512 points.

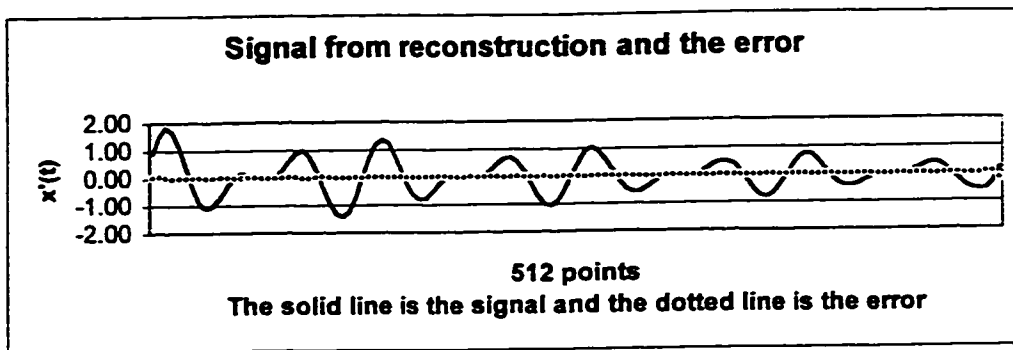


Figure 3.5 Reconstruction using 45 wavelet coefficients.

The performance depends on the number of points, the frequency and the number of coefficients that we are using. Figure 3.6 shows the effect when the number of discrete points of the given signal in figure 3.4 decreases (i.e. the input vector $\mathbf{x}(t)$ has a length of 128 and the function is the same as figure 3.4.). Using 12 wavelet coefficients, the result is

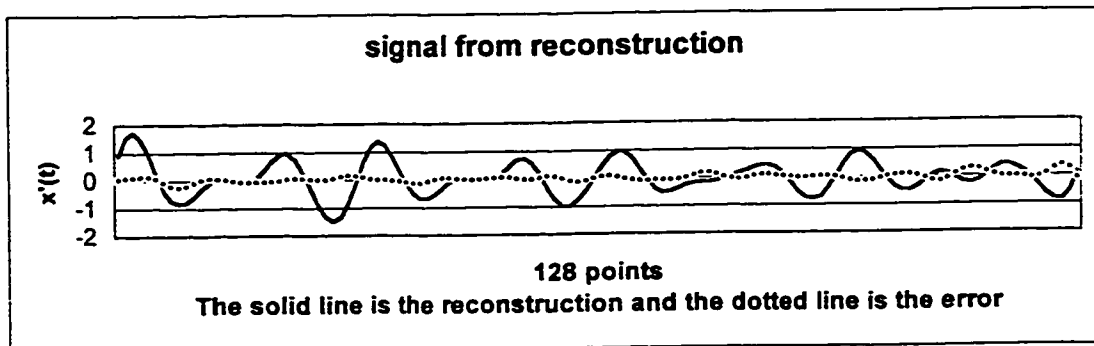


Figure 3.6 Reconstruction using 12 wavelet coefficients.

Using 25 coefficients, the result is:

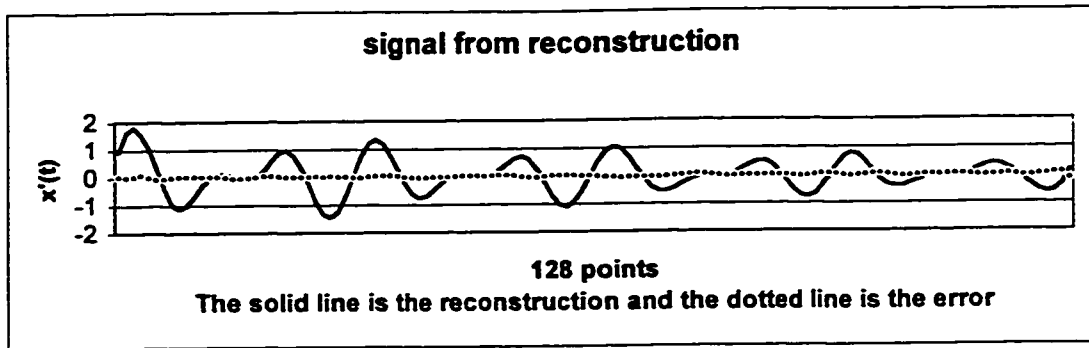


Figure 3.7 Reconstruction using 25 wavelet coefficients.

2. In de-noising, when we have a noisy signal, the selection of the wavelet coefficients also plays an important role. For a noisy signal $x_1(t) = 512$ points of $(1 + 0.2\text{noise})x(t)$, where the noise is a random number from 0 to 1 and $x(t)$ is the function in figure 3.4, the reconstruction is shown below, using different numbers of coefficients.

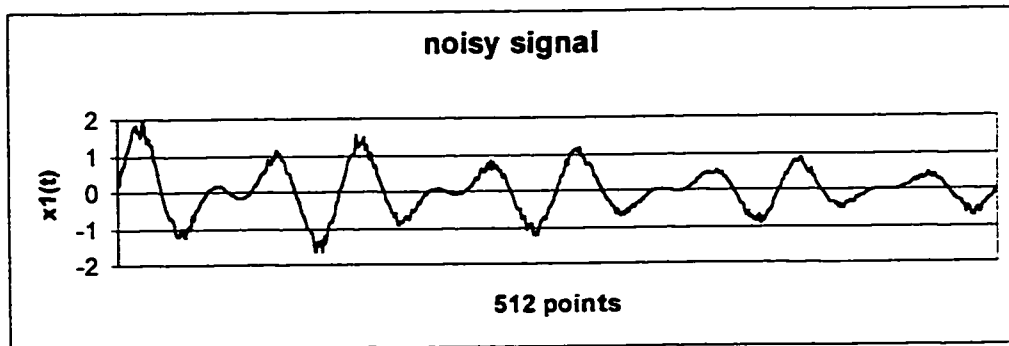


Figure 3.8 Noisy signal of the signal in figure 3.4.

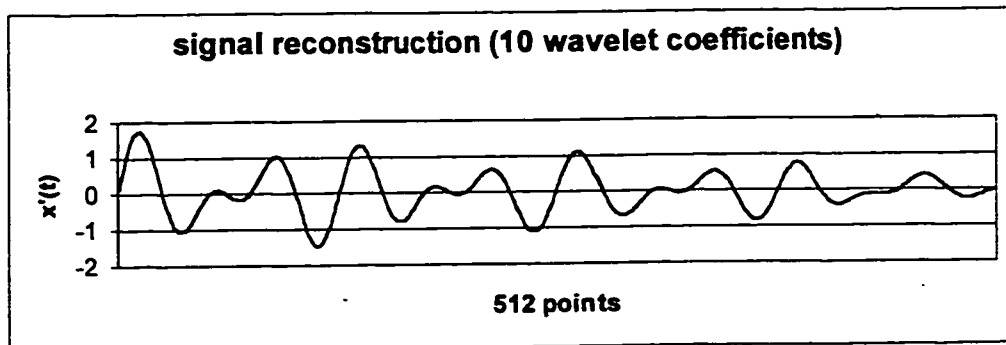


Figure 3.9 Signal reconstructed by 10 wavelet coefficients.

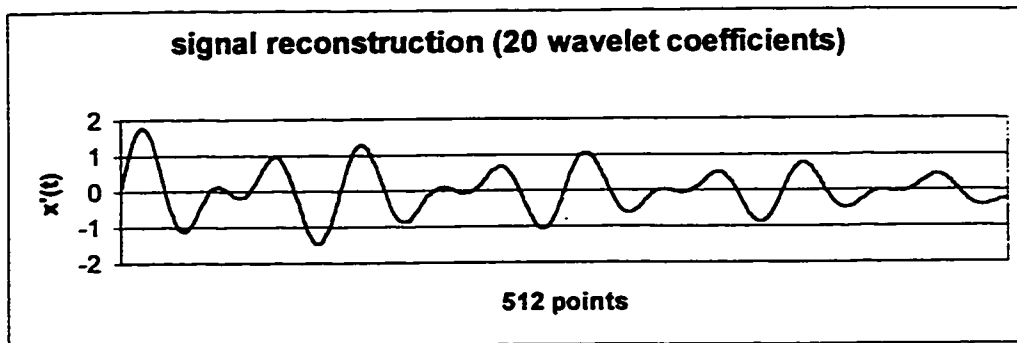


Figure 3.10 Signal reconstructed by 20 wavelet coefficients.

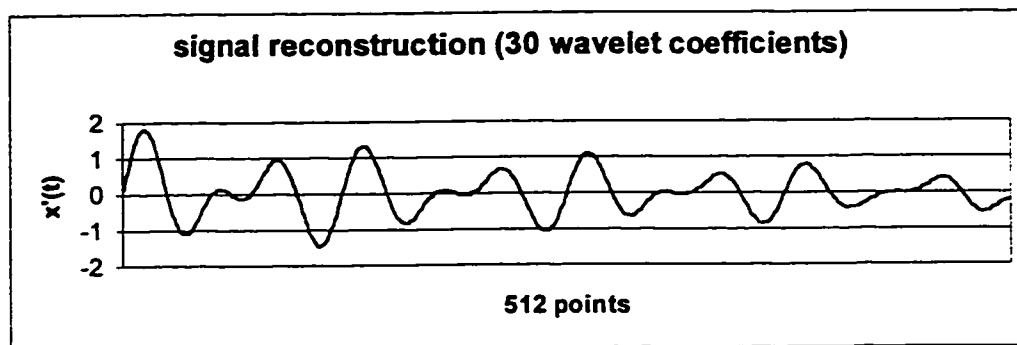


Figure 3.11 Signal reconstructed by 30 wavelet coefficients.

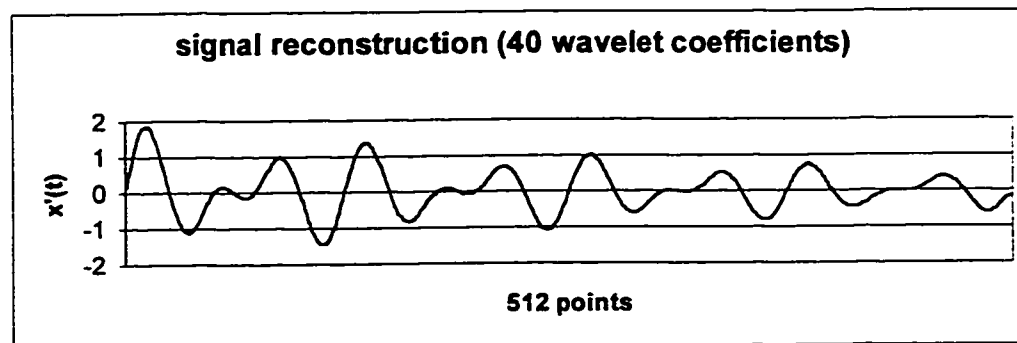


Figure 3.12 Signal reconstructed by 40 wavelet coefficients.

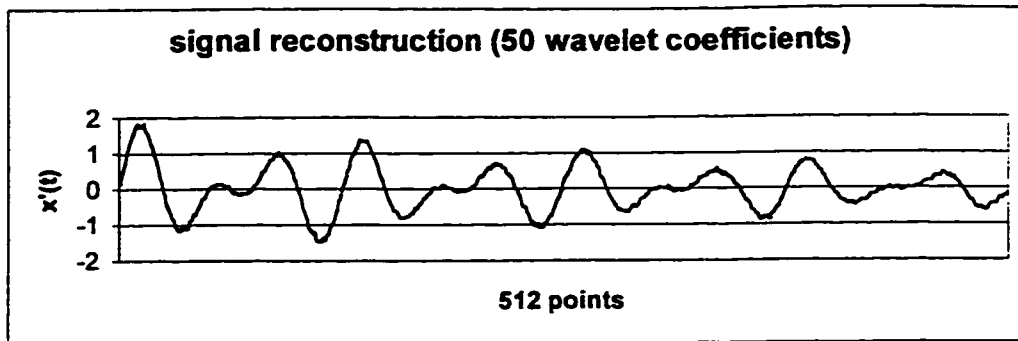


Figure 3.13 Signal reconstructed by 50 wavelet coefficients.

Using 10 to 40 wavelet coefficients, some of the signal's power is removed from the noisy signal. Using 50 coefficients, the noise effect re-occurs. It is generally impossible to filter out all the noise without affecting the signal. To improve the result, people may use a different basis and a different number of coefficients to de-noise the signal [29].

3. Given a complex signal $x(t) = \exp(-at)\sin(2b\pi t) + \exp(-ct)\sin(2d\pi t)$, use wavelet coefficients can be used to separate $x(t)$ and get approximations of $\exp(-at)\sin(2b\pi t)$ and $\exp(-ct)\sin(2d\pi t)$. The next example uses $x(t) = \exp(-0.4t)\sin(6\pi t) + \exp(-0.7t)\sin(16\pi t)$.

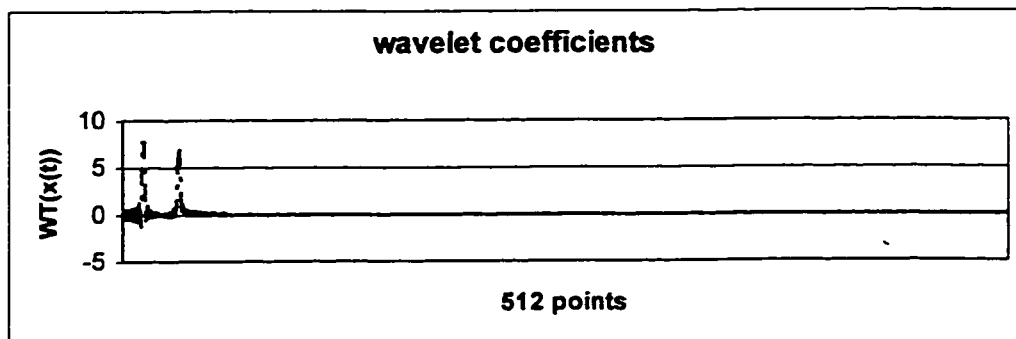


Figure 3.14 Multi-resolution result of $x(t)$.

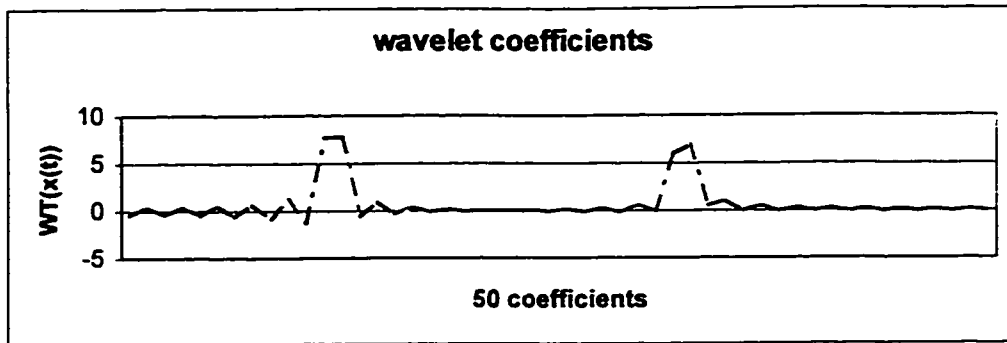


Figure 3.15 First 50 coefficients in figure 3.12.

Using the first 25 coefficients to reconstruct the signal, we will have:

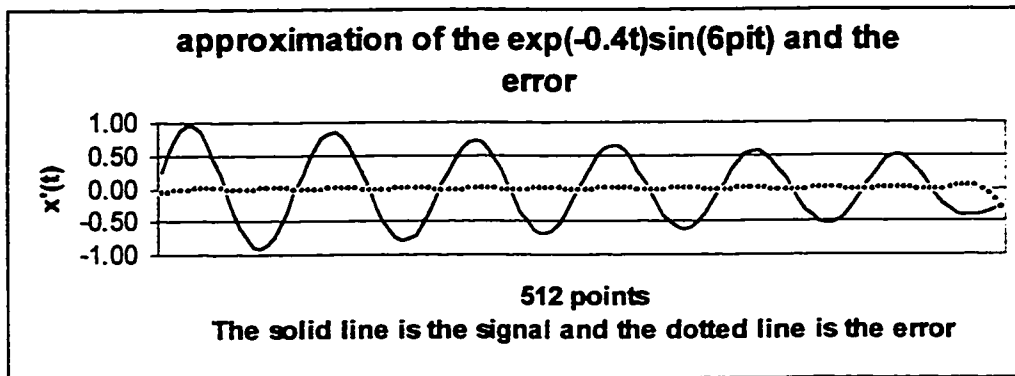


Figure 3.16 Reconstruction using the first 25 coefficients and the error.

