WAVELETS AND ARTIFICIAL NEURAL NETWORKS IN POWER SYSTEM TRANSIENT CLASSIFICATION AND SHORT-TERM POWER LOAD PREDICTION

By

Fan Mo

A Thesis
Submitted to Faculty of Graduate Studies
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba, Canada

Thesis Advisor: W. Kinsner, Ph. D., P. Eng.

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION PAGE

WAVELETS AND ARTIFICIAL NEURAL NETWORKS IN POWER SYSTEM TRANSIENT CLASSIFICATION AND SHORT-TERM POWER LOAD PREDICTION

BY

FAN MO

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirements of the degree of

Master of Science

ABSTRACT

This thesis focuses on applications of wavelets and artificial neural networks in power system transients analysis, modelling, classification, and short-term power load prediction. A power system transient classification system framework based on wavelet transform preprocessing and probabilistic neural networks (PNN) is proposed in the thesis. A new type of neural networks, resource allocating networks (RAN), which can adjust its computing structure dynamically, is also investigated for the short-term power load prediction.

Wavelet modelling of power system transients is studied by examining (i) the capability of multiresolution analysis, (ii) time-scale representation of transient signals, and (iii) accurate detection and compact representation of transient signals, which are useful for power system transient recording, storing, and classification.

The PNN is used as a classifier in the proposed transient classification system. Experimental results show that the PNN has a great speed advantage in its training over backpropagation neural networks (BPN), which makes it a good candidate for real-time usage. The wavelet transform preprocessing of the transient signals improved the performance of the PNN, and demonstrated the feasibility of the real-time transients recording and automatic classification.

The RAN is studied for short-term power load prediction because of the capability of RAN's adjusting its computing structure dynamically, which makes it a good candidate for modelling nonstationary signals. Experimental results on real data from Manitoba Hydro revealed that the short-term power load prediction is more accurate than other published results.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Kinsner for proposing this interesting topic and giving me his time and patience throughout the process of this thesis. I would also like to thank Dr. Kinsner for his major improvements and editing of this thesis document.

I would like to thank Manitoba Hydro for giving me great financial support for this thesis. Without their support, it would be impossible for me to finish this thesis.

I would like to acknowledge everybody in our Delta Group, past and present, including Bin Huang, Jin Chen, Richard Dansereau, Eric Jang, Rasekh Rifaat, Hongjin Chen, Tina, LuoTao Sun, etc. for their help, support, friendship, and useful discussions with me for the thesis.

I would also like to thank my mother and mother-in-law for their unconditional support and inspiration. Above all this thesis is for my wife Ying, my son Bo, and my daughter Kelsey.

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

LIST OF ABBREVIATIONS

ANNs	artificial neural networks
APE	absolute persentage error
BP	backpropagation
BPN	backpropagation network
CWT	continuous wavelet transform
Daub4	Daubechies wavelet with four filter coefficients
DFT	discrete Fourier transform
DWT	discrete wavelet transform
FT	Fourier transform
MAPE	mean value of absolute persentage error
M-RAN	minimal resource allocation network
MSD	multiresolution signal decomposition
PDF	probability density function
PNN	probabilistic neural network
RAN	resource allocating network
RBFN	radial basis function network
RMS	root mean square
SNR	signal to noise ratio
STFT	short-time Fourier transform
WT	wavelet transform

LIST OF SYMBOLS

$c_i[n]$	the smoothed signal at scale i
$d_i[n]$	the detailed signal at scale i
e_n	the prediction error
e_{min}	the minimum prediction error
e_{rmsn}	the RMS value of network output error at nth observation in RAN
$f_k(X)$	probability density function (PDF) for a class k
$g[n]$	high-pass filter for discrete wavelet transform
$h[n]$	low-pass filter for discrete wavelet transform
h_k	prior probability of a class k
l_k	loss function associated with misclassifying a sample of the class k
κ	overlap factor of the hidden units in the input space
N	total number of data points
T	a time instance
ΔT	the time difference between two time instances
$o_k^{(n)}$	output of the kth hidden units at step n
$\left\|o_{max}^{(n)}\right\|$	the largest absolute hidden unit output value at step n
P_c	correct classification rate of the PNN
$r_k^{(n)}$	the normalized output value of RAN
u_{nr}	the stored pattern nearest to X_n in the input space
u_k	the unit center or mean of the Gaussian
σ_k	the spread of the neighbourhood or width of the Gaussian
V_j	subspace at scale j

W_j	orthogonal complement of subspace at scale j
$W(x)$	weight function (kernel)
$WT(a, b)$	continuous wavelet transform
$\mathbf{x}$	neural network input vector
$x[n]$	discrete-time signal
y	neural network output vector
$\|\bullet\|_2$	Eucliean norm
σ	scaling parameter for the single-sigma PNN
$\sigma_1, \ldots, \sigma_p$	scaling parameters for the multi-sigma PNN
ε_n	threshold for the distance between an input and a stored pattern
ε_{max}	the largest scale interested in RAN
α_k	contribution of each hidden unit to a particular output
$\nabla_W f(X_n)$	gradient of the function f with respect to W evaluated at $W^{(n-1)}$
η	adaptation step size of RAN
γ	decay constant used in RAN
δ	threshold used for removing hidden units in RAN
$\Phi(t)$	scaling function
$\Psi(t)$	mother wavelet or primary wavelet function

CHAPTER I
INTRODUCTION

1.1 Problem Definition and Motivation

A power system transient is defined as a change in the *root mean square* (RMS) voltage or current at the power frequency for a short duration (0.5 cycles to 1 minute). These variations include power interruptions, sags, and swells. These transients may be caused by lightning strikes, different faults, switching of heavy loads, starting of large motors, etc. Different transients may have different effects on the system and electrical devices. Some may cause damage on devices, some may trigger mis-operation of devices, some may even cause system wide failure. To be able to take correct actions based on different types of transients, especially fast action to different faults, the ability to record and classify transients in real time is crucial to maintain high reliability of the power system.

With a broader application of highly nonlinear electronic-controlled devices in power apparatus and systems, the amount of waveform distortion has been found to be more significant nowadays. Such distortion brings the power quality as an increasing concern to utilities and industries. On the other hand, the deregulation and the more open market of power industry are forcing the power supplier to provide even higher quality electricity. As a result, the fault management and diagnostic system, the power quality monitoring system have become more and more important for electricity suppliers to differentiate themselves from others in the more and more competitive power supply market.

Digital transient recorders have been used to capture and record interesting events occurring, such as transient faults, permanent failures, and some important operations, like capacitor bank switching. These transient recorders have been used for real-time system monitoring and post-fault analysis. But there is no automatic selection and classification mechanism built in the transient recorders currently being used, so it is a very laborious to find and select the most important events from the massive recordings.

Digital transient recorders have also been deployed as power quality monitoring tools to give a quantitative measurement on the quality of electricity they supply to customers. But without an automatic selection, classification and measurement mechanism, it would be very hard to give an accurate quantitative measurement on power quality.

In order to achieve the automatic selection and classification of different types of power system transients, and make the transient recorders more intelligent, an efficient transient analysis and modelling tool is required.

Wavelets have been of great interests to many researchers active in diverse areas, such as in image processing, image compression, and denoising [KiLa93][DaKi98]. Recently, wavelets have been investigated to find new applications in power systems. One of the possible major application domains is the power system disturbance analysis, including long-term disturbance and short-term disturbance (transients). Within this interesting area, monitoring and classification of the short-term disturbance, transients, has become a focal point for many researchers [MoKi97].

One of the major focal points of this thesis is to investigate the new modelling method of

wavelets in power system transients, and neural network classification of transients. The objective to develop more intelligent transient recorders is the direct driving force behind this thesis.

Another interesting topic covered in the thesis is on short-term power load prediction. Comparing to the transient analysis and modelling, the short-term power load prediction is actually the modelling of long-term nonstationary signals. The commonalities between these two topics are nonstationary signal modelling and analysis. By studying the modelling and analysis of both short-term and long-term nonstationary signals with wavelets and neural networks, this thesis is trying to provide a deeper understanding of how to model nonstationary signals in general. This has become another driving force behind the thesis.

1.1.1 The Modelling of Transients

Wavelet Transform (WT) theory provides a unified framework for signal processing applications. In contrast to the Fourier transform based approaches where a window is used uniformly for spreaded frequencies, the WT uses short windows at high frequencies and long windows at low frequencies. The characteristics of nonstationary disturbances can be monitored more accurately. With the compact representation capability of wavelet transform, and time-scale localization, wavelet transform can be studied further for applications in power system transient recording, storing, and automatic classification area.

In the past, Huang and Hsieh [HuHs99] presented their experimental results of using Morlet wavelets to supervise power system transients. In [MoKi97], Daub4 wavelets are used for power system transients modelling. In [ChBr96], wavelets are studied as a new tool for the resonant grounded power distribution system relaying. In [PaSa98], wavelets are used as a new tool for fault detection in power transformers during impulse tests.

Since the WT is well suited to transient signal modelling, it has become a natural choice in this thesis to be used as a feature extraction tool to extract time-frequency features of power transients from the wavelet time-scale domain.

1.1.2 Power System Transient Classification

With the advances of artificial intelligence and computing hardware, artificial neural networks have attracted great attention from many researchers in power systems. One of the major applications of neural networks is in the classification of power system faults and other types of transients. Different architectures of neural networks have been investigated in this application area [KaSo92][LiYa01][CoJo98][HuYa01]. This thesis proposed a transients classification framework based on a neural network, *probabilistic neural networks* (PNN) [Spec88][Spec90][MoKi98]. Within this framework, the *discrete wavelet transform* (DWT) is used in the signal preprocessing stage for feature extraction, and the extracted features are used as an input for the PNN. The PNN acts as the classifier for different input transient patterns. The performance of the PNN has been demonstrated in the thesis in Chapter 4. This framework can be used not only for distinguishing different faults, but also for classifying different power quality problems, and provide concrete measurement of power quality.

1.1.3 Short-Term Power Load Prediction

Another interesting application of artificial neural networks in power systems is short-term power load prediction. Short term power load prediction not only plays a central role in potentially costly operation scheduling, like generation dispatch, capacity commitment, fuel purchasing and energy transactions, but also has significant impact on the security of power systems. Deregulation and competition of the power industry are now propelling the utilities to operate the system

at an even higher efficiency level. This trend further intensifies the concern as to the accuracy of short term power load prediction.

Nonstationary and uncertain characteristics exist in the loads of power systems. Such characteristics lead to difficulties in constructing an adequate model to predict the load variations precisely. Different methods have been developed to improve the accuracy of the forecasting, such as statistical regression and stochastic time-series models, and more recently, artificial neural networks [SiMo00][VeBo98][DaLi97]. The most frequently used neural networks are the fixed-structure neural networks trained by different training algorithms, either supervised training algorithms, like the *backpropagation* (BP), or unsupervised training algorithms, like the Kohonen *self-organizing feature map* (SOFM) [Koho90]. These algorithms first assume a fixed structure of the neural networks. Then the weights and bias of the neural networks are calculated by a certain amount of training samples. Several problems exist in such schemes. First, the ratio between the training samples and the size of the parameters which is determined by the structure of the neural networks is hard to determine. Secondly, the assumption of the parameters of the structure, like the number of neurons, the connections in the networks are usually made under certain environment. When the environment changes, all these assumptions cannot be modified because the structure of the neural networks is fixed. As a result, such kind of model lacks the evolving capability which is indeed required by a good model to be able to capture the evolving nature of a dynamic system.

A new type of neural networks, the *resource allocating networks* (RAN), which was first proposed by Plat [Plat91], is introduced in the thesis to be applied in short term power load forecasting [MoKi98]. The RAN is not only capable to evolve itself by adjusting the weights by track-

ing the changes of the input, but also able to adjust its structure by adding new neurons if it finds the underlying system being modelled is becoming more complex, or deleting neurons when it finds some of its neurons no longer make contributions in the modelling process [YiSu97]. With this evolving capability of the RAN, the complexity of the underlying system is tracked and then optimized matching could be achieved. This also solves the overfitting or overparameterization problem usually encountered by other types of neural networks.

1.2 Thesis Statement and Objectives

The purpose of this thesis is to investigate new transient modelling techniques based on time-frequency domain analysis and new structurally evolvable neural networks and the applications based on these new transient modelling techniques.

One of the objective is to investigate time-frequency domain analysis and modelling of power transients and develop an intelligent transient recorder that can classify different types of power system transients based on this modelling technique and an efficient neural network classifier. In order to achieve this objective, the following sub-objectives are required:

i) Time-frequency domain modelling of power transients; and

ii) Efficient neural network classifier for transient classification.

Another objective of this thesis is to investigate a new modelling techniques using structurally evolvable neural networks, and develop a more accurate predicator for short-term power load prediction.

1.3 Thesis Organization

This thesis is organized into six chapters around two focal points, power system transients

modelling and classification, and short-term power load prediction.

Chapter 2 provides a general introduction to artificial neural networks and wavelets, and discussed some important concepts and characteristics about neural networks and wavelets.

Chapter 3 presents the study and experimental results of wavelet modelling of power systems. By using wavelet transform, power system transient signals are represented in the time-scale space. This type of representation provides more accurate measurement for nonstationary signals with transient behaviour than the traditional Fourier transform. Based on the wavelet transform, a framework which can be used for power system transient classification is proposed.

In Chapter 4, a new type of neural network, the PNN, is presented and discussed for transient classification. Because of its fast training time and capability to approximate the Bayes optimal decision surface, the PNN is selected as the classifier in the proposed transient classification system. Experimental results are also presented in this chapter to demonstrate the advantage of the PNN as a classifier.

In Chapter 5, a new nonstationary signal modelling technique based on structurally evolvable neural networks, RAN, and its application in short-term power load prediction is discussed. With the capability of evolving itself not only by adjusting the values of its parameters, but also by changing its computing architecture, and therefore matching the computing complexity with the complexity of the underlying dynamic system being modelled, RAN is a suitable system modelling tool for modelling dynamic systems, and therefore for predicting nonstationary signals generated by dynamic systems. Experimental results presented in Chapter 5 show RAN as a promising tool for short-term power load prediction. Finally, conclusions, contributions and recommendations are presented in Chapter 6.

CHAPTER II
BACKGROUND

The basic methods used in this thesis are *artificial neural networks* (ANNs) and wavelets. So, in this chapter, some background on ANNs and wavelets are presented first. Based on the background knowledge presented in this chapter, the applications of ANNs and wavelets in power systems, specifically, transient signal analysis and classification, and short term load prediction will be presented in the following chapters.

2.1 Artificial Neural Networks

2.1.1 What are ANNs?

Work on ANNs has been motivated by the fact that our brain "computes" in an entirely different way from the conventional digital computer. Typically, neurons are five to six orders of magnitude slower than silicon logic gate. Events in a silicon chip happen in the nanosecond (10^{-9} seconds) range, whereas neural events happen in the millisecond (10^{-3} seconds) range. However, the brain makes up for the relatively slow rate of operation of a neuron by having a staggering number of neurons (nerve cells) with massive interconnections between them. The net result is that the brain is a highly efficient structure with 10^{-16} Joules (J) per operation per second, whereas the corresponding value for the best computers of today is about 10^{-6} J per operation per second [Hayk94].

A neural network is a massive parallel distributed processor that has a natural propensity for storing experimental knowledge and making it available for use. It resembles the brain in two respects:

a) Knowledge is acquired by the network through a learning process; and

b) Interneuron connection strengths (known as synaptic weights) are used to store the knowledge.

2.1.2 Capabilities of ANNs

Some of the major benefits provided by the ANNs are:

1. *Nonlinearity.* A neuron is a nonlinear device. Also, the network built by massive number neurons are nonlinear, which is distributed throughout the network. Because of this important property, the modelling of many physical processes or systems become possible. As we know, many physical process or system, e.g., the generation system of speech signal, the power load requirement, are inherently nonlinear.

2. *Input-Output mapping.* ANNs learn the input-output mappings during the learning process, and because of the nonlinearity of the underlying physical system, these mappings are nonlinear. This capability of ANNs makes them suitable for pattern classifier. In this thesis, an ANN-based power system transient analysis and classification system will be presented.

3. *Adaptivity.* Neural networks have a built-in capability to adapt their synaptic weights to changes in the surrounding environment. In particularly, a neural network trained to operate in a specific environment can be easily retrained to deal with minor changes in the operating environmental conditions.

4. *Contextural Information*. Knowledge is represented by the structure and activation state of a neural network. Every neuron in the network is potentially affected by the global activity of all other neurons in the network. Consequently, contextural information is dealt with naturally.

2.1.3 Models of a Neuron

A neuron is an information-processing unit that is fundamental to the operation of a neural network. Figure 2.1 shows the model for a neuron.