Using the next 25 coefficients to reconstruct the signal, we will have:

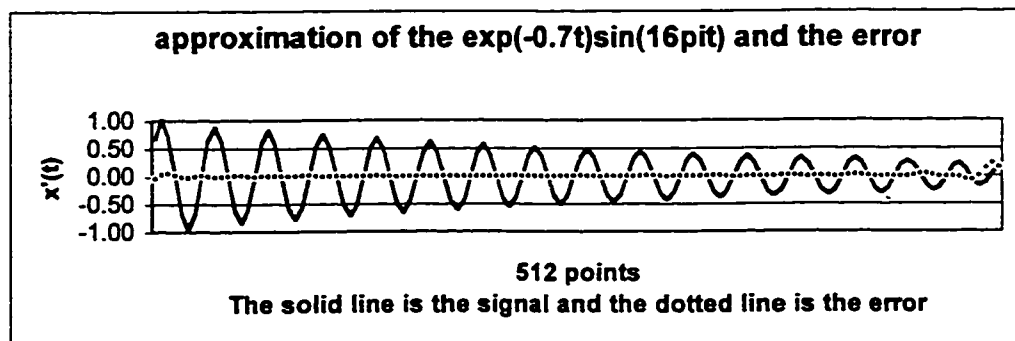


Figure 3.17 Reconstruction using the next 25 coefficients and the error.

This method may also apply to more complex signals, such as $x(t) = \sum x_i(t)$ for signals $x_1(t), x_2(t), \dots, x_n(t)$, where $x_i(t) = \exp(-a_i t) \sin(b_i t)$.

Remarks

When the difference between the two frequencies of the signal is small, it may cause difficulty in applying a wavelet analysis. Consider an example,

$$x(t) = \exp(-0.4t)\sin(20\pi t) + \exp(-0.7t)\sin(20.1\pi t), \quad 2 \geq t \geq 0.$$

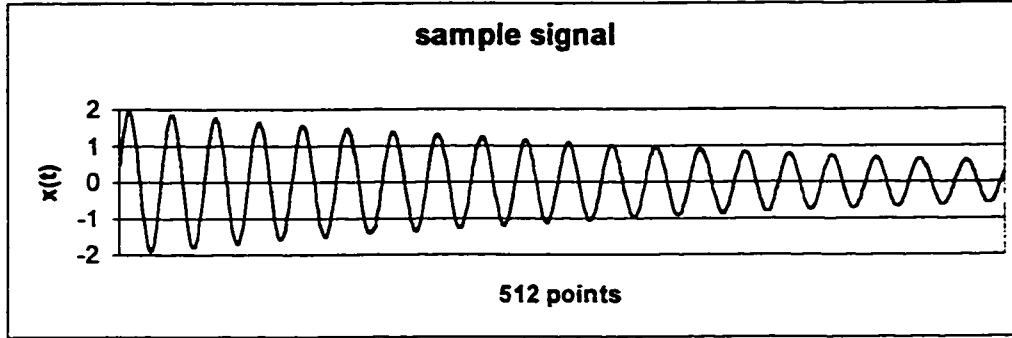


Figure 3.18 Example signal using 512 points.

Then the wavelet coefficients are:

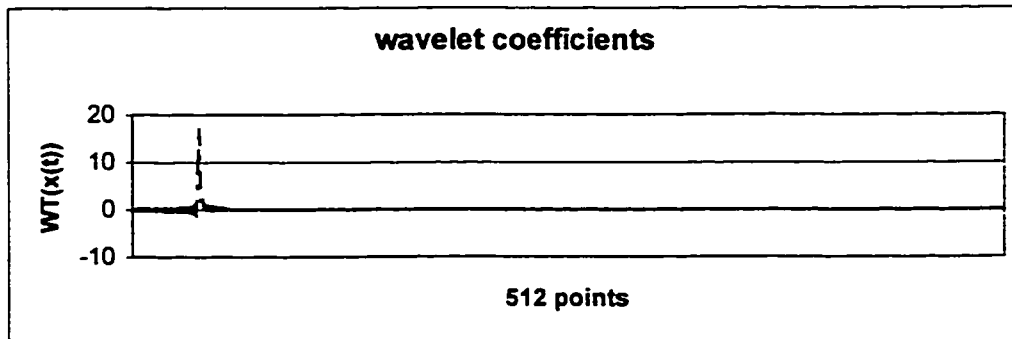


Figure 3.19 Wavelet coefficients of the signal in 3.18.

In figure 3.19, the two 'pulses' are joined together. We cannot easily choose the coefficients to reconstruct the two signals. So, we need to know the minimum difference between the two frequencies for the signal. Consider the signal $x_1(t) = \exp(-0.4t)\sin(2(10)\pi t) + \exp(-0.7t)\sin((2(10+\Delta f)\pi t))$, $0 \leq t \leq 2$; we need $\Delta f = 0.4$ to separate these signals. If $\Delta f < 0.4$, then in light of figure 3.19, the signal $x_1(t)$ is very

hard to separate into two components. Figure 3.20 shows the first 70 coefficients of the signal $x_1(t)$ when $\Delta f = 0.4$.

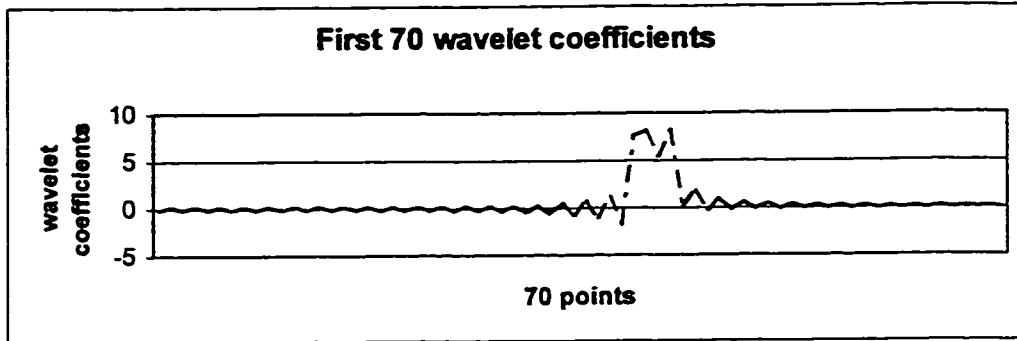


Figure 3.20 First 70 coefficients of the signal $x_1(t)$ when $\Delta f = 0.4$.

Consider the signal $x_2(t) = \exp(-0.4t)\sin(2(100)\pi t) + \exp(-0.7t)\sin(2(100+\Delta f)\pi t)$, $0 \leq t \leq 0.5$, we need $\Delta f = 2.5$ to separate these signals. So, in general, given $x(t) = \exp(-at)\sin(2b\pi t) + \exp(-ct)\sin(2(b+\Delta f)\pi t)$; the minimum value of Δf depends on the variable t . Note that figure 3.21 is similar to figure 3.20. We use different sampling rate (i.e. for example 1000 samples per second.) for different signals (i.e. $x_1(t)$ and $x_2(t)$). The result of Δf is shown in table 3.2 for $t = 0$ to 2 and in table 3.3 for $t = 0$ to 0.5.

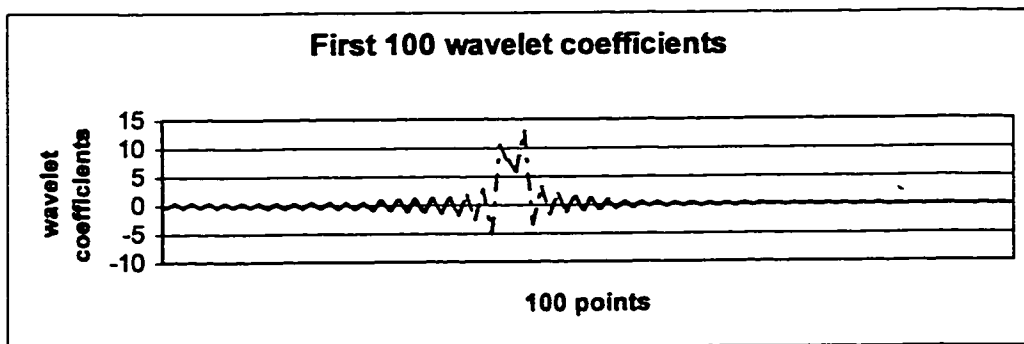


Figure 3.21 First 100 coefficients of the signal $x_2(t)$ when $\Delta f = 5$.

Time t is from 0 to 2.

Base frequency	$x_2(t)$	$x_1(t)$
Sampling rate (N/sec)	Δf	Δf
500	0.4	0.4
1000	0.4	0.4
2000	0.4	0.4

Table 3.2 Minimum Δf for $t=0$ to 2 with different sampling rates.

Time t is from 0 to 0.5.

Base frequency	$x_2(t)$	$x_1(t)$
Sampling rate (N/sec)	Δf	Δf
500	2.5	2.5
1000	2.5	2.5
2000	2.5	2.5

Table 3.3 Minimum Δf for $t=0$ to 0.5 with different sampling rates.

CHAPTER 4

Software Programs

In this chapter, we will briefly describe the two neural networks, namely Sinon and the MATLAB Neural Network toolbox. Sinon is an in-house developed software based on a two-layer feedforward neural network, and it is implemented in C++. The MATLAB Neural Network toolbox is a family of application-specific solutions based on the MATLAB environment. Several types of neural networks can be developed from this toolbox, for example a feedforward multi-layer neural network and a recurrent neural network. Moreover, many training methods are available for this toolbox. In section 5.4 we will compare the performance of these two programs.

4.1 Sinon

Sinon is an in-house software development using C++. This software is a simple tool for creating, training and executing a two-layer neural network. To use this tool, several files are required. First, the training file, which contains the training sets and the corresponding targets, is created. Second, the testing file, containing the data sets as an input to the neural network, is generated. Third, there is the executable file, which is called `sinon.exe`. Last, `sinon.cfg` contains all the necessary information including the number of neurons in the hidden layer and some names of data files. After setting up these files, type “`sinon start`” to initialize the parameters (weights) in the neural network. Type “`sinon continue`” to train the neural network using backpropagation with the conjugate gradients method to optimize the performance index. After training, type “`sinon run`” to get the output file. In this section, I will list all the files involved in producing `sinon.exe` and the relation of these files.

These files include: actfun.c, actfun.h, anneal.c, conigrad.c, direcmin.c, execute.c, layer.c, layer.h, mem.c, mem.h, misc.c, misc.h, network.c, network.h, save.c, sinon.c, svd.c, svd.h, tset.c, tset.h, sinon.cfg, makefile, const.h.

The file sinon.c calls most of the files above. In figure 4.1, we illustrate when misc.c is called by sinon.c.



Figure 4.1 Notation of file calling.

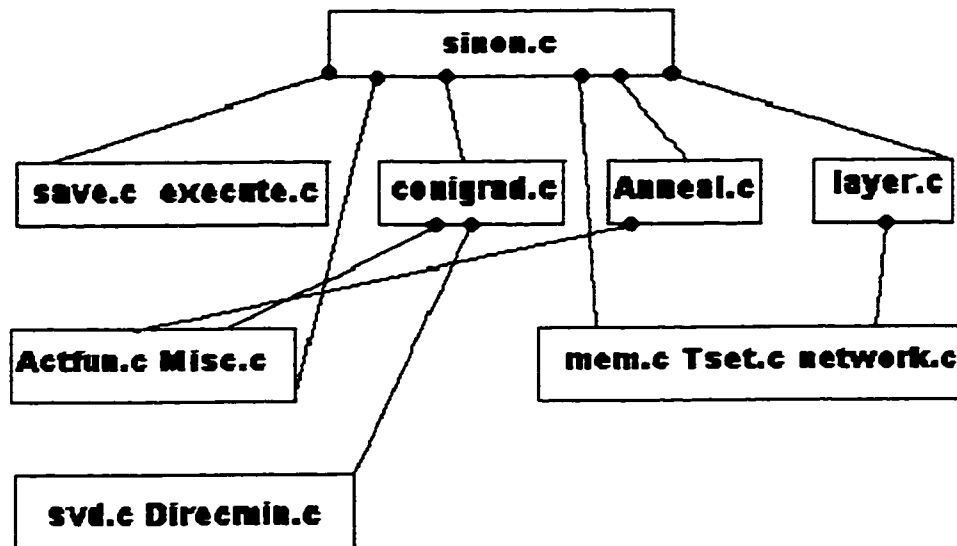


Figure 4.2 System of sinon.exe.

A feedforward two-layer network is used in Sinon. Let R be the number of inputs, n be the number of outputs, k be the number of neurons in the hidden layer, \mathbf{p} be the $R \times 1$ input vector, \mathbf{a} be the $n \times 1$ output vector, \mathbf{W}_1 be the weighted matrix of the first layer and \mathbf{W}_2 be the weighted matrix of the second layer. The transfer function in the first layer is the logsig function and the transfer function in the second layer is the purelin function. So, network output is $\mathbf{a} = \mathbf{W}_2 * \text{logsig}(\mathbf{W}_1 * \mathbf{p})$. The training method in Sinon is

backpropagation with conjugate gradient (discussed in chapter 2), and the performance index is the sum-squared error, which is the sum of squared differences between network outputs and target outputs. Let m be the number of training cases, let $netout_{ij}$ be the network output at the i -th training case and j -th output and let $target_{ij}$ be the target output at the i -th training case and j -th output. Note that $m \geq i \geq 1$, $n \geq j \geq 1$

$$Performance\ index = \sum_{i=1}^m \sum_{j=1}^n (netout_{ij} - target_{ij})^2$$

File `sinon.cfg`:

This sets the number of neurons in the hidden layer, the number of inputs, the names of the files that store the training set, testing set, weighted matrices and the output and also sets the stop condition. The stop condition uses the target error, which is defined by:

$$\sqrt{\frac{Performance\ index}{mn}}$$

The target error is the square root of the performance index divided by product of the number of training cases and the number of outputs.

File `actfun.c` :

This computes the transfer function (activation function) and the corresponding functions.

File `anneal.c`:

Uses simulated annealing to optimize weights in the network.

Note that simulated annealing can be performed to optimization by randomly perturbing the independent variables (weights in the case of a neural network) and keeping track of the best (lowest error) function value for each randomized set of variables. It is very

useful to initialize the weights and thus help the conjugate gradient method to find the global minimum.

File conigrad.c:

Conjugate gradient optimization.

File direcmin.c:

Minimize along a direction; for the normal process, return a real value from 0 to 1; for any error that occurs, it will return a negative real number.

File execute.c:

Read and write the input and output files (for example read training file and write the output file).

File layernet.c: Include all principal routines for neural network processing.

File mem.c: Supervised memory allocation

File misc.c: Include some message functions and random number generator.

File network.c: Network routines specific to the Network parent class.

File save.c: Save and restore learned weights to/from disk files.

File sinon.c: Main program for Simplified Neural Network.

File svd.c:

SingularValueDecomp - Singular value decomposition of matrices: object routines for performing singular value decomposition on a matrix and using back substitution to find least squares solutions to simultaneous equations.

File tset.c: Including all routines related to training.

4.2 MATLAB toolbox.

The name MATLAB stands for matrix laboratory. It is a technical computing environment for high-performance numeric computation and visualization. MATLAB integrates numerical analysis, matrix computation, signal processing, and graphics in an easy-to-use environment. It also features a family of application-specific solutions that are called toolboxes. Areas in which toolboxes are available include signal processing, control systems design and the neural network. This thesis uses the neural network toolbox. Many different network architectures are available. In this thesis, I focus only on the two-layer network. Let R be the number of input, n be the number of outputs, k be the number of neurons in the hidden layer, \mathbf{p} be the $R \times 1$ input vector, \mathbf{a} be the $n \times 1$ output vector, \mathbf{W}_1 be the weighted matrix of the first layer, \mathbf{W}_2 be the weighted matrix of the second layer, \mathbf{b}_1 be the bias vector of the first layer and \mathbf{b}_2 be the bias vector of the second layer. The transfer function in the first layer is the logsig function, and the transfer function in the second layer is the purelin function. So, network output is $\mathbf{a} = \mathbf{W}_2 * \text{logsig}(\mathbf{W}_1 * \mathbf{p} + \mathbf{b}_1) + \mathbf{b}_2$, which is called a feedforward two-layer network. Many training methods are available. In this thesis, training using backpropagation with Momentum and Adaptive Learning Rate (see chapter 5) will be used. The performance index is the sum-squared error. For different networks and different training methods, the neural network toolbox provides many functions to do the job. Functions such as `initff`, `trainbpx` and `simuff` are used in this thesis. To call up these functions in MATLAB: Let \mathbf{W}_1 , \mathbf{W}_2 , \mathbf{b}_1 , \mathbf{b}_2 be the weight matrices and bias vectors, \mathbf{P} be a matrix of input vectors $[\mathbf{p}_1, \mathbf{p}_2, \dots]$, let the number of neurons in the hidden layer be `num`, f_1 , f_2 be the transfer functions and \mathbf{T} be the corresponding target output. Call `[\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1,`

$\mathbf{b}_2]$ = initff(P , num , f_1 , T , f_2) will produce the weight matrices and bias vectors of the neural network with output equals to $f_2(\mathbf{W}_2 f_1(\mathbf{W}_1 \mathbf{p}_1 + \mathbf{b}_1) + \mathbf{b}_2)$. Let display_frequency=100, max_loop=1000, error_goal= 0.01, starting_learning_rate = 0.01, tv = [set display_frequency max_loop error_goal starting_learning_rate]. Then calling [\mathbf{W}_1 , \mathbf{W}_2 , \mathbf{b}_1 , \mathbf{b}_2 , number_loop, training_error] = trainbpx (\mathbf{W}_1 , \mathbf{b}_1 , f_1 , \mathbf{W}_2 , \mathbf{b}_2 , f_2 , P, T, tv) produces the weight matrices, bias vectors of the neural network, the number of loops trained and a record of training errors. Let \mathbf{p}_k be the input vector. Then calling out = simuff (\mathbf{p}_k , \mathbf{W}_1 , \mathbf{b}_1 , f_1 , \mathbf{W}_2 , \mathbf{b}_2 , f_2) produces the output of the neural network

CHAPTER 5

Computer Simulations

To demonstrate the performance of the developed two-layer network, the Sinon program and the MATLAB neural network toolbox, the neural network programs are applied to classification and parameter extraction problems. A comparison of these two network programs will also be presented. All simulation results reported here were obtained from running the neural networks on a PC (Pentium 133MHz with 48 MB RAM).