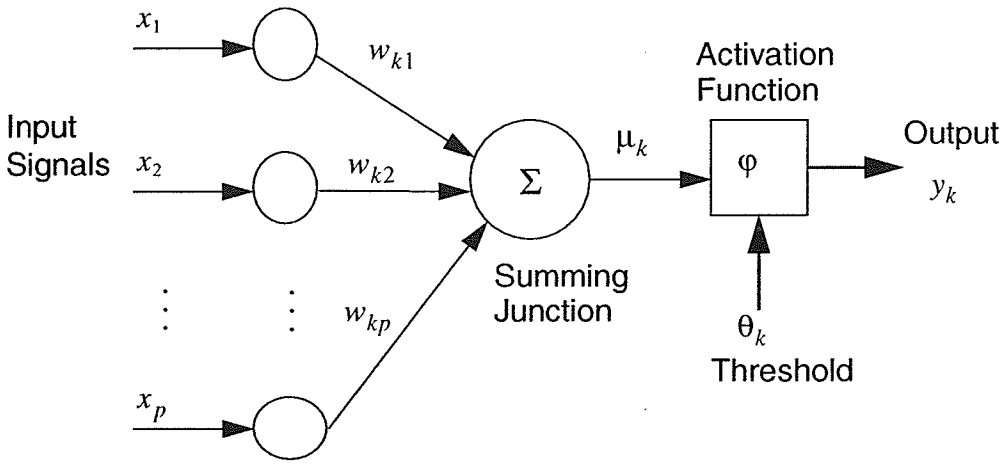

Fig. 2.1 Nonlinear model of a neuron.

There are three basic elements of the neuron model:

- A set of synapses or connecting links, each of which is characterized by a weight;
- An adder for summing the input signals, weighted by the respective synapses of the neuron;
- An activation function for limiting the amplitude of the output of a neuron. Typically, the normalized amplitude range of the output of a neuron is [0,1] or [-1, -1].

The model of a neuron shown in Fig. 2.1 also includes an externally applied threshold θ_k

that has the effect of lowering the net input of the activation function, or a bias that increase the net input.

A neuron k can be described mathematically by writing the following pair equations

$$u_k = \sum_{j=1}^{p} w_{kj} x_j \tag{2.1}$$

and

$$y_k = \varphi(u_k - \theta_k) \tag{2.2}$$

where $x_1, x_2, \ldots, x_p$ are the input signals; $w_{k1}, w_{k2}, \ldots, w_{kp}$ are the synaptic weights of neuron k; u_k is the linear combiner output; θ_k is the threshold; φ is the activation function; and y_k is the output signal of the neuron.

2.1.4 ANN Architectures

There are different architectures of ANNs, but the most often used structure of ANNs is multilayer feedforward networks. Fig. 2.2 shows the structure.

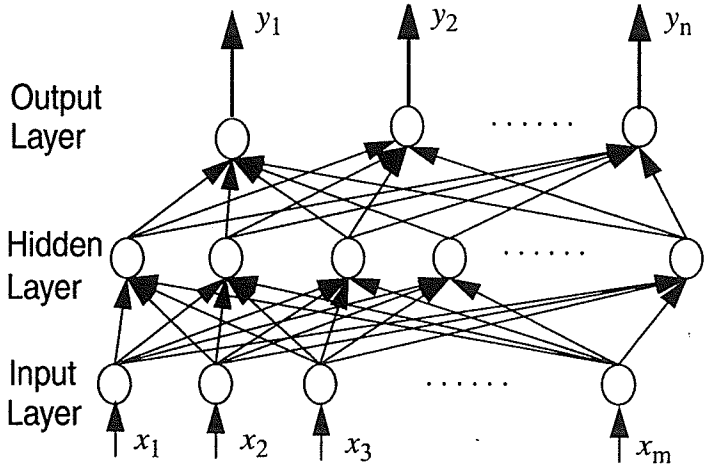

Fig. 2.2 Feedforward network with one hidden layer.

The source nodes in the input layer of the network supply respective elements of the acti-

vation pattern (input vector), which constitute the input signals applied to the neurons (computation nodes) in the second layer (i.e., the first hidden layer). The output signals of the second layer are used as inputs to the third layer, and so on for the rest of the network. Typically, the neurons in each layer of the network have as their inputs the output signals of the preceding layer only. The set of the output signals of the neurons in the output (final) layer of the network constitutes the overall response of the network to the activation pattern supplied by the source nodes in the input (first) layer.

The function of the hidden neurons is to intervene between the external input and the network output. By adding one or more hidden layers, the network is enabled to extract higher-order statistics. The ability of hidden neurons to extract higher-order statistics is particularly valuable when the size of the input layer is large.

2.1.5 Learning Tasks

ANNs can be used for several learning tasks. Since this thesis will apply ANNs in power system transients analysis and classification and power load prediction, which belongs to pattern classification and prediction learning tasks respectively, some basic description of this two learning tasks are first described shortly in the following paragraphs.

1. *Pattern Classification.* In this learning task there is a fixed number of categories (classes) into which activations are to be classified. To resolve it, the neural network first undergoes a training session, during which the network is repeatedly presented a set of input patterns and the category to which each pattern belongs. After the training session, the neural network is presented a new pattern which belongs to the same population of patterns used to train the network. Now the task of the network is to classify this new pattern correctly. The advantage of using

a neural network to perform pattern classification is that it can construct nonlinear decision boundary between the different classes in a nonparameteric fashion, and therefore is a practical method for solving complex pattern classification problems.

2. *Prediction*. The issue of predicting is a basic learning task. It is a temporal signal processing problem in that we are given a set of M past samples $x(n-1), x(n-2), \ldots, x(n-M)$ that are usually uniformly spaced in time, and the requirement is to predict the present sample $x(n)$. Prediction may be viewed as a form of model building in the sense that the smaller the prediction error in a statistical sense, the better will the network serve as a physical model of the underlying stochastic process responsible for the generation of the time series. When the process is nonlinear, the use of ANNs provides a powerful method for solving the prediction problem by virtual of the nonlinear processing units built into its structure. The only exception to the use of the nonlinear processing units is the output unit of the network, which is a linear operator. This makes it possible for the dynamic range of the predictor output to match that of the predictor input.

2.2 Wavelets and Multiresolution Signal Decomposition

As a great breakthrough of mathematics and applied mathematics, wavelets have been widely used in signal processing, such as image compression, speech analysis and recognition, electrocardiograms analysis, and radar signal processing. As a tool to detect the irregular structure and transient phenomena in a signal, wavelets have demonstrated the superiority in transient signal analysis comparing to the traditional signal analysis tools, the Fourier transform. The Fourier transform is global and provides a description of regularity of signals, but it is not well adapted for finding the location and the spatial distribution of transient phenomena. Although the *short-time*

Fourier transform (STFT) can partly alleviate the problem, the STFT still has the limitation of a fixed window width, which means the trade-off between the frequency resolution and the time resolution should be determined *a prior* to observe a particular characteristic of the signals. The limitation of a fixed window width in fact is inadequate for the analysis of the transient nonstationary signals. Wavelets provides a way to decompose signals into building blocks that are well localized both in time and frequency. The window width in wavelet transform is automatically adjusted to give proper resolutions of both the time and the frequency. In this approach, a larger resolution of time is provided to high-frequency components of a signal, and a larger resolution of frequency to low-frequency components. These features make the wavelet transform well suited for the analysis of the power system transients caused by various disturbances.

2.2.1 Multiresolution Signal Decomposition (MSD)

The goal of multiresolution analysis is to develop representation of a signal $f(t)$ at various levels of resolution. To achieve this goal, the signal can be expanded in terms of its orthogonal basis $\phi(t)$, which can be scaled to give multiple resolutions of the original signal. The signal can be approximated at any scale j as [Mall89]

$$f(t) = \sum_{n} c_j(n) 2^{j/2} \phi(2^j t - n) \tag{2.3}$$

where, c_j is the j level scaling coefficient.

However, the important features of the signal can better be described by defining a slightly different set of functions, wavelets $\psi_{j,n}(t)$, which span the difference between the spaces spanned by the various scales of the scaling function. Therefore, $f(t)$ can be represented in terms

of both the scaling and wavelet functions as

$$f(t) = \sum_{n} c_0(n)\phi(t-n) + \sum_{n}\sum_{j=0}^{J-1} d_j(n)2^{j/2}\psi(2^j t - n) \tag{2.4}$$

where, d_j is the j level wavelet coefficient.

$$d_j(k) = \langle f(t), \psi_{j,k}(t) \rangle \tag{2.5}$$

This gives us the multiresolution signal decomposition of the original signal $f(t) \in V$; and spans it in J spaces.

$$V_j = V_{j-1} \oplus W_{j-1} \tag{2.6}$$

where the subspace W_{j-1} is the orthogonal complement of the subspace V_{j-1} and it represents the needed detail to move to finer subspace. The representation of the subspace V_j in terms of detailed subspaces (W_0 to W_{j-1}) can be written as:

$$V_j = V_0 \oplus W_0 \oplus W_1 \oplus W_2 \oplus \ldots \oplus W_{J-1} \tag{2.7}$$

So using the multiresolution decomposition approach, the time domain signal $f(t)$ can be mapped into the wavelet domain and represented at different resolution levels in terms of the following expansion coefficients C_{Signal}, where

$$C_{Signal} = c_0 + d_0 + d_1 + \ldots + d_{J-1} \tag{2.8}$$

where d_j present the detail coefficients at different resolution levels, and c_0 presents the last approximate coefficients.