5.1 Classification problem

In the modern flight flutter test, accurate determination of the frequency and damping coefficients is an important task [30]. Before extracting the frequency and damping coefficients, we first want to use the neural network to classify the correct type of signal, namely a signal with exponential decaying sine waves.

Problem I :

In this problem, we want to classify two sets of functions.

Let $C_0 = \{y(t) : y(t) = Ae^{-at}\sin(bt+c), \text{ where } t_{\min} \leq t < t_{\max}, a_{\min} < a < a_{\max}, b_{\min} < b < b_{\max}, 0 < c < 2\pi\}$ be class 0, $C_1 = \{y(t) : y(t) = Ae^{-at}\sin(bt+c), \text{ where } t_{\min} \leq t < t_{\max}, a_{\min} < a < a_{\max}, b_{\min} < b < b_{\max}, 0 < c < 2\pi\}$ be class 1.

The neural network is applied to solve this problem, and when the network output (netout) is between 0.9 and 1, (i.e. $0.9 < \text{netout} < 1$), we consider the input function is in class 1 (C_1); similarly, when $0 < \text{netout} < 0.1$, the input function is considered to be in class 0 (C_0). Let the training set be generated from 400 random samples from C_0 , where $0 \leq t < 4$, $0.2 < a < 1$, $2 < b < 10$, $A=1$ and from 400 random samples from C_1 , where $0 \leq t < 4$,

$0.2 < a < 1$, $2 < b < 10$, $A=1$, and let the testing set be generated from 500 random samples from C_0 and 500 random samples from C_1 .

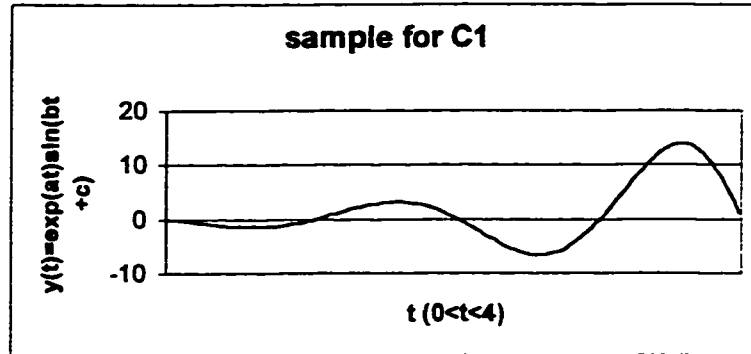


Figure 5.1.1 Example of the function in C_1 for problem I.

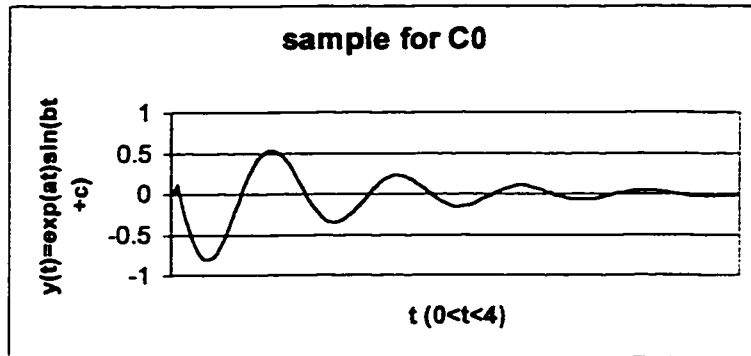


Figure 5.1.2 Example of the function in C_0 for problem I.

Problem II:

As a modification of problem I, problem II adds a noise level ranging from 10% to 50% to the clean signal in problem I.

Let $C_0 = \{y(t) : y(t) = (1+0.1\varepsilon) e^{-at}\sin(bt+c), \text{ where } 0 \leq t < 4, 0.2 < a < 1, 2 < b < 10, 0 < c < 2\pi\}$ be class 0, and $C_1 = \{y(t) : y(t) = (1+0.1\varepsilon) e^{at}\sin(bt+c), \text{ where } 0 \leq t < 4, 0.2 < a < 1, 2 < b < 10, 0 < c < 2\pi\}$ be class 1. Here ε is generated from a random number with coefficients in

$[0,1]$, and 0.1ϵ represents a 10% noise level. The setting of the training set of the neural network is the same as problem I.

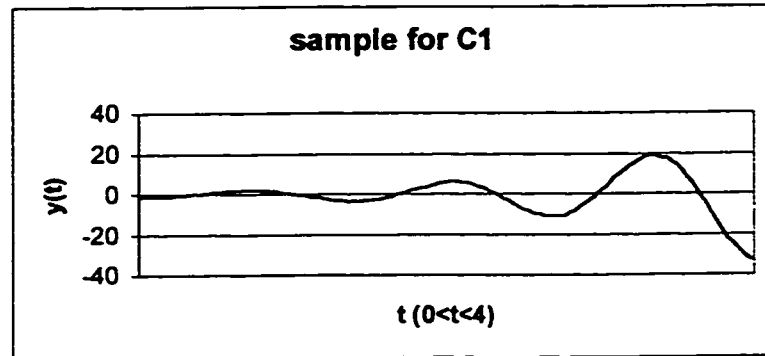


Figure 5.1.3 Example of the function in C_1 with 10% noise level.

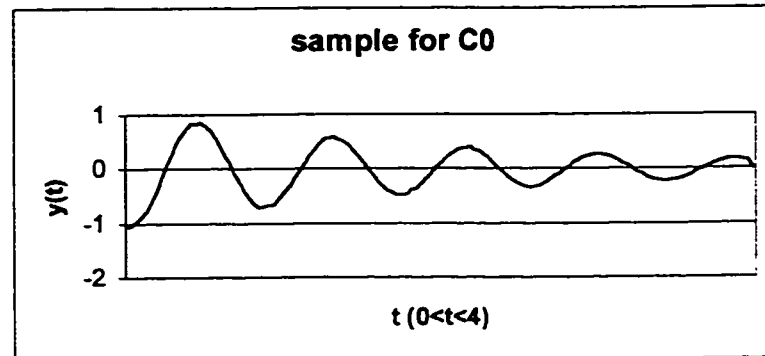


Figure 5.1.4 Example of the function in C_0 with 10% noise level.

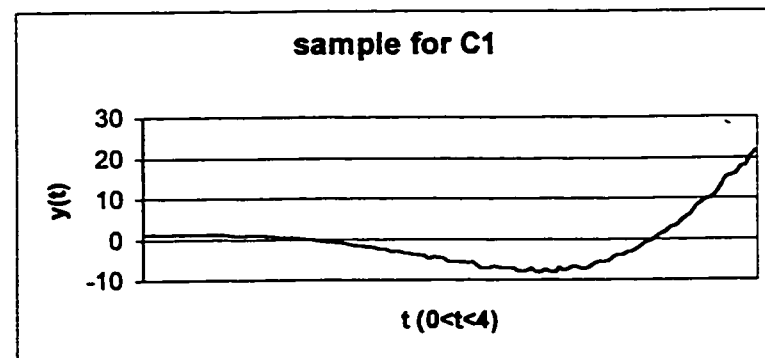


Figure 5.1.5 Example of the function in C_1 with 20% noise level.

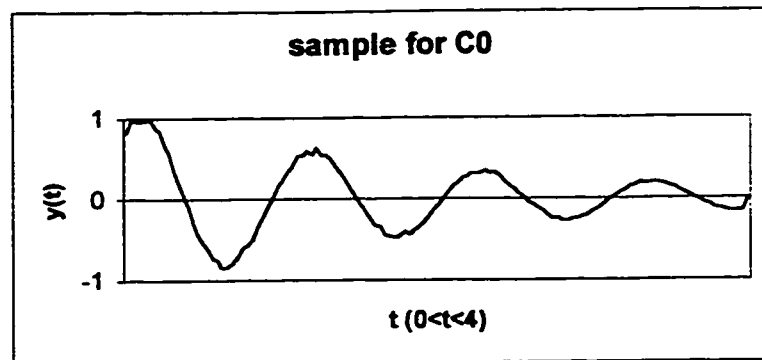


Figure 5.1.6 Example of the function in C_0 with 20% noise level.

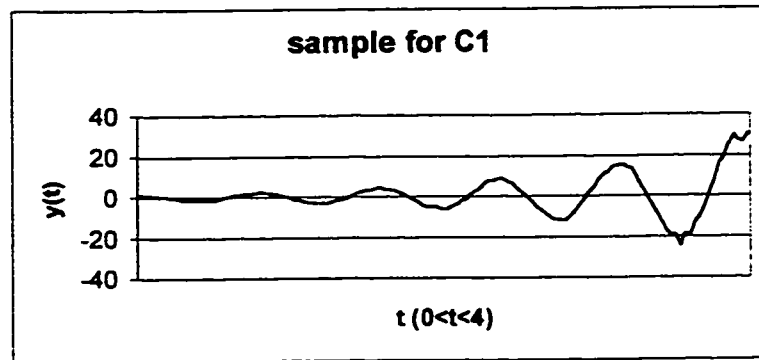


Figure 5.1.7 Example of the function in C_1 with 30% noise level.

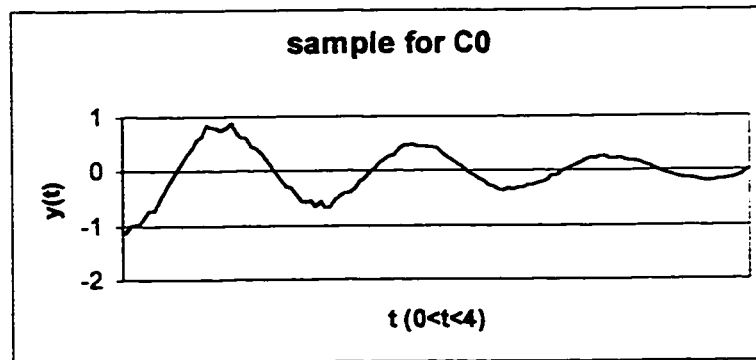


Figure 5.1.8 Example of the function in C_0 with 30% noise level.

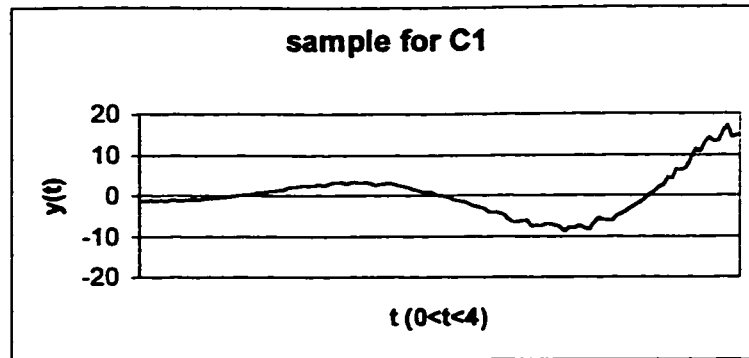


Figure 5.1.9 Example of the function in C_1 with 40% noise level.

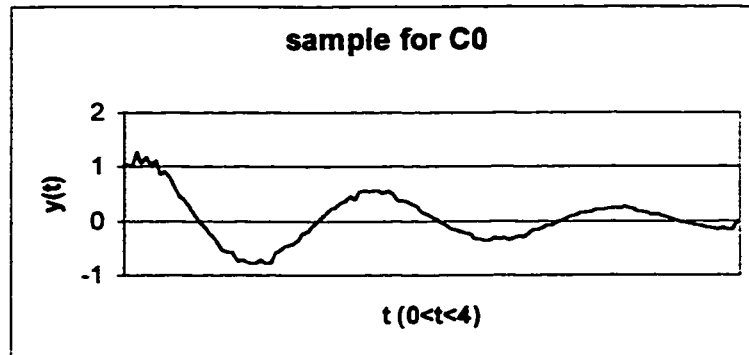


Figure 5.1.10 Example of the function in C_0 with 40% noise level.

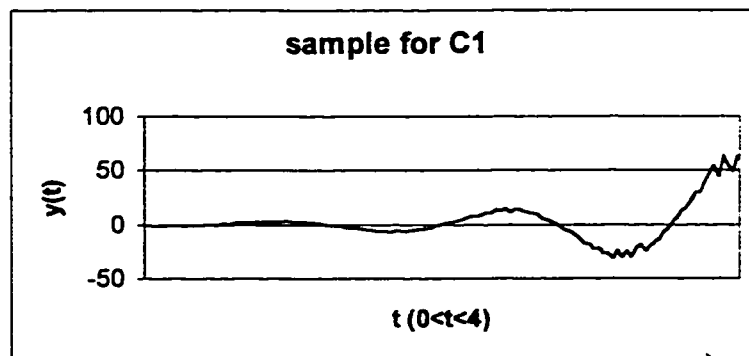


Figure 5.1.11 Example of the function in C_1 with 50% noise level.

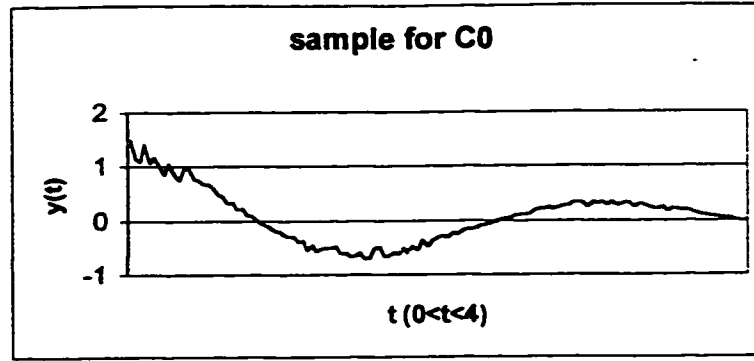


Figure 5.1.12 Example of the function in C_0 with 50% noise level.

Overall result for the classification problem:

Let $C_0 = \{y(t) : y(t) = (1 + \text{noise } \epsilon) e^{-at} \sin(bt+c), \text{ where } 0 \leq t < 4, 0.2 < a < 1, 2 < b < 10, 0 < c < 2\pi\}$ be class 0, and let $C_1 = \{y(t) : y(t) = (1 + \text{noise } \epsilon) e^{at} \sin(bt+c), \text{ where } 0 \leq t < 4, 0.2 < a < 1, 2 < b < 10, 0 < c < 2\pi\}$ be class 1, $\text{noise} \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5\}$. Table 5.1.1 presents the results of using the neural network to classify the signals with noise levels ranging from 0% (clean signal) to 50%. This table shows the percentage for 1000 testing samples.

noise	Correct results	Reject cases	Incorrect results
0.0	99.4%	0.6%	0%
0.1	99.1%	0.9%	0%
0.2	98.7%	1.2%	0.1%
0.3	98.5%	1.2%	0.3%
0.4	99.4%	0.3%	0.3%
0.5	98.9%	1.1%	0%

Table 5.1.1 Result of the classification problem.

Next we change the conditions for determining the correct classification, such that if $(0.8 < \text{netout} < 1)$ then the input is in C_1 , and if $(0 < \text{netout} < 0.2)$ then the input is in C_0 .

The percentage for 1000 testing samples is displayed in table 5.1.2.

Noise	Correct results	Reject cases	Incorrect results
0.0	99.6%	0.4%	0%
0.1	99.3%	0.7%	0%
0.2	98.9%	1.0%	0.1%
0.3	99.2%	0.5%	0.3%
0.4	99.6%	0.1%	0.3%
0.5	99.6%	0.4%	0%

Table 5.1.2 Result of the classification problem.

Conclusion: From the above results, it is clear that the neural network performs well, with more than 95% correct results and with less than 1% incorrect results.

Remarks

1. The neural networks to solve these problems are based on two-layer feedforward neural networks with 12 hidden neurons. Define the result as $(\text{netout}) = f_2(W_2(f_1(W_1(\text{input})+b_1))+b_2)$, where input is a 128×1 vector, W_1 is a 12×128 matrix, b_1 is a 12×1 vector, W_2 is a 1×12 matrix, b_2 is a 1×1 vector and $f_1(n)=f_2(n)=\text{logsig}(n)=1/(1+\exp(-n))$.
2. The training method to solve this problem is based on the backpropagation (steepest descent) momentum and adaptive learning rate. The initial values are taken from a random number generator, update weight matrices and bias vectors by $\Delta W_m(k) = \gamma \Delta W_m(k-1) - (1-\gamma)\alpha s_m(a_{m-1})^T$, $\Delta b_m(k) = \gamma \Delta b_m(k-1) - (1-\gamma)\alpha s_m$, here, k represents the k -th iteration, m is the identifier to identify the layer, α is the learning rate, s_m is the sensitivity for the m -th layer, which is calculated using the same method as in backpropagation, a_{m-1} is the output for the $(m-1)$ th layer, which is calculated from

the network, and γ is the momentum coefficient. Note that $0 \leq \gamma < 1$. In this case, set $\gamma=0.95$. The learning rate α is changed according to the following rules. If the new error exceeds the old error by more than a certain ratio (set at 1.04) then the result is not used and α is multiplied by 0.7. If the new error is less than the old error, then keep the result and α is multiplied by 1.05.

3. First apply a wavelet transform to the input data. After sorting the coefficients by descending order, we use the wavelet multi-resolution tool to pick the first three largest coefficients and their corresponding level information as inputs to the neural network. The number of parameters in the neural network is reduced. The number of inputs is 6 and the number of hidden neurons is 8. So, the sizes of weight matrices and bias vectors are 8×6 , 1×8 , 8×1 and 1×1 . The percentage for 1000 testing samples is displayed in table 5.1.3.

Noise	Correct results	Reject cases	Incorrect results
0.0	100%	0%	0%
0.3	100%	0%	0%
0.5	100%	0%	0%

Table 5.1.3 Successful rate (in %) for 1000 testing samples.

Note that, without using wavelet coefficients, the number of parameters needed to store is 1537, whereas the number of parameters needed to store using wavelet coefficients is only 65.

4. Pick the first five largest coefficients and their corresponding level information from the sorted coefficients. Train the neural network using the clean signal in problem I. Use this resulting matrix to test the cases in problem II. The de-noising property of wavelet multi-resolution can reduce the training time when we have to deal with signals

that are in the same class as noise. The percentage for 1000 testing samples is displayed in table 5.1.4

Noise	Correct results	Reject cases	Incorrect results
0.0	100%	0%	0%
0.3	100%	0%	0%
0.5	99.9%	0.1%	0%

Table 5.1.4 Successful rate (in %) of 1000 testing samples.

Problem III

In this problem, we want to classify two sets of functions.

Let $C_0 = \{y(t) : y(t) = A_1 \exp(-a_1 t) \sin(b_1 t + c_1) + A_2 \exp(-a_2 t) \sin(b_2 t + c_2), \text{ where } t_{\min} \leq t$

$< t_{\max}, a_{\min} < a_1, a_2 < a_{\max}, b_{\min} < b_1, b_2 < b_{\max}, 0 < c_1, c_2 < 2\pi\}$ be class 0,

$C_1 = \{y(t) : y(t) = A_1 \exp(a_1 t) \sin(b_1 t + c_1) + A_2 \exp(a_2 t) \sin(b_2 t + c_2), \text{ where } t_{\min} \leq t < t_{\max},$

$a_{\min} < a_1, a_2 < a_{\max}, b_{\min} < b_1, b_2 < b_{\max}, 0 < c_1, c_2 < 2\pi\}$ be class 1. Picking the first five

largest wavelet coefficients and their corresponding level information from the sorted

coefficients. Let the training set be generated from 1000 random samples from C_0 ,

where $0 \leq t < 4, 0.2 < a_1, a_2 < 1, 2 < b_1, b_2 < 10, A_1, A_2 = 1$ and 1000 random samples from

C_1 , where $0 \leq t < 4, 0.2 < a_1, a_2 < 1, 2 < b_1, b_2 < 10, A_1, A_2 = 1$, and let the testing set be

generated from 500 random samples from C_0 and 500 random samples from C_1 . In

figure 5.1.13, a sample input from class 1 is shown, and in figure 5.1.14, a sample input

from class 0 is shown. The result is shown in table 5.1.5.

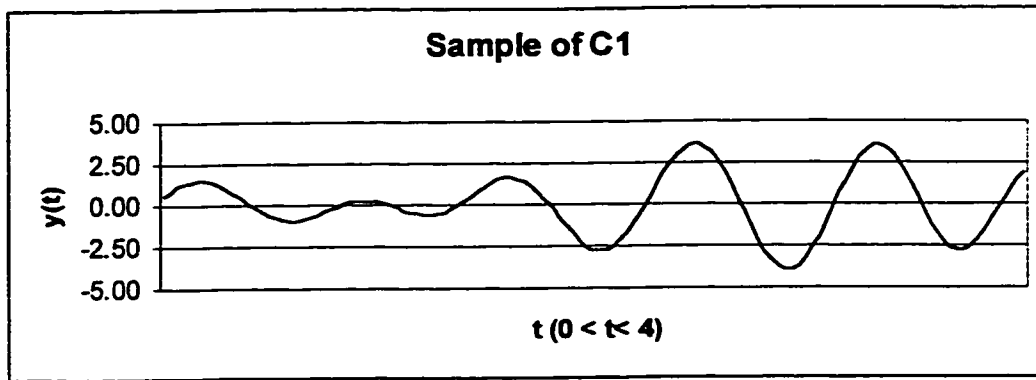


Figure 5.1.13 Example of the function in C_1 .

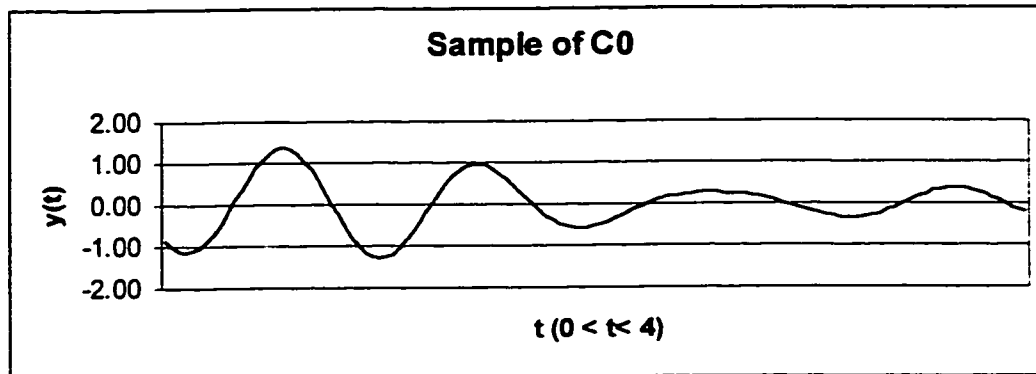


Figure 5.1.14 Example of the function in C_0 .

The following result is obtained:

Correct results	Reject cases	Incorrect results
99.7%	0.2%	0.1%

Table 5.1.5 Successful rate (in %) for 1000 testing samples.

5.2 Parameters Extraction Problem I

This problem is to extract the frequency and damping coefficients from a simulated flutter signal. This problem has also been studied by Lee and Wong [9].

Problem IV :

In this problem, we want to use a neural network to extract two parameters, a and b , from a given function $y(t) = A\exp(-at)\sin(2\pi bt)$, where $t_{\min} \leq t < t_{\max}$, $a_{\min} < a < a_{\max}$, $b_{\min} < b < b_{\max}$. Here, a and b denote the value of damping and frequency, respectively.

Method 5.2.1: Use one neural network to find two parameters, see figure 5.2.1.



Figure 5.2.1 Method 5.2.1.

The input p is a vector of 512 discrete points of $y(t)$, where $A=1$, $0 < t \leq 2$, $0.3 < a < 0.8$, $3 < b < 8$. The output out is a vector with two entries. The first entry out_1 gives the approximation of a , and the second entry out_2 gives the approximation of b . This method uses a two-layer feedforward neural network with 35 neurons in the hidden layer.

(Note that W_1 is a 35×512 matrix, W_2 is a 2×35 matrix, b_1 is a 35×1 vector and b_2 is a 2×1 vector.) The backpropagation (steepest descent) momentum and adaptive learning rate method is used to train the network. The training set is a set of input vectors (i.e. p) and the corresponding target output vectors $tout$. The testing set is a set of input vectors. To construct the training set, we start with 200 random samples, and, after some training, construct a testing set of 500 for each testing case; if the relative error is

more than ϵ (say 10%), then increase the training data until the testing set gives an acceptable error. In figure 5.2.2, the sample input (512 discrete points of $y(t) = \exp(-0.45t) \sin(2\pi 6t)$) is given. In figure 5.2.3 and figure 5.2.4, the relative error (in %) is given using 1000 randomly generated samples. This relative error is defined by $\text{relative error} = 100 \text{abs}(\text{out} - \text{exact solution}) / \text{exact solution}$. We used about 4 to 5 weeks to train this network. If the training time increases, then the accuracy can be improved. The total size of all weight matrices and bias vectors is 18,027, and this is the storage requirement of method 5.2.1.

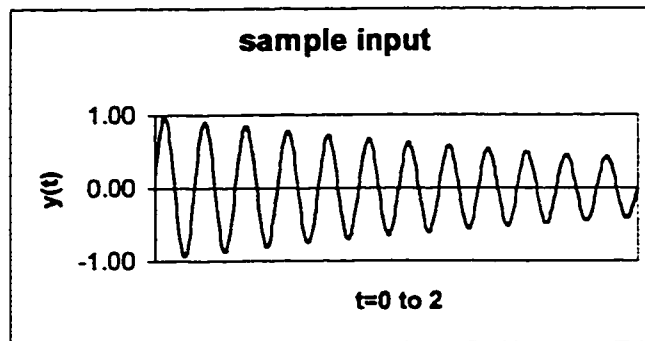


Figure 5.2.2 Input of the neural network using 512 points.

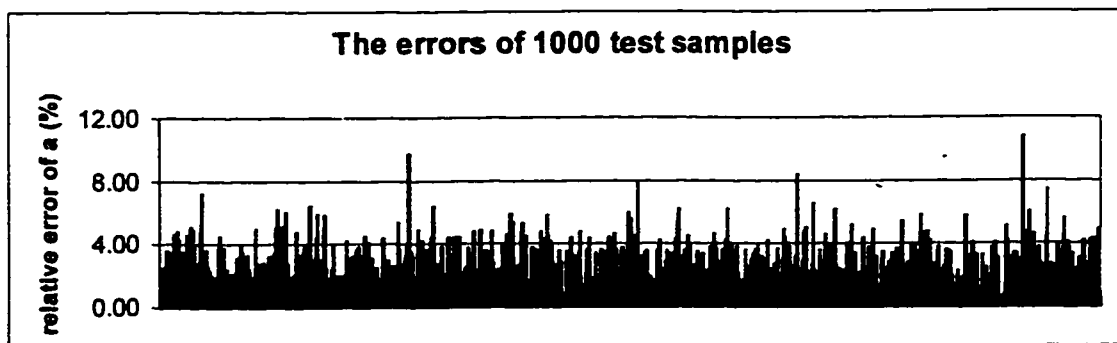


Figure 5.2.3 Relative error (in %) of the first output (i.e. a).

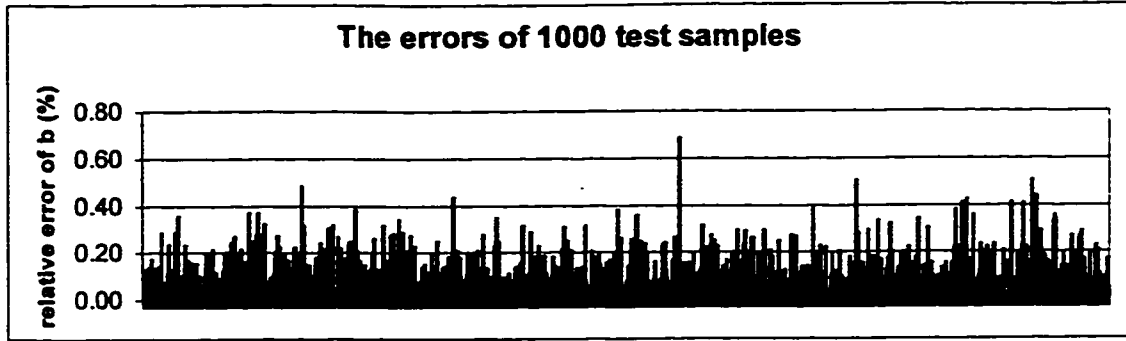


Figure 5.2.4 Relative error (in %) of the second output (i.e. b).

Method 5.2.2: Use two neural networks to find two parameters, see figure 5.2.5.

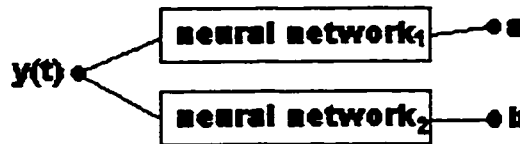


Figure 5.2.5 Method 5.2.2.

The input vector p is a vector of 256 discrete points of $y(t)$ where $A=1$, $0.5 < t \leq 1.5$, $0.3 < a < 0.8$, $3 < b < 8$. The outputs out_1 and out_2 are two real numbers that give the approximation of a and b , respectively. Two two-layer feedforward neural networks with 18 neurons in the hidden layer are used. (Note that W_1 is a 18×256 matrix and W_2 is a 1×18 matrix; also, for both networks, b_1 is a 18×1 vector and b_2 is a 1×1 vector networks.) The training method, the way to construct the training set and the testing set are the same as in method 5.2.1. In figure 5.2.6, the sample input (256 discrete points of $y(t) = \exp(-0.45t) \sin(2\pi 6t)$) is given. In figure 5.2.7 and figure 5.2.8, the relative error (in %) is given using 1000 random generated samples. We used 2 to 3 weeks to train the first network and about 3 days to train the second network. The storage requirement of the neural networks in method 5.2.2 is 9,290. Since the number of both inputs and neurons in these neural networks decrease, the time to train these networks

decreases. The accuracy of the result in figure 5.2.7 is also better than that of the result in figure 5.2.3. In this method parallel computation can be easily applied.

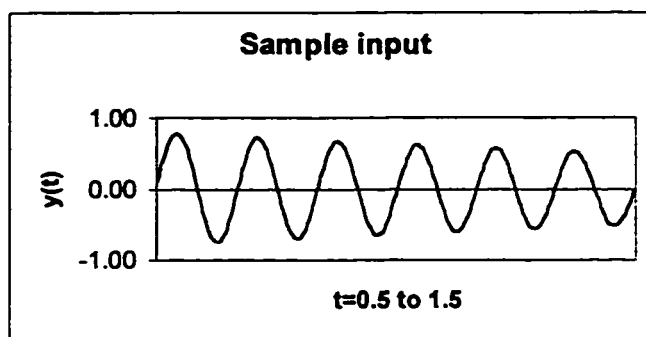


Figure 5.2.6 Example of the input of the neural network.

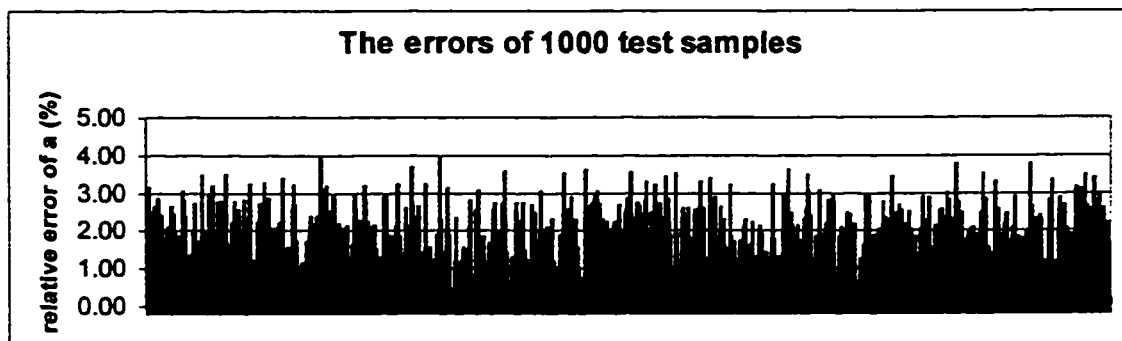


Figure 5.2.7 Relative error (in %) of the first output (i.e. a).