To get the approximate coefficients c_j and the detail coefficients d_j at scale j for an input signal $f(t)$, the following steps are necessary (as shown in Fig. 2.3):

1. Convolve the expansion coefficients c_{j+1} at scale $(j+1)$ by the selected wavelet function coefficients, $h_1(n)$, and the scaling function coefficients $h_0(n)$; and
2. Obtain the approximate coefficients c_j and the detail coefficients d_j at scale j by down-sampling.

2.2.2 Comparison of Wavelet and Fourier Transform

The inner product defined in (2.9) is a means of comparison of two functions. The more the functions $g(t)$ and $x(t)$ are similar, the higher the integral of their product (i.e. their inner product), which is

$$\langle x, g \rangle = \int_{-\infty}^{\infty} x(t) g^*(t) dt \tag{2.9}$$

($g^*(t)$ denotes the complex conjugate of $g(t)$). This is shown in Fig. 2.3.

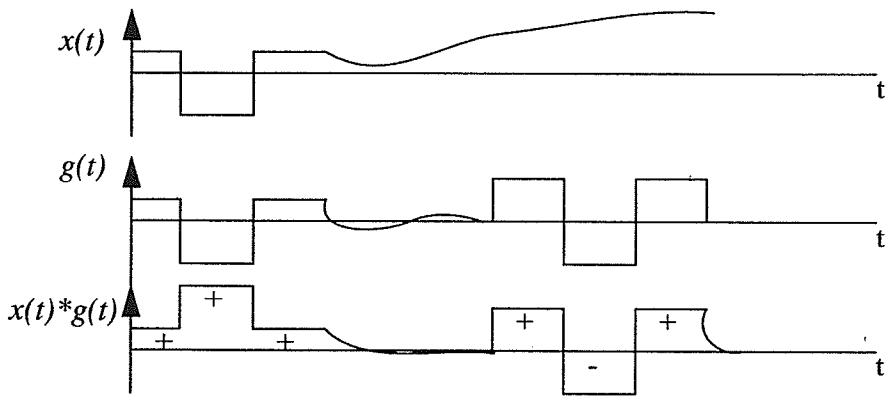

Fig. 2.3 Comparison of *x(t)* with *g(t)*.

As shown in Fig. 2.3, the two functions are more similar at the beginning part, so the inner product has a higher value (because each part is positive, the sum is higher). At the end part two functions are less similar, so the inner product has lower value (each part of the inner product has different sign).

The Fourier transform (FT) of a signal $x(t)$ at frequency f in (2.10), is the comparison between $x(t)$ and the time infinite complex exponential of frequency f

$$\hat{X}(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft}dt \tag{2.10}$$

Furthermore, the FT assumes that the analyzed signals are infinitely time repetitive, and cannot take the change in the signal spectrum into account, particularly in transient events. The global function e^{jwt} transform the signal from the time-domain into the frequency-domain. Consequently, all the information in the time domain, which can help in detecting the starting time t_1 of a transient disturbance and the duration of the disturbance, is lost.

In order to achieve a high degree of localization, both in time and frequency, a window function with a sufficiently narrow time is required, as is done with the windowing FT or short-time FT (STFT). This is done by selecting only a desired portion of the signal (achieved by multiplying the original signal by another function with zero magnitude outside the desired interval). The FT of this portion of the signal then provides the frequency information. The STFT of a signal $x(t)$ with window function $g(t)$ is

$$STFTX_g(t, f) = \int_{-\infty}^{\infty} x(u)g^*(u-t)e^{-j2\pi fu}du \tag{2.11}$$

According to (2.9) and (2.11), the STFT calculates similarities between a signal $x(t)$ and the time infinite complex exponential of frequency f through a window function $g(t)$.

Unlike the FT, the STFT takes into account time changes in signal spectrum. The STFT gives a more understandable representation of signal $x(t)$ because it shows a time-frequency related representation plane. The tiling of the time-frequency plane is fixed, given by Δt and Δf, time and frequency integral steps (See Fig. 2.4). Furthermore, $g(t)$ determines time and frequency tiling uncertainty (short duration window $g(t)$ generates low time uncertainty and major frequency uncertainty, and vice versa).

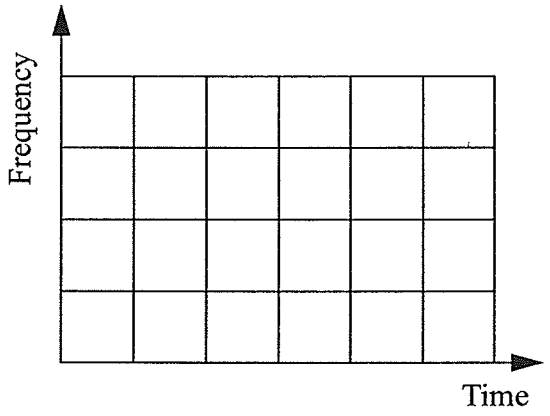

Fig. 2.4 STFT time-frequency plane.

However, the method is limited due to the constraint on the window size. Using a wide window results in good frequency resolution but poor time resolution, and using a narrow window results in poor frequency resolution but good time resolution. Therefore, this representation is not suitable for short duration/high frequency and long duration/low frequency signals because of the fixed duration of $g(t)$. Furthermore, Heisenberg's uncertainty principle imposes a theoretical lower bound on the area of the time-frequency window of any window function. This principle

indicates that the signal's feature (frequency component) and location's feature (position at which that frequency component is found) cannot both be measured to an arbitrary degree of precision simultaneously [Daub92].

All these constraints have led to more accurate signal processing tools using connected time and frequency representations. The wavelet transform (WT) is the tool which overcomes the limitations of the STFT and implements a flexible window function capable of operating over a wide frequency range. Using wavelet transform, a signal is mapped into 2D and decomposed at different resolution levels. The time-scale plane of WT is shown in Fig. 2.5. As shown in Fig. 2.5, the duration of windowing function changes according to the frequency analyzed. The higher frequency is analyzed, the shorter duration of the window function is used; the lower frequency is analyzed, the longer duration of the window function is used. In all the cases, the Heisenberg-Gabor principle of uncertainty is always satisfied.

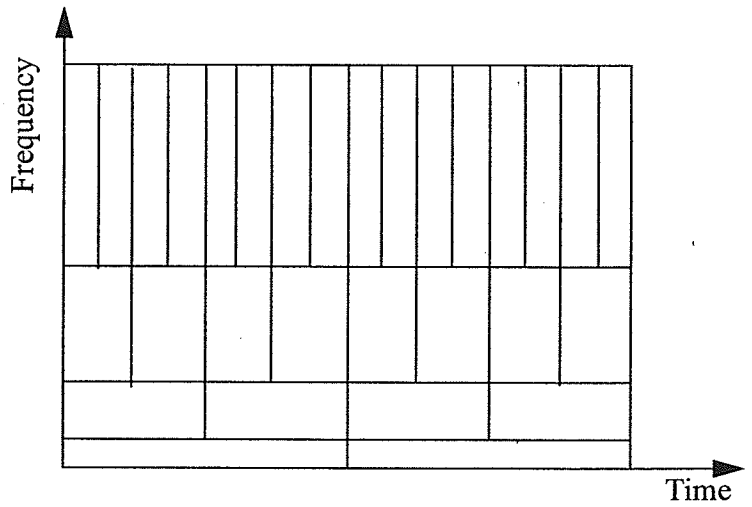

Fig. 2.5 WT time-scale plane.

Due to the more flexible and more accurate representation of signals, WT is effective in monitoring the signal dynamics as time varies. For those time intervals where the signal changes rapidly, the WT method can zoom on the area of interest for a better extraction of signal characteristics. In other words, the WT method is much more local. Instead of transforming a pure "time description" into a pure "frequency description", the WT finds a good promise by providing a time-frequency description.

FT has shown great success in modelling harmonics and has been applied widely in power system analysis and control. However, with the increasing use of nonlinear equipment in power systems, like arc furnace, especially in a AC-DC combined system, power systems have shown more nonstationary, nonlinear characteristics. Such nonstationary transient signals requires a more efficient modelling and analysis tools. Based on the discussion of the previous two sections, WT appears to have more potential for use in power system transient signal analysis. In the next section, one of the mathematical attributes of WT is explained in more detail to show why WT is more suitable to analyze transient signals, and therefore why it is a better tool to model power system transient disturbances.

2.3 Wavelet Transform and Power System Transient Signal (Disturbance) Analysis

The WT is useful in detecting and extracting disturbance features of various types of power system transient signals because it is sensitive to signal irregularities but insensitive to the regular-like signal behaviour. In most power system transient signals, the waveshape of a transient event is irregular compared to that of its 60 Hz background signal. As a consequence, the WT coefficients associated with the transient event have very large magnitudes compared to those of a transient-free waveform. The reason for this is related to the vanishing moment of the analyzing

wavelet $\psi(t)$ [Daub92]

$$\int_{-\infty}^{\infty} t^n \psi(t) dt = 0 \tag{2.12}$$

where $n = 0, 1, 2, \ldots, N-1$. The above expression is called a vanishing moment of order N. The order of the vanishing moment of an analyzing wavelet $\psi(t)$ is at least equal to one, i.e., $N = 1$.