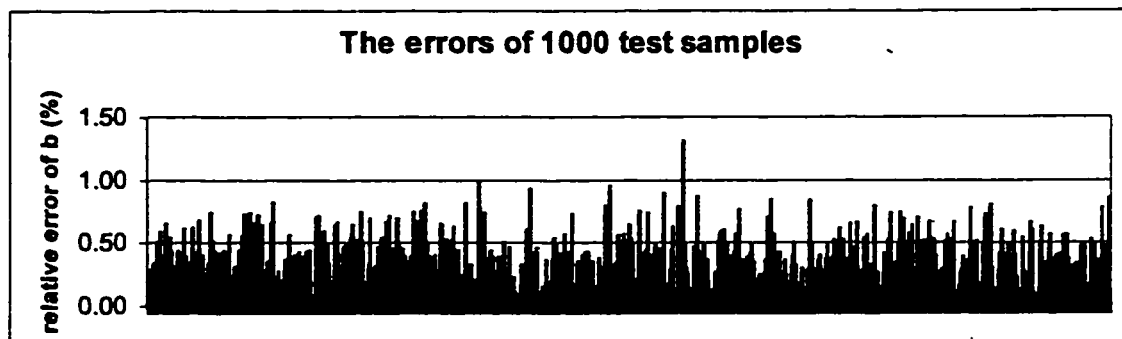


Figure 5.2.8 Relative error (in %) of the second output (i.e. b).

Method 5.2.3: Use two neural networks to find two parameters. One of the networks uses wavelet compression and the other reduces the number of discrete points to 64. See figure 5.2.9.



Figure 5.2.9 Method 5.2.3.

Network1: The input **wavecoeff** is a vector of 60 entries; the first 30 entries are the first 30 largest squared-wavelet coefficients, and the next 30 entries are their corresponding positions. These coefficients are the wavelet coefficients of $y(t)$ where $A=1$, $0.5 < t \leq 1.5$, $0.3 < a < 0.8$, $3 < b < 8$. The network provides the approximation of a . A two-layer feedforward neural network with 18 neurons in the hidden layer is used. (Note that W_1 is a 18×60 matrix, W_2 is a 1×18 matrix, b_1 is a 18×1 vector and b_2 is a 1×1 vector.

Applying the same training method in method 5.2.1 the result is shown in figure 5.2.13. Figure 5.2.10 shows the first 30 entries of **wavecoeff**, and figure 5.2.11 shows the next 30 entries. The data from the coefficients in figure 5.2.12 are the wavelet coefficients of 256 discrete points of $y(t) = \exp(-0.45t)\sin(2\pi 6t)$. It took about 10 days to train this network. The number of parameters in the network was reduced to 1,117.

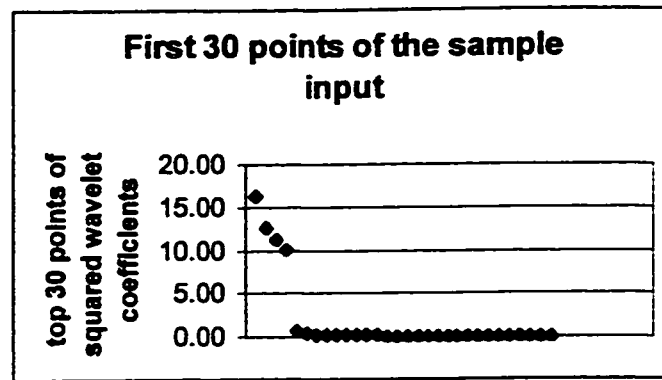


Figure 5.2.10 First 30 points of `wavecoeff`.

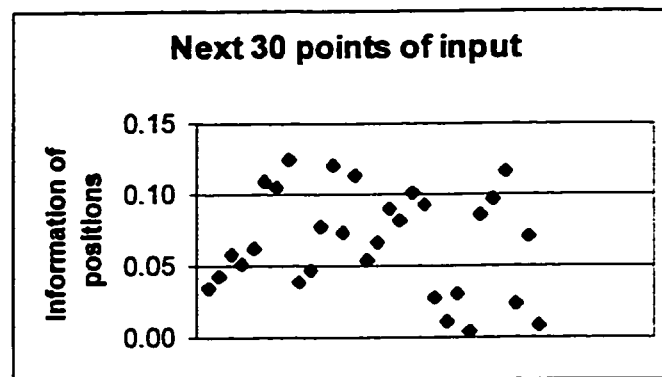


Figure 5.2.11 Next 30 points of `wavecoeff`.

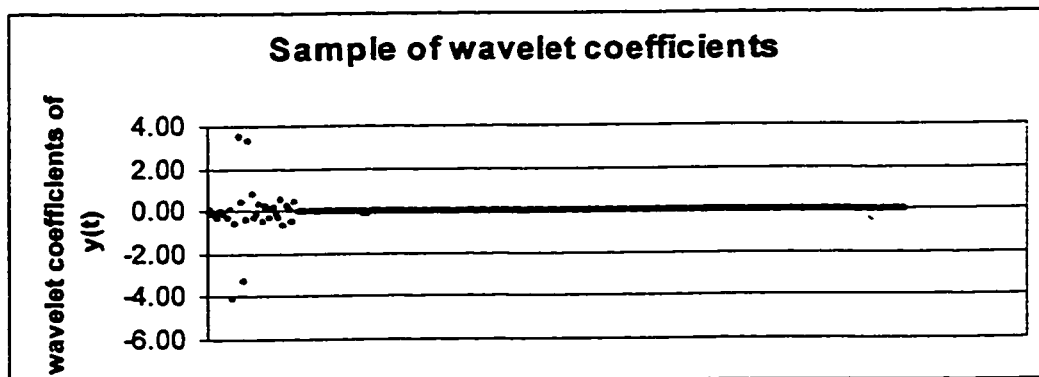


Figure 5.2.12 256 wavelet coefficients of $y(t)$.

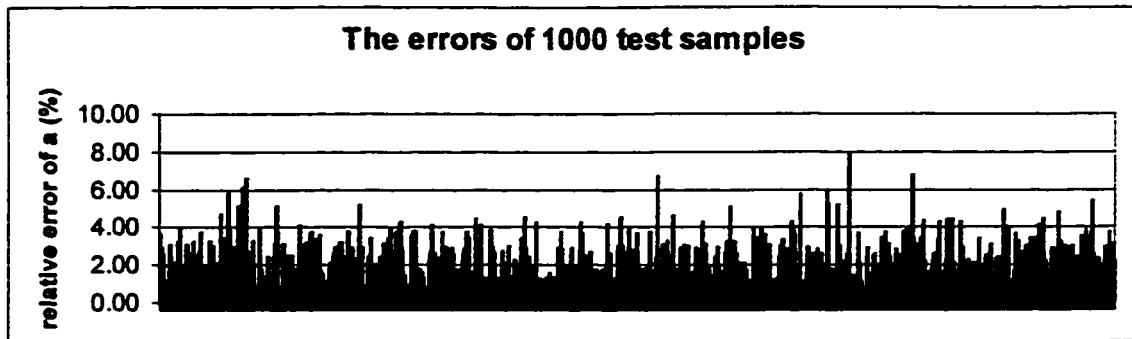


Figure 5.2.13 Relative error (in %) of the first output (i.e. a).

Network 2: The input \mathbf{p} is a vector of 64 discrete points of $y(t)$, where $A=1$, $0.5 < t \leq 1.5$, $0.3 < a < 0.8$, $3 < b < 8$. The network gives the approximation of b . Fourteen hidden neurons are used for a two-layer feedforward in the network. (Note that \mathbf{W}_1 is a 14×64 matrix, \mathbf{W}_2 is a 1×14 matrix, \mathbf{b}_1 is a 14×1 vector and \mathbf{b}_2 is a 1×1 vector.) In figure 5.2.14, the sample input is given and the relative error is shown in figure 5.2.15. Only one day was used to train this network and the number of parameters was reduced to 925.

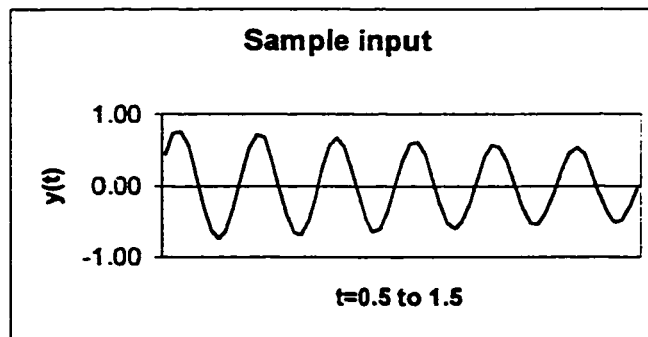


Figure 5.2.14 Sample input of network 2.

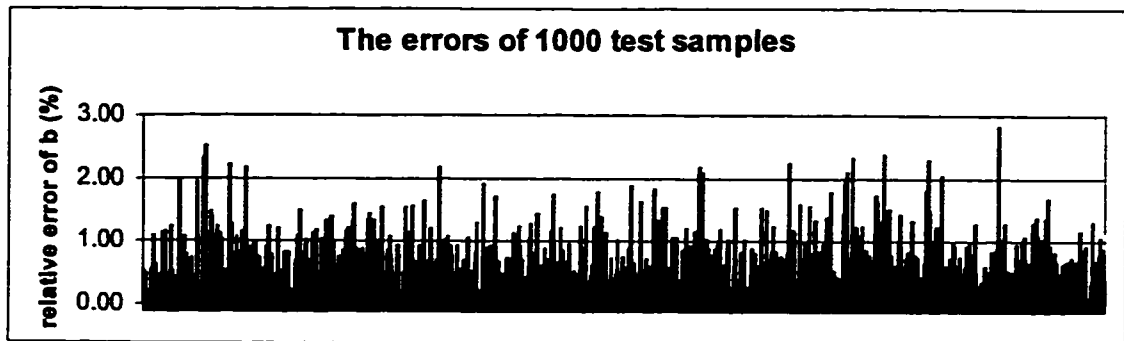


Figure 5.2.15 Relative error (in %) of the second output (i.e. b).

Remarks:

1. Method 5.2.1 needs to store 18,027 real numbers, but method 5.2.3 only needs 2042.
2. For each loop of training, the CPU time for method 5.2.1 takes longer than method 5.2.3.
3. In method 5.2.3, using the wavelet transformation could decrease the convergent rate of the training. The number of wavelet coefficients could be changed. Start from 10 coefficients and add 5 coefficients if the inverse transformation cannot provide a good approximation of the function, until it does provide a good approximation.
4. Using the same method of training with sufficient time and a sufficient number of neurons, the neural network could provide an accurate approximation (relative error within 5%).

5.3 Parameters Extraction problem II

When flutter signals are complicated and more parameters need to be determined, direct application of the neural network not only requires significant time for the training process, but may also fail to provide an acceptable result. The difficulties are illustrated in the following example. In this problem, we want to extract four parameters, a_1 , a_2 , b_1 , b_2 , from a given function $y(t) = A_1 \exp(-a_1 t) \sin(2\pi b_1 t) + A_2 \exp(-a_2 t) \sin(2\pi b_2 t)$, where A_1 and A_2 are constant, $t_{\min} \leq t < t_{\max}$, $a_{\min} < a_1, a_2 < a_{\max}$, $b_{\min} < b_1, b_2 < b_{\max}$ and $b_1 - b_2 > 0$.

Method 5.3.1: Use one neural network to find all four parameters. See figure 5.3.1.

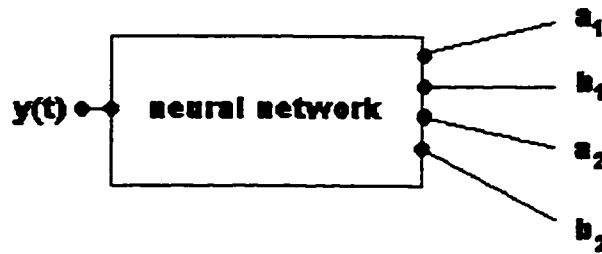


Figure 5.3.1 System of method 5.3.1.

The input p is a vector of 512 discrete points of $y(t)$, where $A_1 = A_2 = 1$, $0 < t \leq 2$, $0.3 < a_1, a_2 < 0.8$, $3 < b_1, b_2 < 8$, $b_1 - b_2 > 0.5$. The output out is a vector with four entries. The first entry out_1 gives the approximation of a_1 , the second entry out_2 gives the approximation of b_1 , the third entry out_3 gives the approximation of a_2 and the fourth entry out_4 gives the approximation of b_2 . There are 50 neurons in the hidden layer of the two-layer feedforward neural network. (Note that W_1 is a 50×512 matrix, W_2 is a 4×50 matrix, b_1 is a 50×1 vector and b_2 is a 4×1 vector.) The training method, the way to construct the training set and the testing set use the same method as in method 5.21.

The condition $b_1 - b_2 > 0$ is a means of ordering these four parameters. Without ordering these four parameters, the neural network will never work. Consider the example given by $y(t) = \exp(-0.3t)\sin(2\pi 6t) + \exp(-0.5t)\sin(2\pi 4t)$. Both vectors $[0.3, 6.0, 0.5, 4.0]$ and $[0.5, 4.0, 0.3, 6.0]$ are acceptable solutions for $[a_1, b_1, a_2, b_2]$, where a_i and b_i denote the value of damping and frequency, $i = 1, 2$. For a given signal, two acceptable solutions cause difficulty in convergence. One way to overcome this problem is to re-order these four parameters by fixing $b_1 - b_2 > 0$. In figure 5.3.2, the sample input (512 discrete points of $y(t) = \exp(-0.45t)\sin(2\pi 3.5t) + \exp(0.54t)\sin(2\pi 6.5t)$) is given. In figure 5.3.3 to figure 5.3.6, the results are plotted. These results took more than one month to train, and the number of parameters in the network is 25,854.

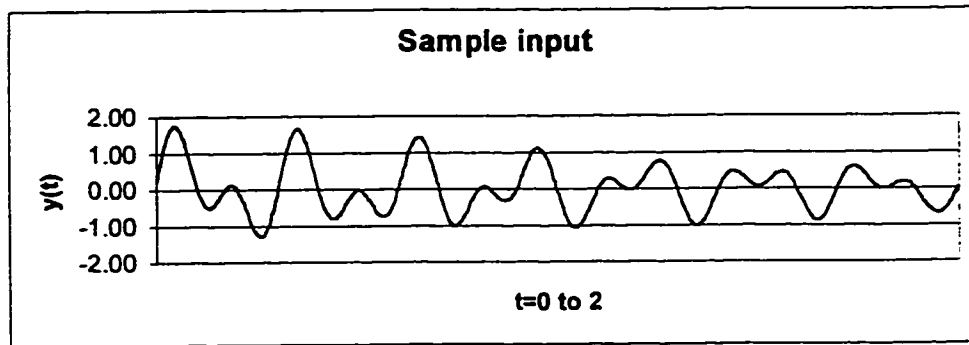


Figure 5.3.2 Sample input of the neural network.

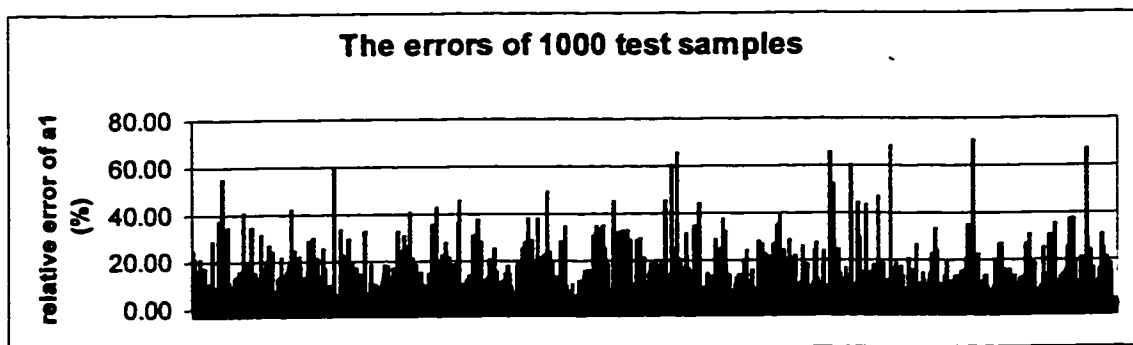


Figure 5.3.3 Relative error (in %) for the first output (i.e. a_1).

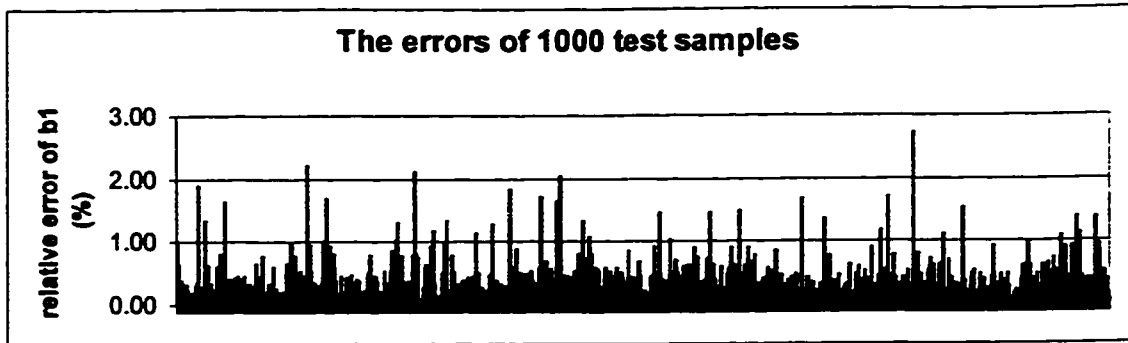


Figure 5.3.4 Relative error (in %) for the second output (i.e. b_1).

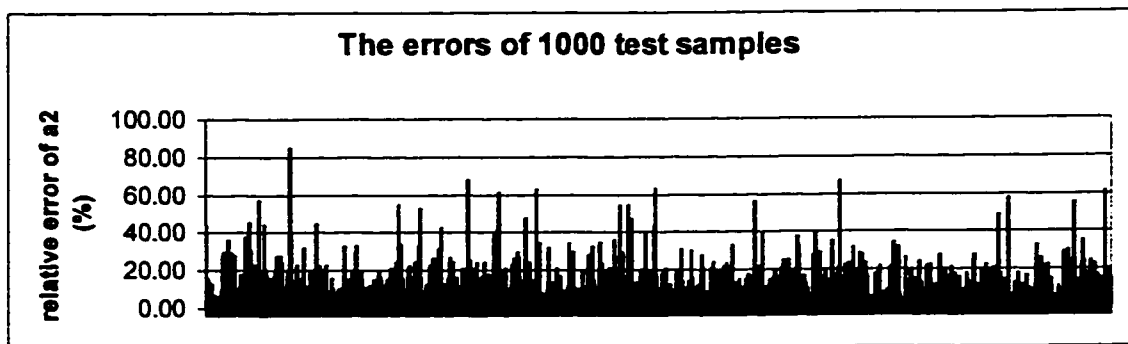


Figure 5.3.5 Relative error (in %) for the third output (i.e. a_2).

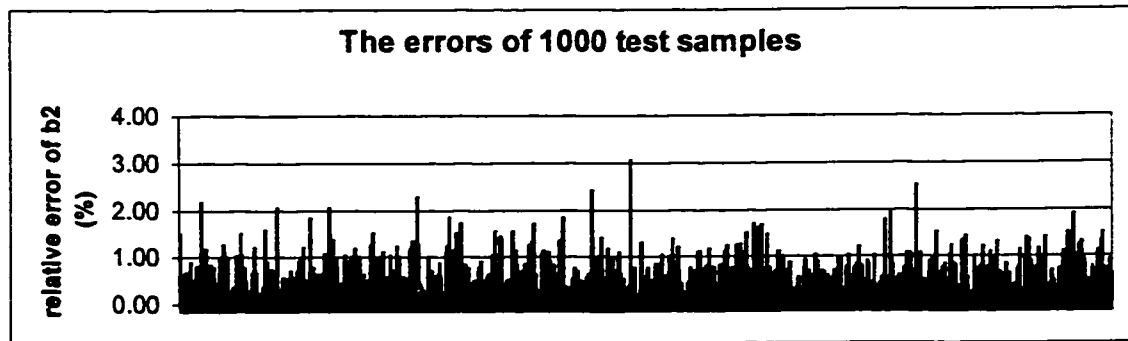


Figure 5.3.6 Relative error (in %) for the forth output (i.e. b_2).

Remarks

1. The frequency values b_1 and b_2 are estimated within 3% error, but the relative errors of a_1 and a_2 are still very large. However, as a greater amount of training time and more neurons are provided in the network, the relative errors in a_1 and a_2 are continuously reduced. Note that most of the relative errors are less than 20%.
2. To improve the performance, we suggest the use of four networks to extract these parameters, as shown in figure 5.3.7. This method will increase the number of weights and biases to store, but it will decrease the training time. Let network_{a_1} , network_{b_1} , network_{a_2} and network_{b_2} be the networks that produce a_1 , b_1 , a_2 and b_2 . Note that the training time to train network_{b_1} and network_{b_2} are much less than for network_{a_1} and network_{a_2} , which required a long time to train.

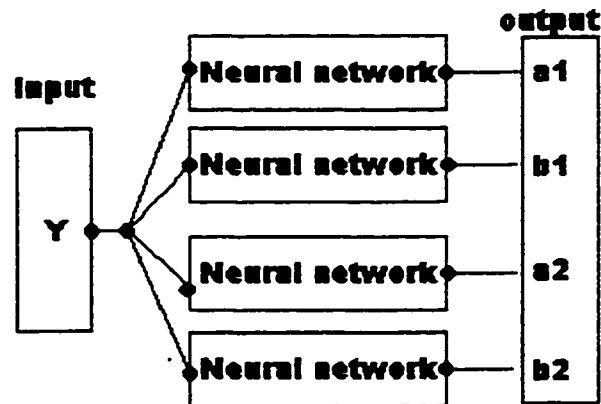


Figure 5.3.7 System for improving performance, as mentioned in remarks 2.

Method 5.3.2: Using wavelet transformation, a further improvement of the method in figure 5.3.7 can be developed, as shown in figure 5.3.8.

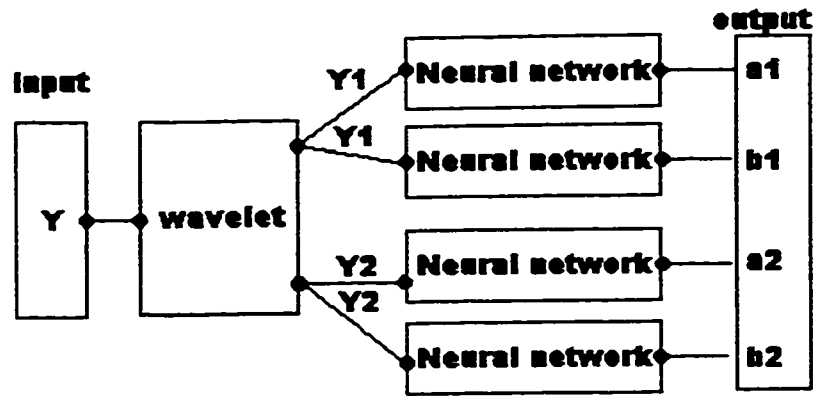


Figure 5.3.8 System for method 5.3.2.

The input Y to the wavelet tool is a vector of 512 discrete points of $y(t) = y_1(t) + y_2(t) = \exp(-a_1 t) \sin(2\pi b_1 t) + \exp(-a_2 t) \sin(2\pi b_2 t)$, where $0 < t \leq 2$, $0.3 < a_1, a_2 < 0.8$, $3 < b_1, b_2 < 8$, $b_1 - b_2 > 0.5$. Note that the condition $b_1 - b_2 > 0$ changes to $b_1 - b_2 > 0.5$ because of the limitation in the separation of two signals (chapter 3). The outputs of the wavelet tool are Y_1 and Y_2 , which are 512 discrete points of the wavelet approximation of $y_1(t)$ and $y_2(t)$. Pick only 256 discrete points from Y_1 and Y_2 for the neural networks. Since Y_1 and Y_2 approximate $y_1(t)$ and $y_2(t)$ very well in the middle parts, we pick the middle parts (i.e. $0.5 < t \leq 1.5$) of Y_1 and Y_2 for the neural networks in method 5.2.2 to improve the accuracy of the overall result. In addition, the neural networks are well trained; no further training is required. In figure 5.3.9 to figure 5.3.12, the graphs show the results of this method.

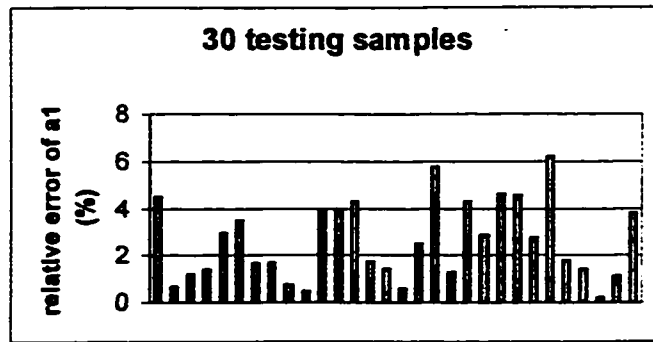


Figure 5.3.9 Relative error (in %) of the first output (i.e. a_1).

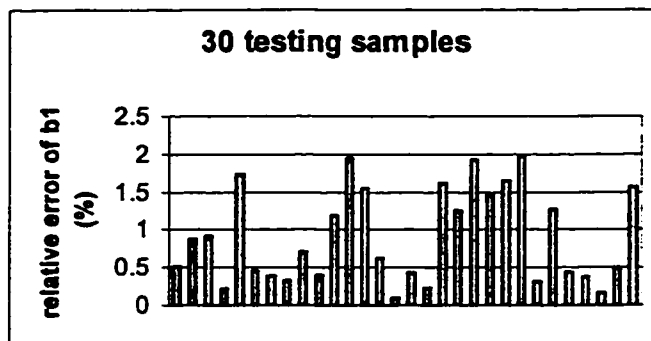


Figure 5.3.10 Relative error (in %) of the second output (i.e. b_1).

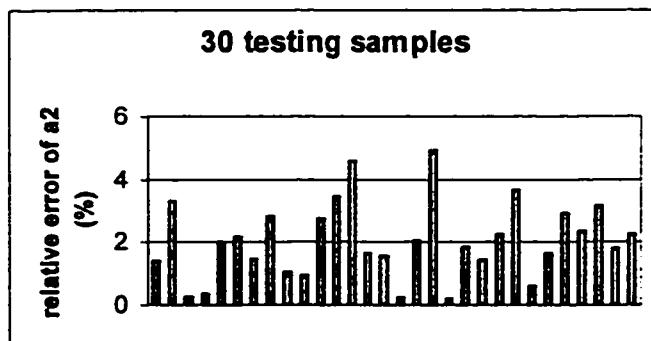


Figure 5.3.11 Relative error (in %) of the third output (i.e. a_2).

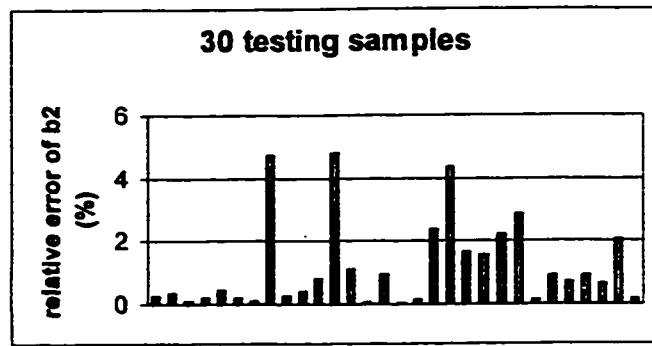


Figure 5.3.12 Relative error (in %) of the forth output (i.e. b_2).

Remarks:

1. Using this method could reduce the training time for neural networks as well as the storage requirements of the neural network.
2. For noisy signals, it is not necessary to train new neural networks, given the de-noising property of wavelets. Let C be a set of functions $\{y(t) : y(t) = (1+0.2\varepsilon)(\exp(-a_1t)\sin(2\pi b_1t) + \exp(-a_2t)\sin(2\pi b_2t))\}$, where $0 < t \leq 2$, $0.3 < a_1, a_2 < 0.8$, $3 < b_1, b_2 < 8$, $b_1 - b_2 > 0.5$, 0.2ε represents 20% noise level. The following result is obtained:

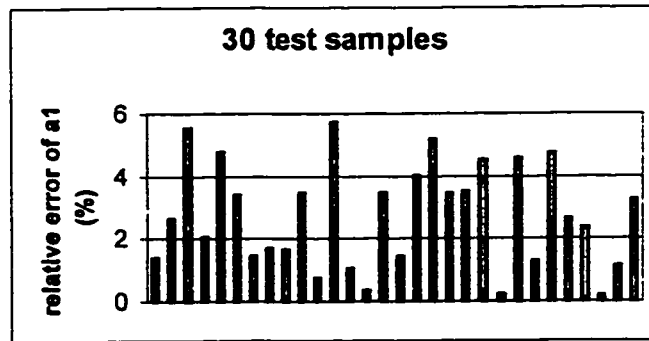


Figure 5.3.13 Relative error (in %) of the first output (i.e. a_1).

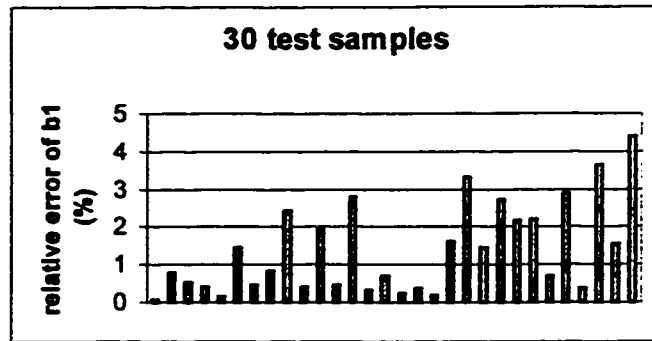


Figure 5.3.14 Relative error (in %) of the second output (i.e. b_1).

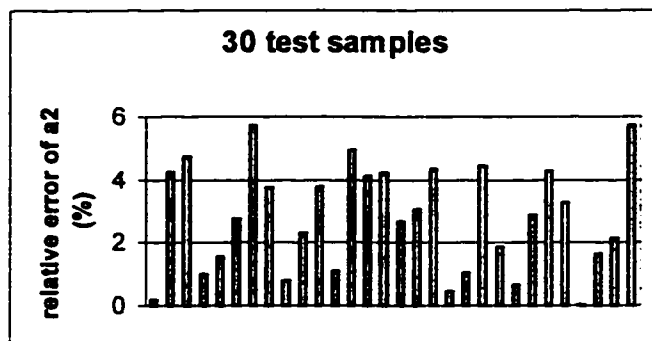


Figure 5.3.15 Relative error (in %) of the third output (i.e. a_2).

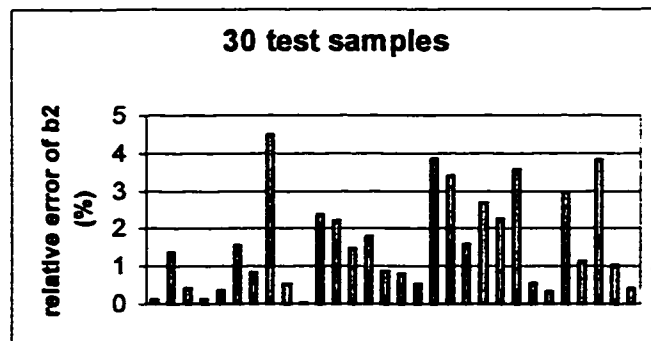


Figure 5.3.16 Relative error (in %) of the forth output (i.e. b_2).

Let the level of noise increase from 20% to 40%. The following result is obtained:

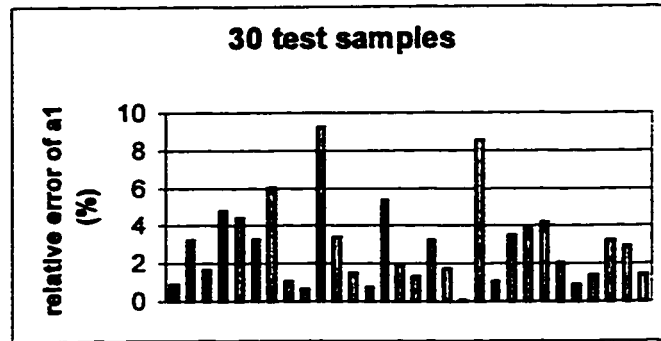


Figure 5.3.17 Relative error (in %) of the first output (i.e. a_1).

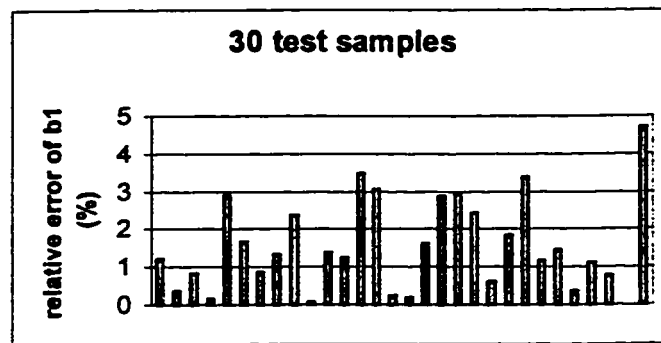


Figure 5.3.18 Relative error (in %) of the second output (i.e. b_1).