To see why the WT is sensitive to the irregularities in a signal, we can use the Taylor series approximation on the wavelet transform to see the effect of the wavelet transform. The continous WT of $x(t))$ can be written as

$$WT\{x(t)\} = \int_{-\infty}^{\infty} x(t) \psi^*_{a,b}(t) dt \tag{2.13}$$

where the asterisk denotes a complex conjugation operator, and

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}} \psi\left(\frac{t-b}{a}\right) \tag{2.14}$$

Using the Taylor approximation, $x(t)$ can be expanded as the following,

$$x(t) = \sum_{m=0}^{\infty} \frac{x^{(m)}(t_0)}{m!} (t-t_0)^m \tag{2.15}$$

So the WT of (2.13) can be recast into the following,

$$WT\{x(t)\} = \int_{-\infty}^{\infty} \left(\sum_{m=0}^{\infty} \frac{x^{(m)}(t_0)}{m!} (t-t_0)^m \psi^*_{a,b}(t) \right) dt$$

$$= \int_{-\infty}^{\infty} x(t_0) \psi^*_{a,b}(t) dt + \int_{-\infty}^{\infty} \left(\sum_{m=1}^{\infty} \frac{x^{(m)}(t_0)}{m!} (t-t_0)^m \psi^*_{a,b}(t) \right) dt$$

$$= x(t_0)\int_{-\infty}^{\infty}\psi^*_{a,b}(t)dt + \int_{-\infty}^{\infty}\left(\sum_{m=1}^{\infty}\frac{x^{(m)}(t_0)}{m!}(t-t_0)^m\psi^*_{a,b}(t)\right)dt$$

$$= \int_{-\infty}^{\infty}\left(\sum_{m=1}^{\infty}\frac{x^{(m)}(t_0)}{m!}(t-t_0)^m\psi^*_{a,b}(t)\right)dt \qquad (2.16)$$

The property of vanishing moment given in (2.12) of analyzing wavelet function $\psi(t)$ is used in (2.16).

According to (2.16), we can see that the wavelet transform is sensitive to the derivation of signal $x(t)$. Generally speaking, the Taylor series coefficients form a monotonically decreasing sequence; therefore the largest term in the series is $x'(t_0)$. Equation (2.16) can now be approximated by

$$WT\{x(t)\} \approx \int_{-\infty}^{\infty} tx'(t_0)\psi^*_{a,b}(t)dt \qquad (2.17)$$

In other words, the WT reacts the most to the gradient of signal $x(t)$. Thus the wavelet transform is indeed analyzing the first derivative of $x(t)$ and not the amplitude of $x(t)$. This is why the wavelet transform is sensitive to signal irregularities but insensitive to the regularities of the 60 Hz background signal and this is why the wavelet transform is a ideal tool to analyzing power system transient signals.

From the discussion in Sec. 2.2.3, we have seen that the FT only gives a pure frequency representation, the dynamic changing prospect of nonstationary signals with irregularities distributed in different time domain is not captured by FT. The STFT tries to capture some part of the

time domain information of nonstationary signals, but due to the fixed tiling, it could not capture all the dynamics/irregularities demonstrated by the all kinds of nonstationary signals.

Due to the capability of WT to capture the dynamic characteristics of nonstationary transient signals, it has been a candidate tool to model and analyze power system transient signals. In the next chapter, the WT modelling of power system transients will be discussed in detail, and some of the experimental results will be presented.

CHAPTER III
WAVELET MODELLING OF POWER SYSTEM TRANSIENTS

This chapter presents a scheme for modelling transients in power systems using wavelets. Since transients are always associated with abnormal conditions in a power system, the analysis and modelling of transient are very important for explaining the causes of transients and provide evidences and supports for operators to take actions to correct the abnormal conditions.

In the past, the Fourier transform (FT) was applied to obtain the spectrum of transient signals. For example, in protection relaying, the fundamental frequency component is obtained through FT to provide fault information. The limitation of FT is that it is not well suited to analyze nonstationary signals because the FT is a pure frequency domain description of the signal and the frequency information revealed by the FT is an average over the entire time duration of the signal. For example, the FT of a transient fault superimposed on the fundamental wave will have no time information of the signal and the frequency information is not accurate.

This chapter gives a description of wavelet modelling of transient disturbances in power systems. The power of wavelet transform derives from its time-scale localization ability of non-stationary signals. As a result, wavelet transform (WT) provides an effective and efficient means of decomposing voltage and current signals of power system transients to detectable and discriminant features. Section 3.1 gives a short introduction of wavelet modelling of power system transient signals. Section 3.2 gives a full description of experimental results of wavelet modelling of power system transients. Based on Sec. 3.2, Sec. 3.3 introduces an application framework, power system transients classification system, which employs the wavelet modelling as a key subsystem.

3.1 Introduction

A power system transient is defined as a change in the root mean square (RMS) of voltage or current at the power frequency for duration from 0.5 cycles to 1 minute. These variations include power interruptions, sags, and swells. The voltage sag describes a drop of 10-90 percent of the system voltage, lasting for 0.5 cycle to 1 minute. They may be caused by the fault current, the switching of heavy loads or the starting of large motors. Voltage sag with 30% drop or more is considered severe. Contrary to voltage sag, the voltage swells are brief increase of the rated system voltage. They often appear on the unfaulted phases of a three-phase circuit where a single-phase short circuit occurs. Swell may stress the equipment components to premature failure. Power momentary interruption can be seen as a momentary loss of voltage on a power system. Such disturbance describes a drop of 90-100 percent of the rated system voltage lasting for 0.5 cycle to 1 minute.

Transients have become a common problem in recently years. They may cause misoperations to the utility customer's sensitive loads and producing effects ranging from blinking digital clocks in residential sector to disrupted industrial process. It has been documented in [SaPo00] that *programmable logic controllers* (PLC's) which are used to control devices such as dc and ac drivers may shut down the devices for voltage variation on the order of 80% to 85% of normal. The quality of a power supply has become more and more important issue for both utility companies and customers. It has been shown that the quality of a power supply is largely synonymous with the voltage quality. The general strategy to assure voltage quality and sound operation of a sensitive load is to monitor the supply voltage and take appropriate countermeasure(s) based on detection of voltage disturbances which exceed the load tolerance limits. Such countermeasures

are usually achieved by means of a power electronic apparatus, e.g. a Static Transfer Switch (STS), a Dynamic Voltage Restorer (DVR), or a VAR compensator. Some transient events that result in voltage disturbances may have no effect on the load, such as a voltage disturbance due to a close by capacitor-switching incident should not activate an STS. But some transient events may result in voltage disturbances that may have serious impact on the load, such as a voltage disturbance due to a remote fault. Thus in order to monitor and control power supply quality, the following features are desired:

a) Rapid detection of voltage disturbances; and

b) Identification of the types of events associated with the voltage disturbances.

Transient recorders have been developed to record such transient signals. However, using the information obtained from these recorders, it is often difficult to determine and classify different transients considering the huge amount of data recorded. For example, survey data [BiKr95] of a distribution system resulted in about 40 MB per day or about 15 GB per year for a modest-sized power system with 200 nodes. With the rapid advance of computer networks, fault recording practices have advanced from individual substation recorders to a network of remote units which transfer fault data records to a central computer for analysis. Considering the huge amount of information obtained from different networked substations, the data analysis process would be very labor intensive and requires considerable knowledge and skill. Manpower limitations may result in the failure to identify items needing correction before large financial losses occur.

In order to overcome these problems, the current used transient recorders need to be improved to add new features to support automatic transient detection and classification. The transient signal analysis and modelling is the key to achieve such objective.

The last ten years have seen an increasing interest in wavelets. The wavelet transform (WT) has become well known as a useful tool in several fields, including signal processing, image compression, biomedical applications, geophysics, speech processing, acoustics, numerical analysis [DaKi98] [Shap93] [LaKi93] [Dono95][Mall89].

More recently, a few studies have reported the feasibility of WT for power system transient analysis [MoKi97], [PaSa98], [HuHH99]. It has been shown that the wavelet analysis has the ability to characterize power system transients.

3.2 Wavelet Modelling of Power System Transients

3.2.1 Multiresolution Modelling of Power System Transients by DWT

Like the discrete Fourier transform (DFT), there is a discrete wavelet transform (DWT) for the sake of computer implementation. The DWT achieves a multiresolution decomposition of $x[n]$ by

$$x[n] = \sum_{j=1}^{J} \sum_{k \in Z} c_{j,k} \overline{h}_j[n - 2^j k] + \sum_{k \in Z} d_{j,k} \bar{g}[n - 2^j k] \tag{3.1}$$

where $\overline{h}_j[n-2^j k]$ and $\bar{g}[n-2^j k]$ correspond to the dual wavelets and behaves as low-pass filter and high-pass filter to the input signal, respectively. Fig. 3.1 illustrates the filter bank implementation scheme.