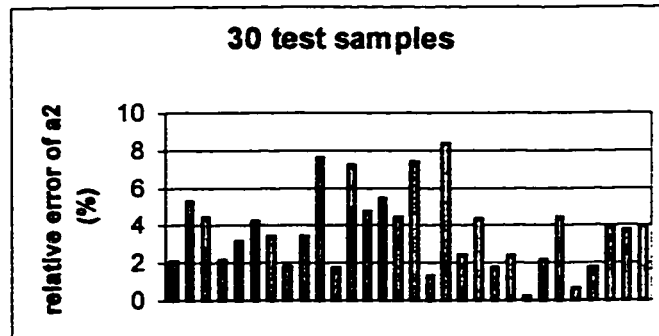


Figure 5.3.19 Relative error (in %) of the third output (i.e. a_2).

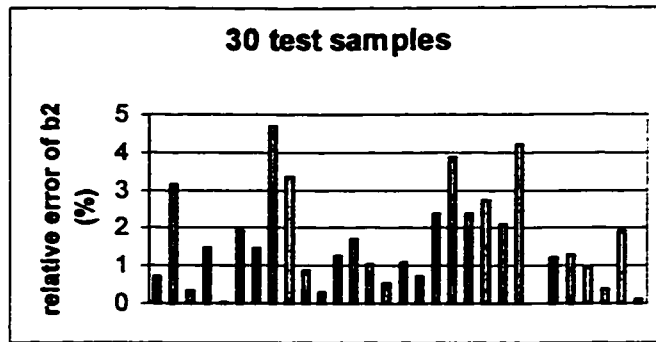


Figure 5.3.20 Relative error (in %) of the fourth output (i.e. b_2).

3. The accuracy of these results depends on the wavelet decomposition and the reconstruction. In this application, relatively more error occurs at the front (t from 0 to 0.5) and the end (t from 1.5 to 2) of the signal approximations. So, using the middle part of the signal ($t=0.5$ to 1.5) will improve the results.

4. To reduce the number of inputs for the neural networks, we can use the wavelet compression as the input. If the signal has N discrete points and assuming $N/10$ wavelet coefficients will be enough to approximate this signal, then the input of the neural networks could be reduced by 90%.

Method 5.3.3: Use properties of wavelet transformation and two neural networks to find these parameters. See figure 5.3.21.

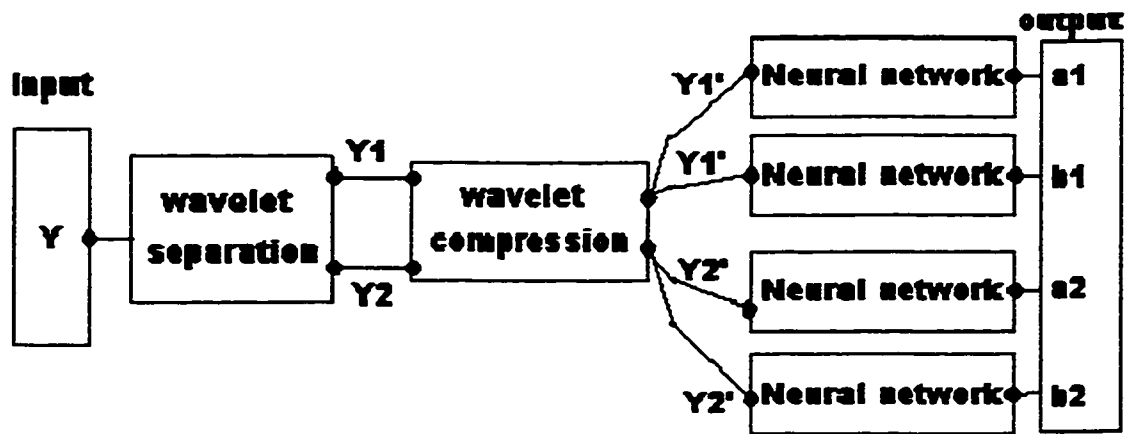


Figure 5.3.21 System of method 5.3.3.

The setting in this method is similar to the setting in method 5.3.2; the only difference is in using the wavelet compression property to reduce the number of inputs to the neural networks. We use 30 wavelet coefficients to find a_1 and a_2 , and use 45 wavelet coefficients to find b_1 and b_2 . Using this method, the results (including the de-noising effect) were similar to those produced by method 5.3.2. In figure 5.3.22 to figure 5.3.25, the results from the clean signal are given, but the result from the noisy signal are not shown.

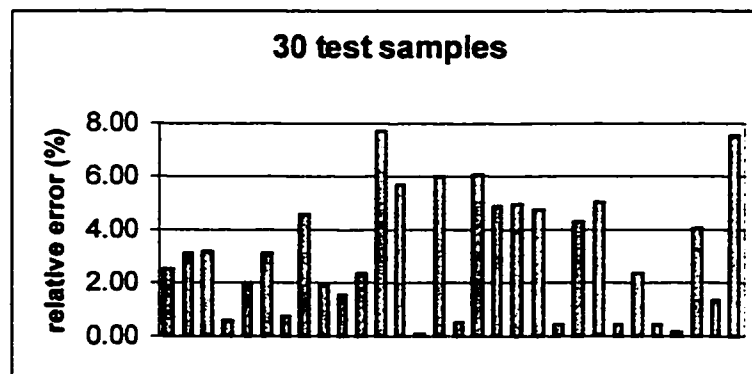


Figure 5.3.22 Relative error (in %) of the first output (i.e. a_1).

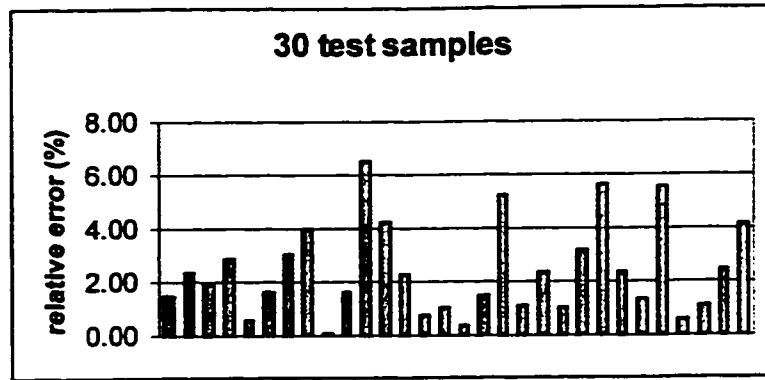


Figure 5.3.23 Relative error (in %) of the second output (i.e. b_1).

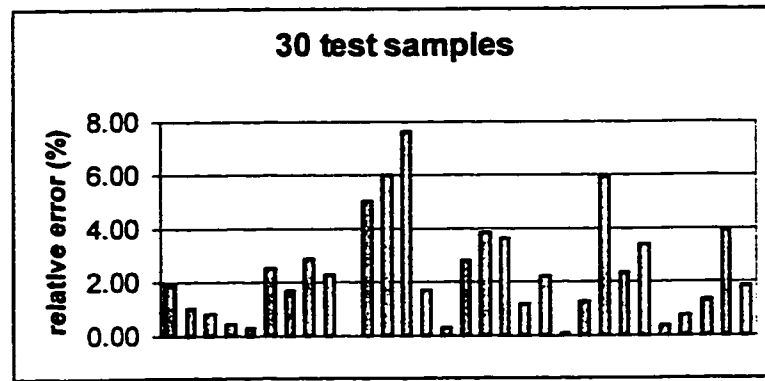


Figure 5.3.24 Relative error (in %) of the third output (i.e. a_2).

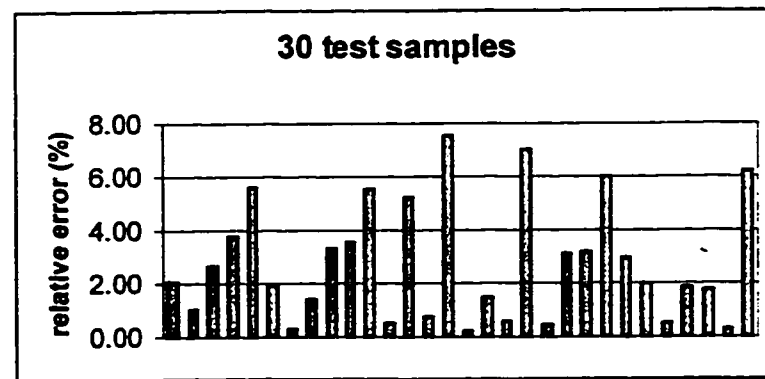


Figure 5.3.25 Relative error (in %) of the fourth output (i.e. b_2).

5.4 Comparison of two Simulating Programs. (Sinon and MATLAB toolbox)

Let C be a set of functions $\{y(t) : y(t) = e^{-at}\sin(bt), \text{ where } 0.5 < t \leq 1.5, 0.3 < a < 0.8,$

$3 < b < 8\}$ and the training set be 800 random samples from C and let the testing set be

500 random samples from C and the number of neurons in the hidden layer be 14.

The input p is a vector of 64 discrete points of $y(t)$ in C . The output is the

approximation of the parameter b . The sample input (64 discrete points of $y(t) = \exp(-$

$0.7725t)\sin(6.7106t)$) is given in figure 5.4.1.

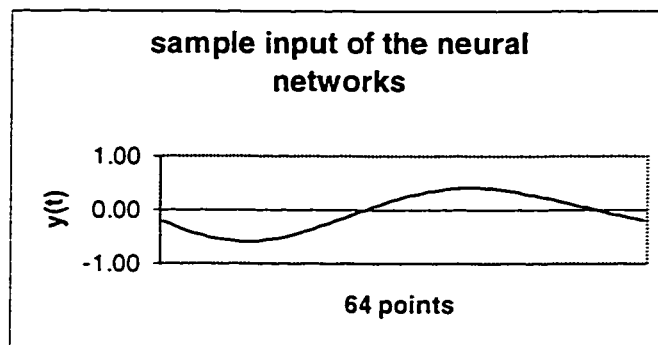


Figure 5.4.1 Sample input of the neural networks.

If 4 discrete points are used in this range of data, the result is shown in figure 5.4.2.

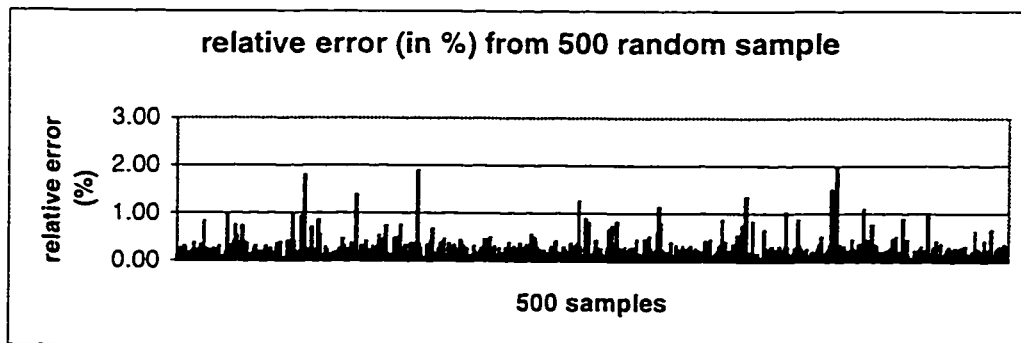


Figure 5.4.2 Result using only 4 discrete points

Using the MATLAB toolbox, the results after 500 to 3,500 loops are given in figure

5.4.3 to 5.4.6. Using Sinon, the results after 1 to 40 loops are given in figure 5.4.7 to

5.4.10.

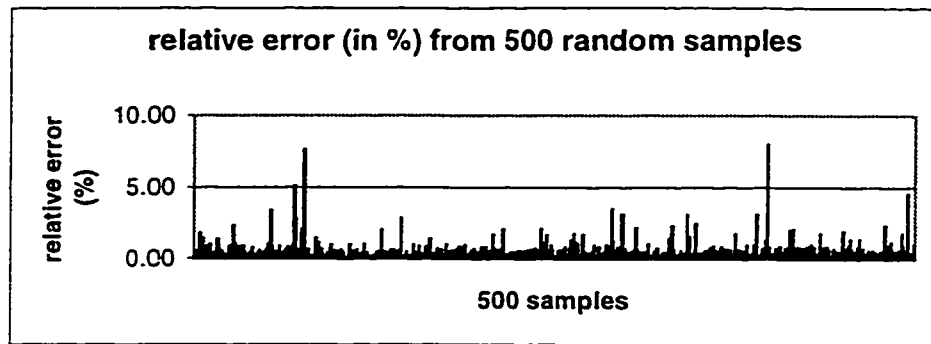


Figure 5.4.3 Relative error of the output after 500 loops.

Note that maximum error is 7.97%.

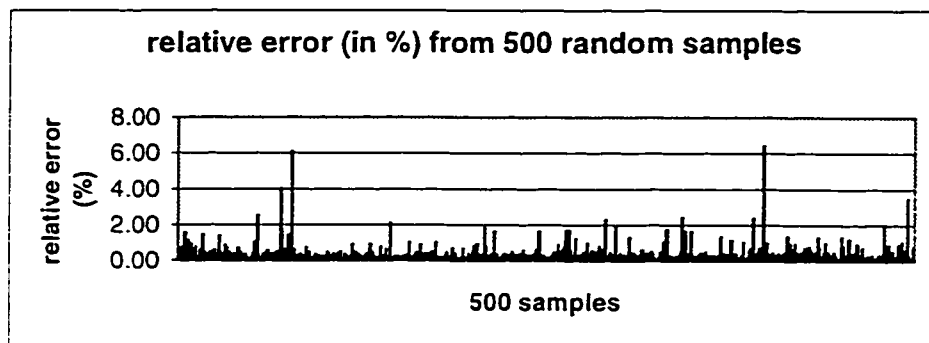


Figure 5.4.4 Relative error of the output after 1000 loops.

Note that maximum error is 6.4%.

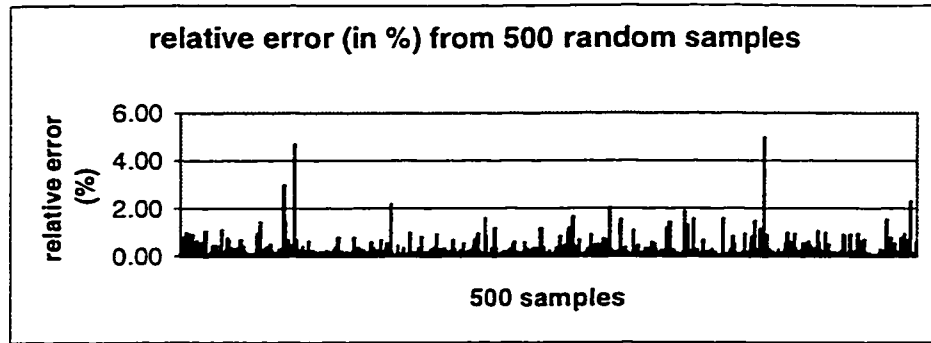


Figure 5.4.5 Relative error of the output after 2000 loops.

Note that maximum error is 4.96%.

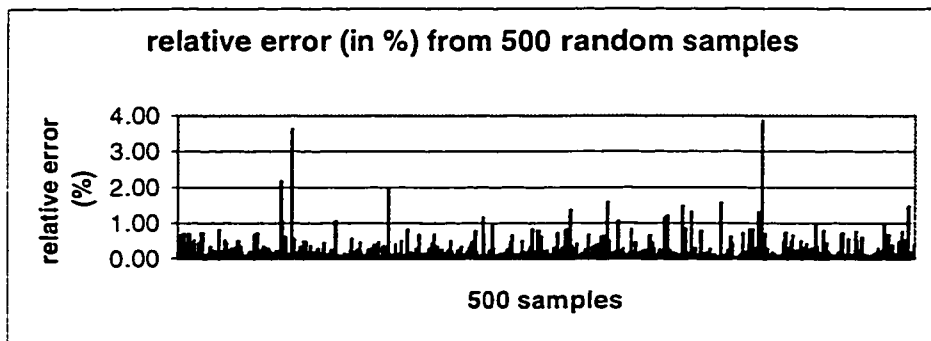


Figure 5.4.6 Relative error of the output after 3500 loops.

Note that maximum error is 3.86%.

Using Sinon:

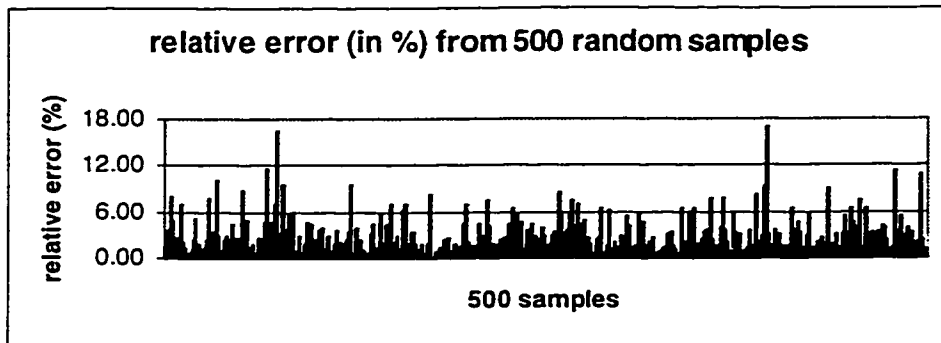


Figure 5.4.7 Relative error of the output after 1 loop.