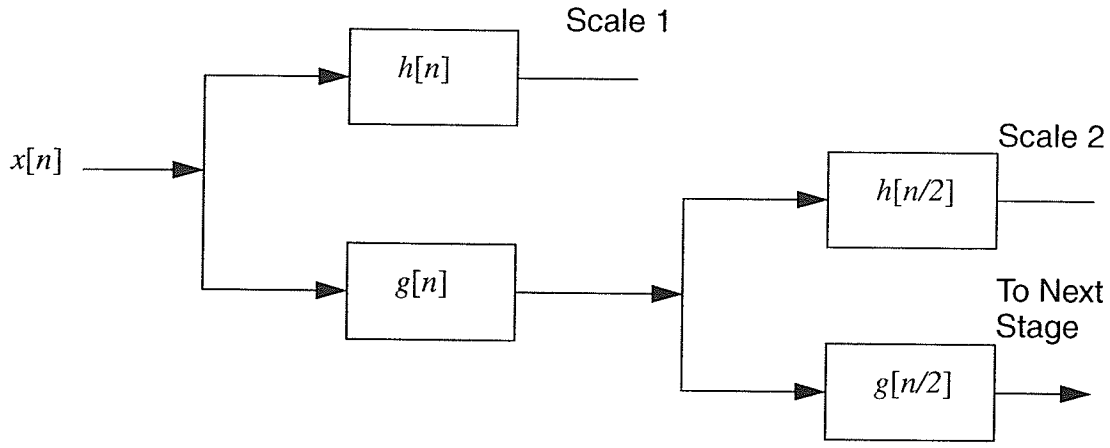

Fig. 3.1 Multi-stage filter bank DWT implementation.

3.2.2 Choice of Mother Wavelet

There are various types of wavelets that can be used in DWT, e.g. Haar, Daubechies. The choice of wavelets depends on the applications. For power system transients which has the characteristics of low amplitude, short duration, fast decaying and oscillatory, Daubechies' wavelet (Daub4), has shown the ability to detect and decompose such transient signals. Another advantage of using Daub4 is the fast calculation since only coefficients are required to perform the DWT.

3.2.3 Choice of Power System Transients to Perform DWT Modelling

In order to demonstrate the DWT modelling of power transients, two major categories of transients in power systems, single-phase fault and disturbances coming from switching operations, are simulated.

The transient simulation is performed using the software package PSCAD [PSCAD], developed by Manitoba HVDC Research Center, Winnipeg. The following simple transmission system is used to simulate the single-phase fault.

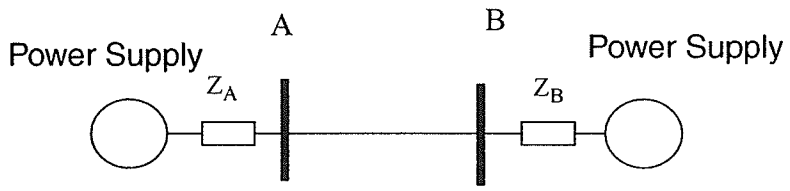

Fig. 3.2. Power system used for transient simulation.

The simulation is done on SUN-Sparc Ultra workstation.

3.2.4 Wavelet Analysis of Power Transients

The Discrete Wavelet Transform (DWT) of a simulated single-phase fault waveform using Daub4 is shown in Figs. 3.3 (a) to (e).

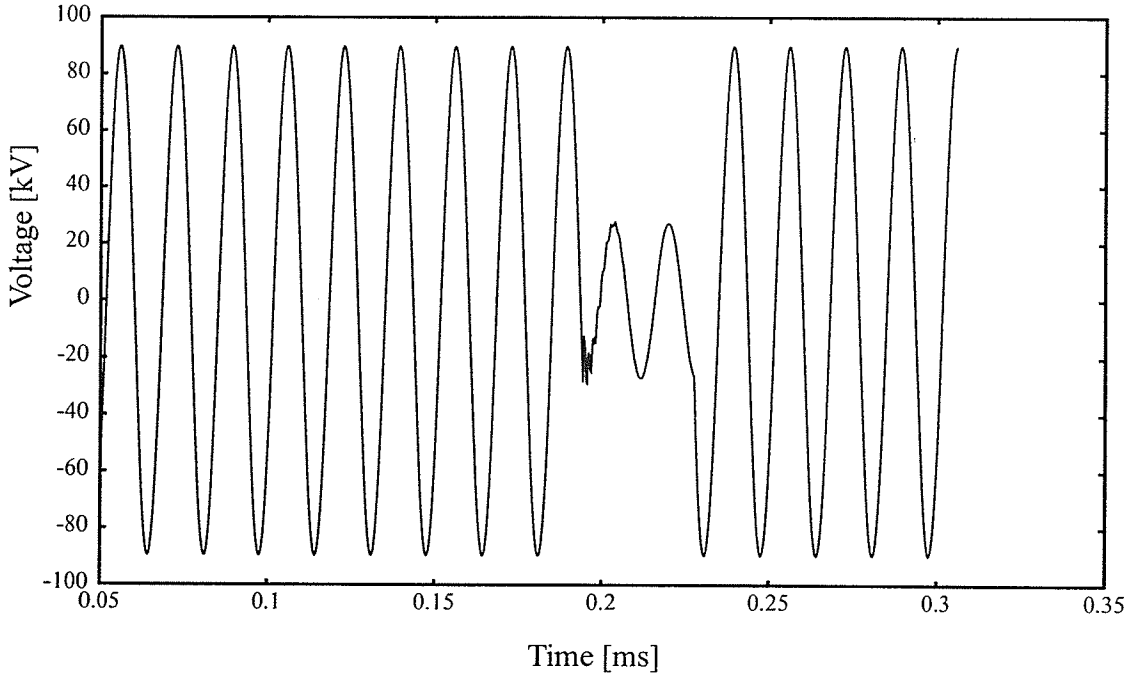

Fig. 3.3 (a) Single phase fault.

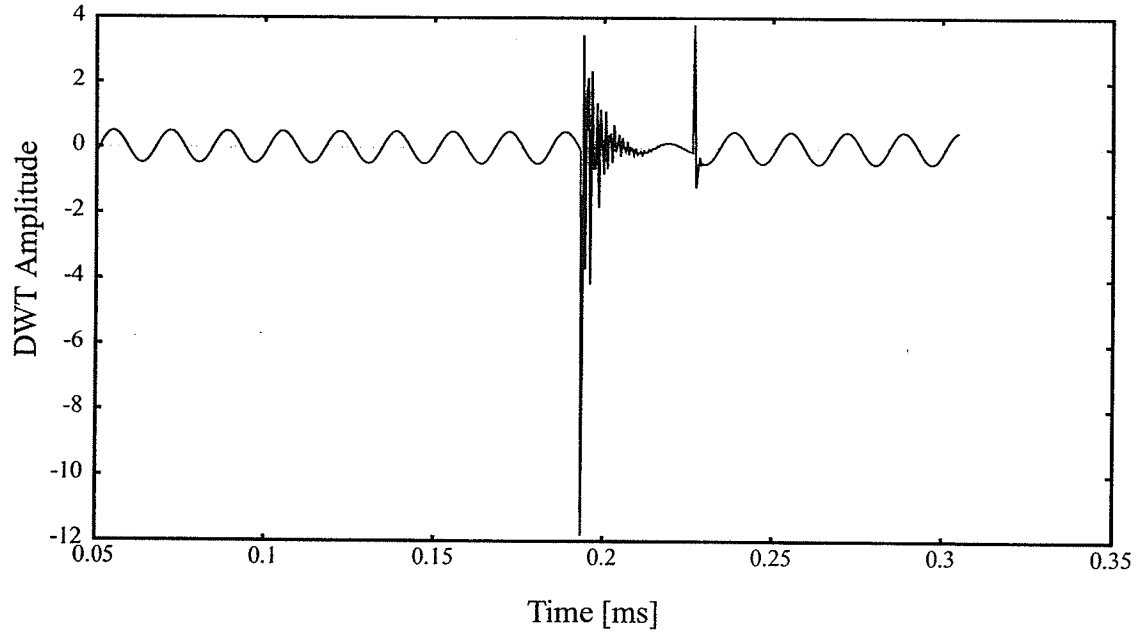

Fig. 3.3 (b) DWT of single phase fault at Scale 1.

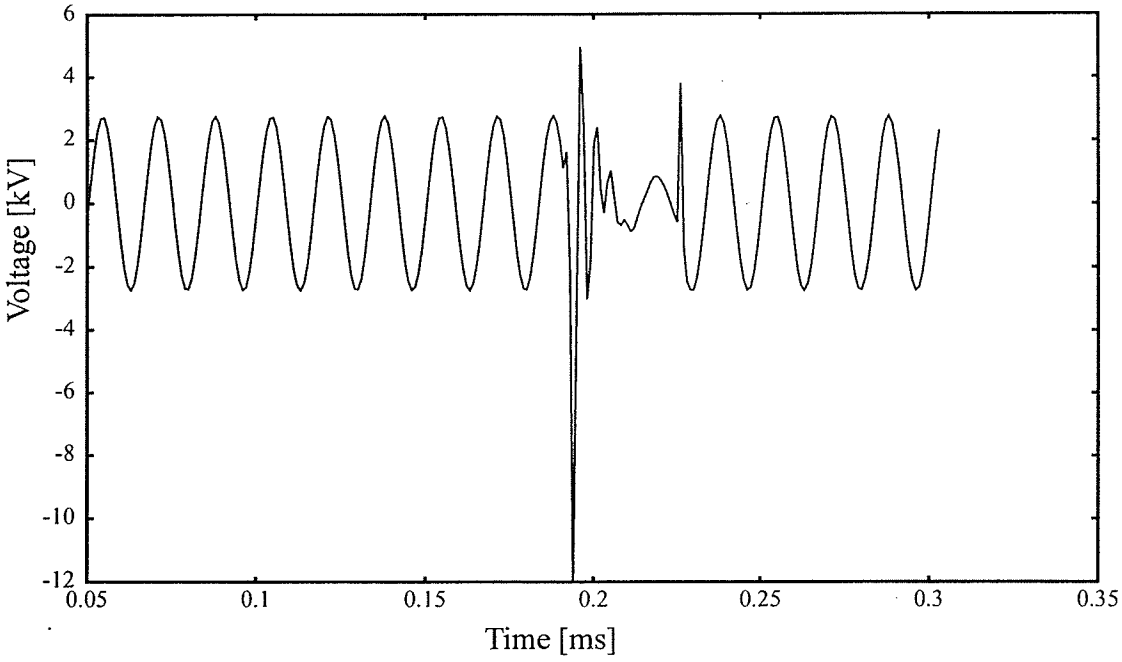

Fig. 3.3 (c) DWT of single phase fault at Scale 2.

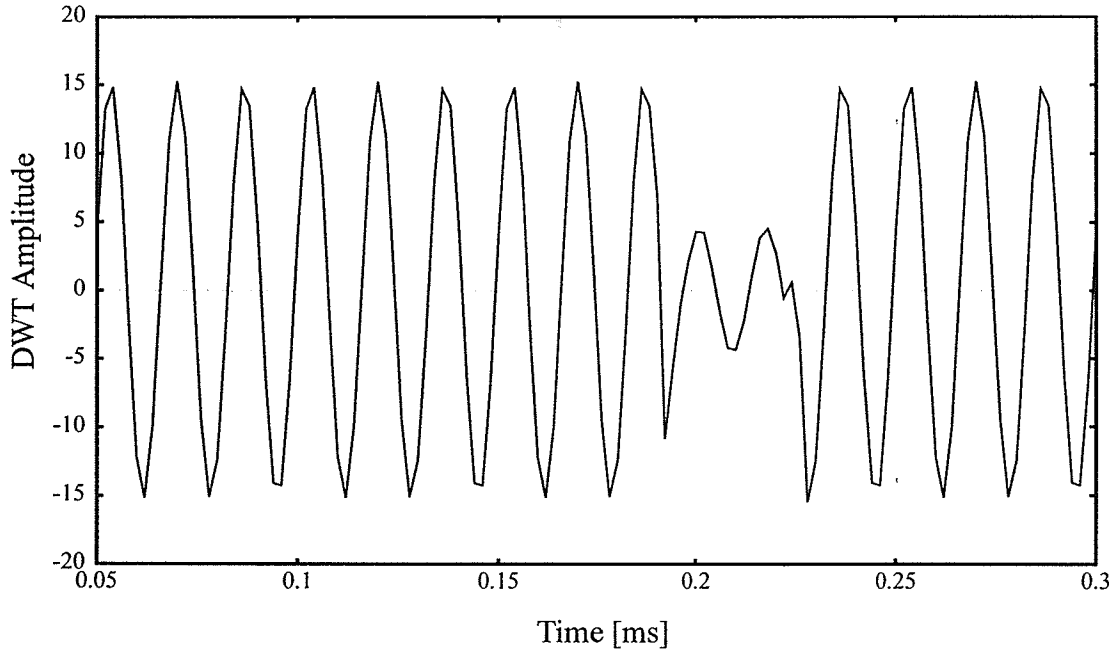

Fig. 3.3 (d) DWT of single phase fault at Scale 3.

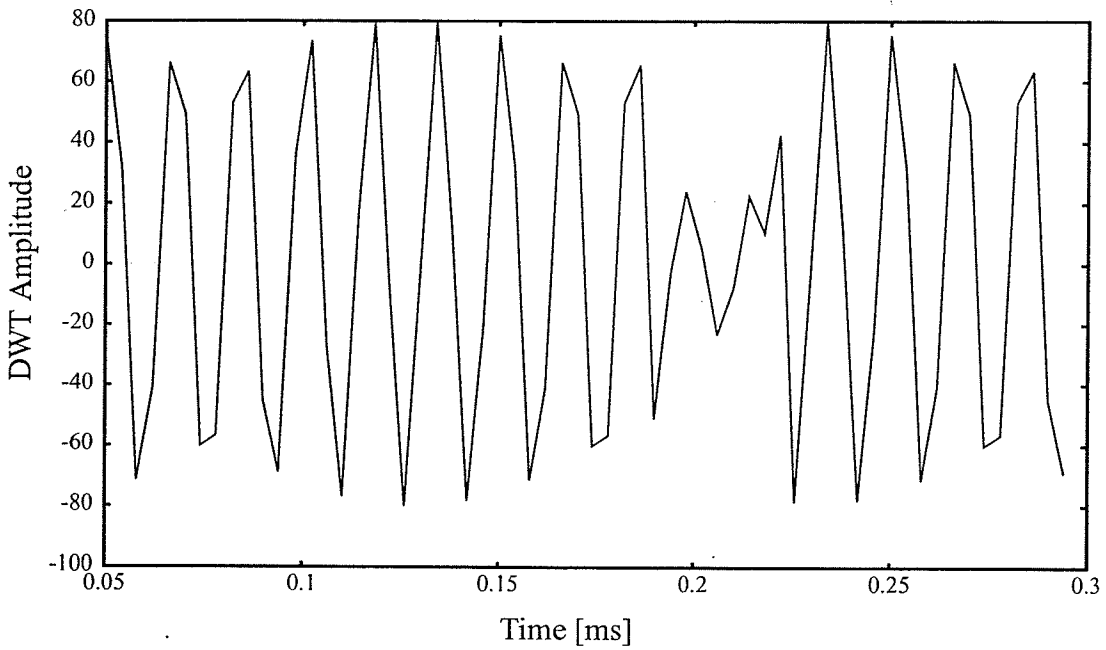

Fig. 3.3 (e) DWT of a single phase fault at Scale 4.

From Fig. 3.3 (b), the DWT of the single phase fault at scale 1, we can clearly see when the transient disturbance begins and ends. From Fig 3.3 (e), the DWT of the single phase fault at scale 4, we can see the voltage drop at the fundamental frequency, which indicates a voltage sag occurs. We can see that the accuracy of the location of the fundamental frequency given at scale 4 is not as high as at scale 1, which reflects the balance of the resolution between time and frequency achieved by wavelet transform decomposition into different scales.

As demonstrated by Fig. 3.3 (a) to (e), the type information of the transient disturbance can be obtained from the higher scale, whereas the timing information can be obtained accurately from the lower scale information. Therefore from both scales, along with the information from other scales, this type of transient disturbance can be classified correctly.

The wavelet transform decomposition of transients from capacitor banks switching operation are shown in Figs. 3.4 (a) to (d).