Note that maximum error is 17.1%.

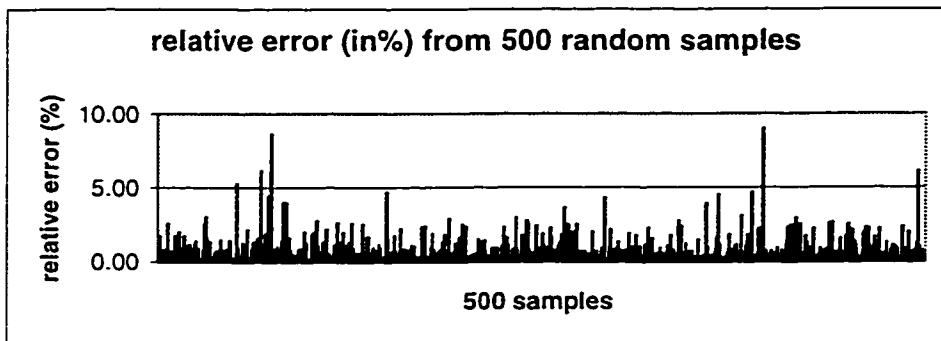


Figure 5.4.8 Relative error of the output after 10 loops.

Note that maximum error is 8.98%.

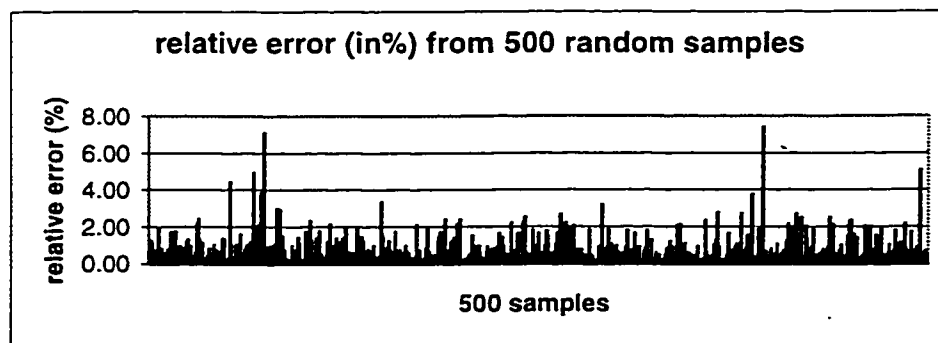


Figure 5.4.9 the relative error of the output after 20 loops.

Note that maximum error is 7.4%.

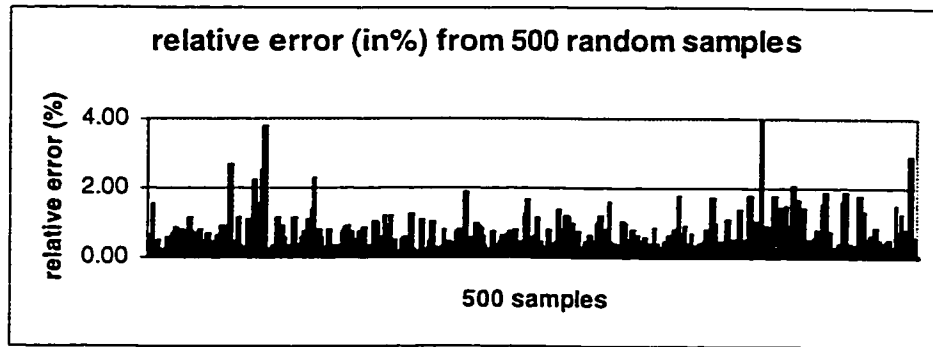


Figure 5.4.10 Relative error of the output after 40 loops.

Note that maximum error is 3.96%.

Remark:

The total time for computation for this particular application is 84 minutes using Sinon and the total time using the MATLAB toolbox is 35 minutes. Since Sinon is an in-house development, there is room for further improvement in the training time for each loop. For example, in the present version of Sinon, the weight matrices are saved for each loop, which requires a significant I/O time. On the other hand, the weight matrix coefficients are saved only at the end of the computation in MATLAB. In terms of relative error, the final results for both programs are about the same.

CHAPTER 6

CONCLUSION

A neural network has the capability to model highly complex nonlinear systems with only limited information. A neural network consists of many simple computing units known as neurons, which are connected to other units by a weighting function and operate in parallel.

In this thesis, we developed a two-layer neural network model, and the network was then applied to study the classification and extraction of aerodynamic parameters from simulated flight flutter data. The flutter data analysis can be regarded as an aircraft parameter estimation problem in which, given data from a flight test, we want to accurately estimate the value of the frequency and damping coefficients. The success of flutter analysis plays a significant role in the design of aircraft.

In the classification problems, when applying the neural network to a clean signal and to those corrupted with up to 50% noise, we were able to classify more than 95% of the correct type. Moreover, when it is used in conjunction with a wavelet multi-resolution method, we could achieve 100% correct classification for both clean and noisy signals. The wavelet multi-resolution method was introduced to perform data compression and de-noising of a noisy signal. Furthermore, as a direct consequence of reducing the complexity of the neural network architecture, not only the storage requirement of the neural network is reduced, but the associated training time is also significantly decreased.

For the parameter extraction problem, we first considered a simple case in which the signal contains only one frequency and one damping coefficient, and this will be referred to as Problem IV. It has been demonstrated that the two-layer network that has been developed can be trained to extract the two parameters within 5% error. However, when the signal consists of multiple signals each of which has its own damping and frequency, the problem becomes much more complex than when dealing with only one signal, as in Problem IV. To illustrate the problem, consider the case in which the signal consists of two exponential decaying sine waves, which will be referred to as Problem V. First, by direct application of the neural network, the performance index (i.e. the sum of squares of the error) fails to converge in the training process because, for a given input, more than one output is expected. To achieve convergence, the training data must be reordered so that the first output set corresponds to the damping and the associated largest frequency, and the second output set to the damping and the associated smallest frequency. Although this measure reduces the performance index in the right direction, the training time that is required becomes enormous. The training time required for Problem IV is about ten days on a pentium PC with 133 MHz. However, even after more than one month of training time, the training process for Problem V was not yet completed. To overcome the difficulty of slow convergence, it is suggested that the multiple signals first be decomposed using a wavelet transform. Each decomposed signal then contains only one damping and frequency coefficient, and the parameters can be extracted by a neural network, as in Problem IV. To improve the performance of the neural network, we also apply the wavelet multi-resolution technique to each

decomposed signal, and use the wavelet coefficients as input to the neural network.

Instead of using a neural network to deal with each decomposed signal, it is suggested that two neural networks be applied for each signal so that only one output neuron, namely one for damping and one for frequency, is needed. Hence, for multiple n signals, the network architecture consists of $2n$ neural networks. However, they are operating in parallel, and thus the training time required can be significantly reduced. The parallel implementation is also very attractive when considering applying the network for parameter extraction in a real time environment.

Finally, based on our experience of the development and application of neural networks, the following comments are offered:

1. The success of the present neural network depends on, first, the ability to decompose multiple signals using a wavelet transformation. Secondly, sufficient training data sets are provided to train the neural network using a supervised learning algorithm. In many real applications, only limited training data may be available, and the use of unsupervised learning algorithms or recurrent neural networks may offer better results than the network presented in this thesis.

2. Only one hidden layer is used in this thesis. It may be of interest to investigate the performance of the neural network by introducing additional hidden layers.

3. The number of neurons used in the hidden layer could affect the training process in the neural network. In this thesis, the number of neurons is taken as approximately equal to

the square root of (IO) , where I and O denote the number of neurons at the input and output, respectively. Depending on the problem and the size of the input data, too many or too few neurons for the hidden layer may cause difficulty in the convergence of the performance index.

REFERENCE

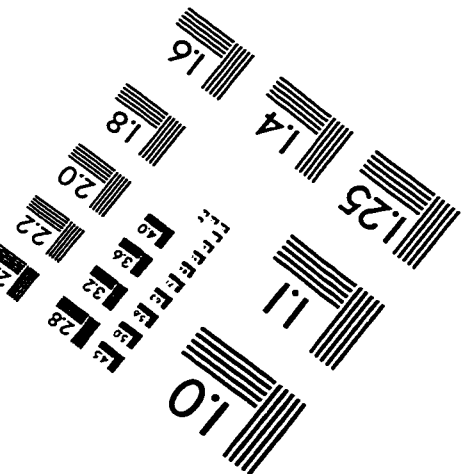
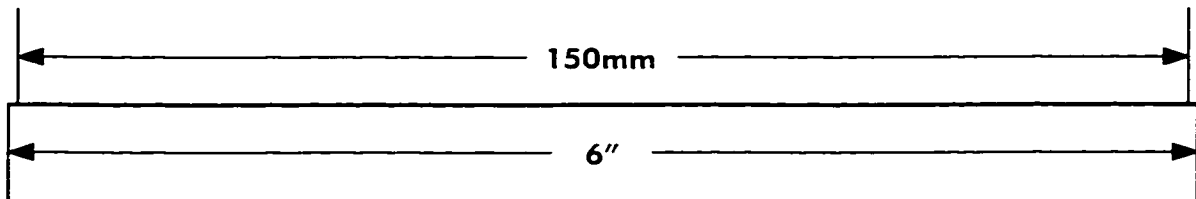
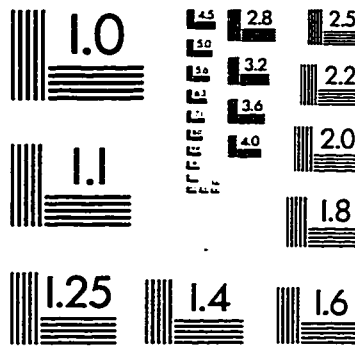
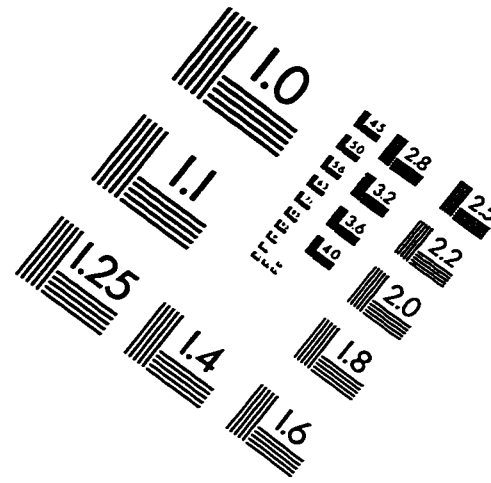
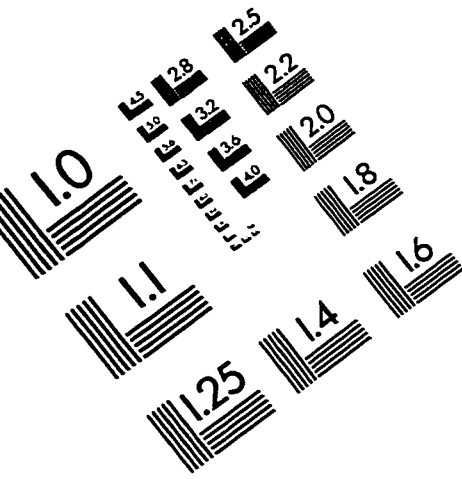
- [1]. T.-C. Chu, Harold Sze. An Artificial Neural Network for Naval Theater Ballistic Missile Defense Program, IEEE International Conference on Neural Networks, Volume 1,1997.
- [2]. R. Barton, D. Himmelblau. On-line Prediction of Polymer Product Quality in an Industrial Reactor Using Recurrent Neural Networks, IEEE International Conference on Neural Networks, Volume 1,1997.
- [3]. Y. Shin, K.-S. Jin, B.-M. Yoon. A complex Pi-Sigma Network and Its Application to Equalization of Nonlinear Satellite Channels, IEEE International Conference on Neural Networks, Volume 1,1997.
- [4]. A. Iwate, Y.Nagasaka, N. Suzumura. Data Compression of the ECG Using Neural Network for Digital Holter Monitor, IEEE Engineering in Medicine and Biology Magazine, September, 1990.
- [5]. Krzysztof J. Cios, Keqing Chen, Robert A. Langenderfer. Use of Neural Networks in Detecting Cardiac Diseases from Echocardiographic, IEEE Engineering in Medicine and Biology Magazine, September, 1990.
- [6]. R.M. Holdaway, M.W. White, A. Marmaron. Classification of Somatosensory Evoked Potentials Recorded from Patients with Severe, IEEE Engineering in Medicine and Biology Magazine, September, 1990.
- [7]. A. Hiraiwa, K. Shimohara, Y. Tokunaga. EEG Topography Recognition by Neural Networks, IEEE Engineering in Medicine and Biology Magazine, September, 1990.

- [8]. J. J. Kaufman, A. Chiabera, M. Hatem, N. Z. Hakim, M. Figueiredo, P. Nasser, S. Lattuga, A. A. Pilla, R. S. Siffert. A Neural Network Approach for Bone Fracture, IEEE Engineering in Medicine and Biology Magazine, September, 1990.
- [9]. B. H. K. Lee and Y. S. Wong. Neural Network Parameter Extraction with Application to Flutter Signals, Journal of Aircraft, volume 35, number 1997.
- [10]. Timothy Masters. Practical Neural Network Recipes in C++. Academic Press, 1993, chapter 12.
- [11]. Gilbert Strang, Truong Nguyen. Wavelets and Filter Banks. Wellesley-Cambridge Press, 1996, chapter 6.
- [12]. Pao, Yoh-Han. Adaptive Pattern Recognition and Neural Networks. Addison-Wesley Publishing Co., Reading, M. A., 1989.
- [13]. Specht, Donald. Enhancements to Probabilistic Neural Networks, International Joint Conference on Neural Networks, Baltimore, M. D. 1992.
- [14]. Meisel, W. Computer-Oriented Approaches to Pattern Recognition. Academic Press, New York, 1972.
- [15]. J. Alianna, Maren et al. HandBook of Neural Computing Applications, Academic Press, Toronto, Ont., 1990
- [16]. Martin T. Hagan, Howard B. Demuth, Mark Beale. Neural Network Design. PWS Publishing Company, 1995.
- [17]. J. Li, A. N. Michel, W. Porod. Analysis and Synthesis of a Class of Neural Networks: Linear system operating on a closed hypercube, IEEE Transactions on Circuits and Systems, volume 36, November, 1989.

- [18]. D. Rumelhart and J. McClelland, editors. Parallel Data Processing, volume 1, Chapter 8, M. I. T. Press, Cambridge, MA 1986.
- [19]. F. Rosenblatt. Principles of Neurodynamics, Spartan Press, Washington D.C., 1961.
- [20]. D. White and D. Sofge, eds. Handbook of Intelligent Control, Van Nostrand Reinhold, New York, 1992.
- [21]. T. Tollenaere. SuperSAB: Fast Adaptive Back Propagation with Good Scaling Properties, Neural Networks, volume 3, number 5, 1990.
- [22]. C. Charalambous. Conjugate Gradient Algorithm for Efficient Training of Artificial Neural Networks, IEEE Proceedings, volume 139, number 3, 1992.
- [23]. M. T. Hagan and M. Menhaj. Training Feedforward Networks with the Marquardt algorithm, IEEE transactions on Neural Networks, volume 5, number 6, 1994.
- [24]. L. E. Scales. Introduction to Non-Linear Optimization, Springer-Verlag, New York, 1985.
- [25]. Recommendation H.262, ISO/IEC 13818, Generic Coding of Moving Picture and associated audio, Draft international standard.
- [26]. D. Estaban and C. Galand. Application of Quadrature Mirror Filters to Split-band Voice Coding Schemes, Proc. IEEE International Conference ASSP, 1977.
- [27]. Q. Jin, K. M. Wong, and Z. Q. Luo, Design of an Optimum Wavelet for Cancellation of Long Echoes in Telephony, Proc. IEEE SP Int. Symp. On Time-Freq., 1992.

- [28]. G.Kaiser. Quantum Physics, Relativity and Complex Spacetime: Towards a New Synthesis, Amsterdam, North-Holland, 1990.
- [29]. D. Donoho. De-noising by soft thresholding, IEEE Trans. Inf. Th. 41,1995.
- [30]. B. H. K. Lee and Laichai, F. Development of Post-Flight and Real Time Flutter Analysis Methodologies, Proceedings From International Aéroélasticité et Dynamique de structures (Strasbourg), Association Aéronautique et Astronautique de France, 1993.

IMAGE EVALUATION TEST TARGET (QA-3)



APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989

© 1993, Applied Image, Inc., All Rights Reserved