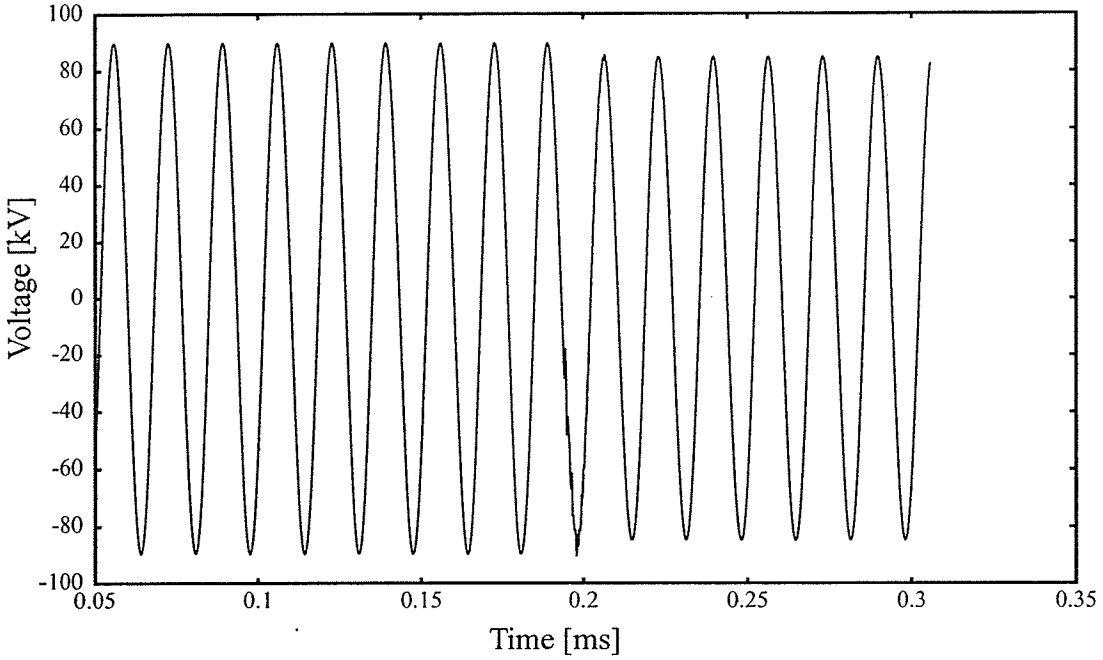

Fig. 3.4 (a) Capacitor bank switching transients.

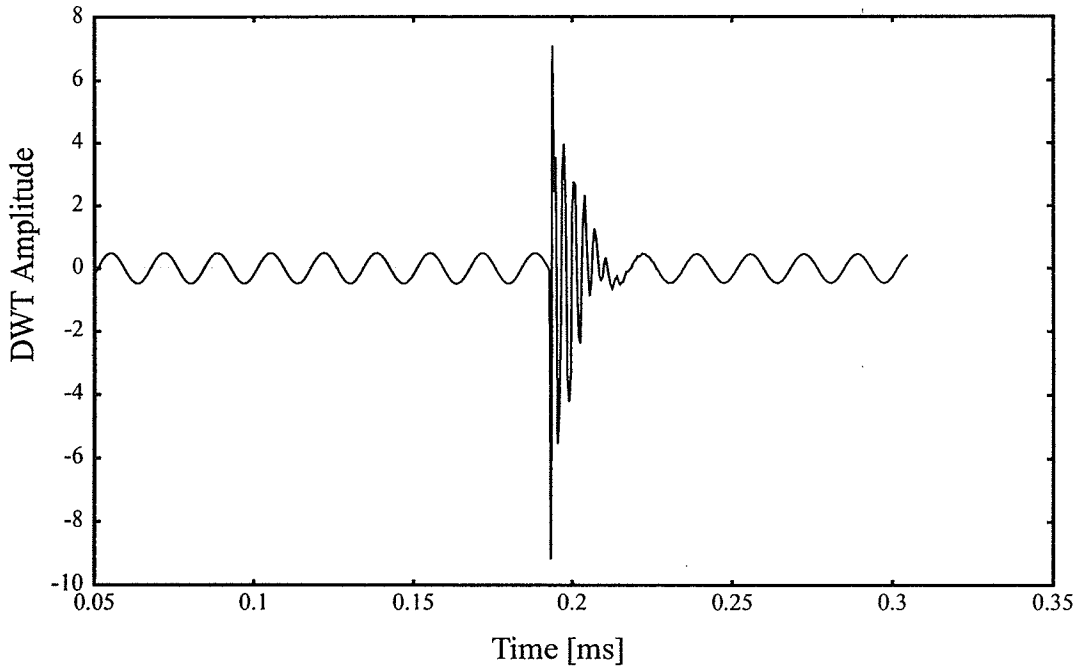

Fig. 3.4 (b) DWT of capacitor switching transient at Scale 1.

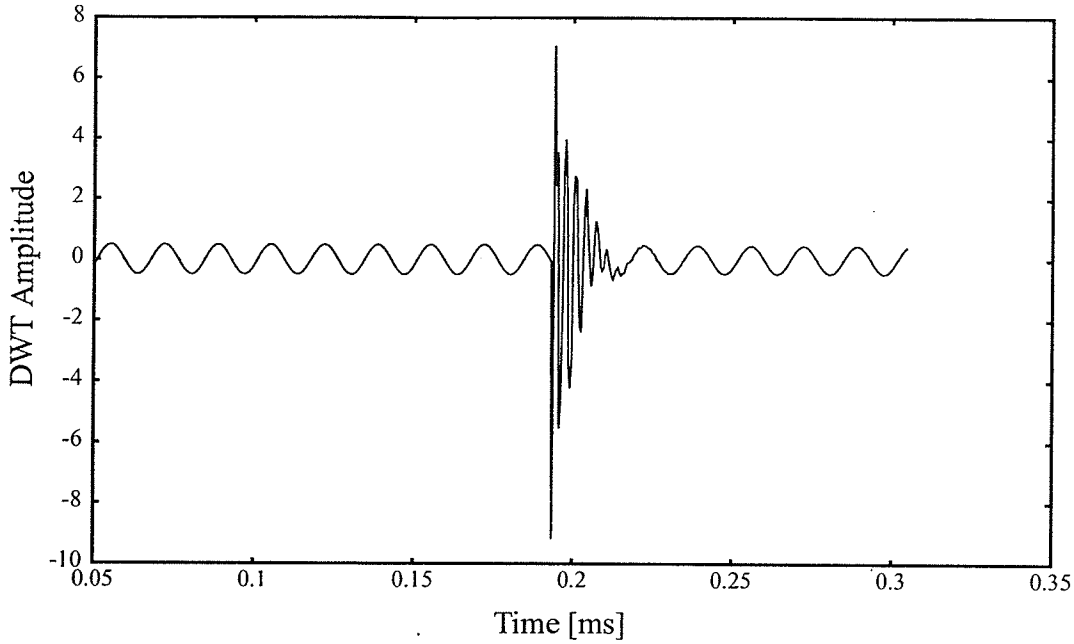

Fig. 3.4 (c) DWT of capacitor switching transient at Scale 2.

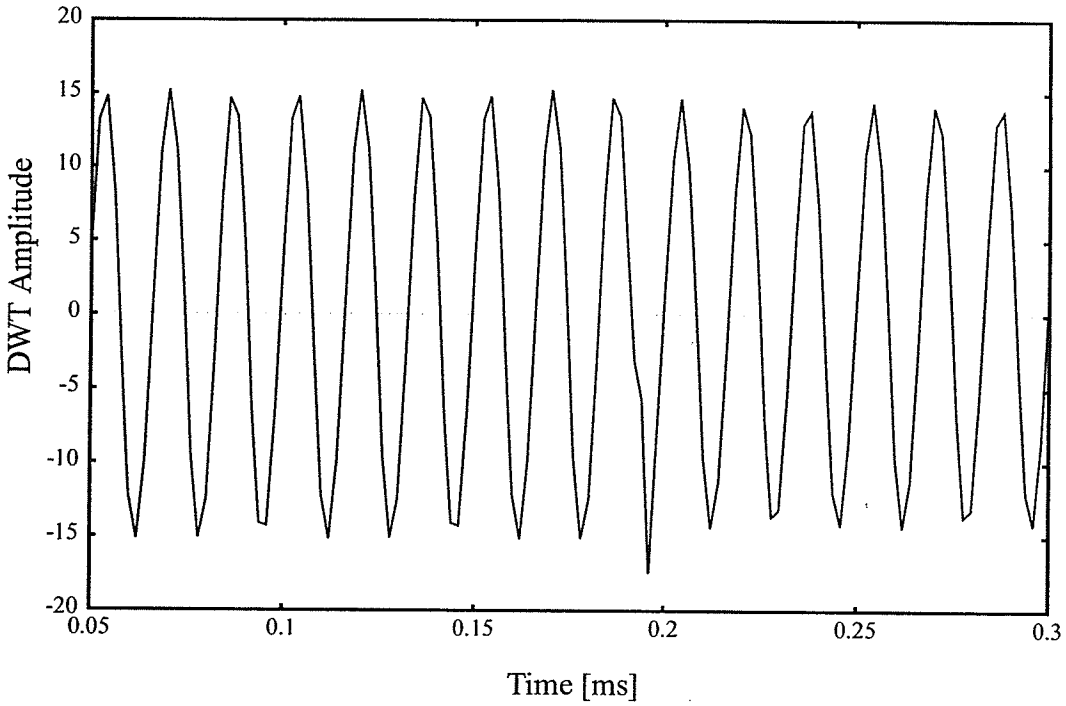

Fig 3.4 (d) DWT of capacitor switching transient at Scale 3.

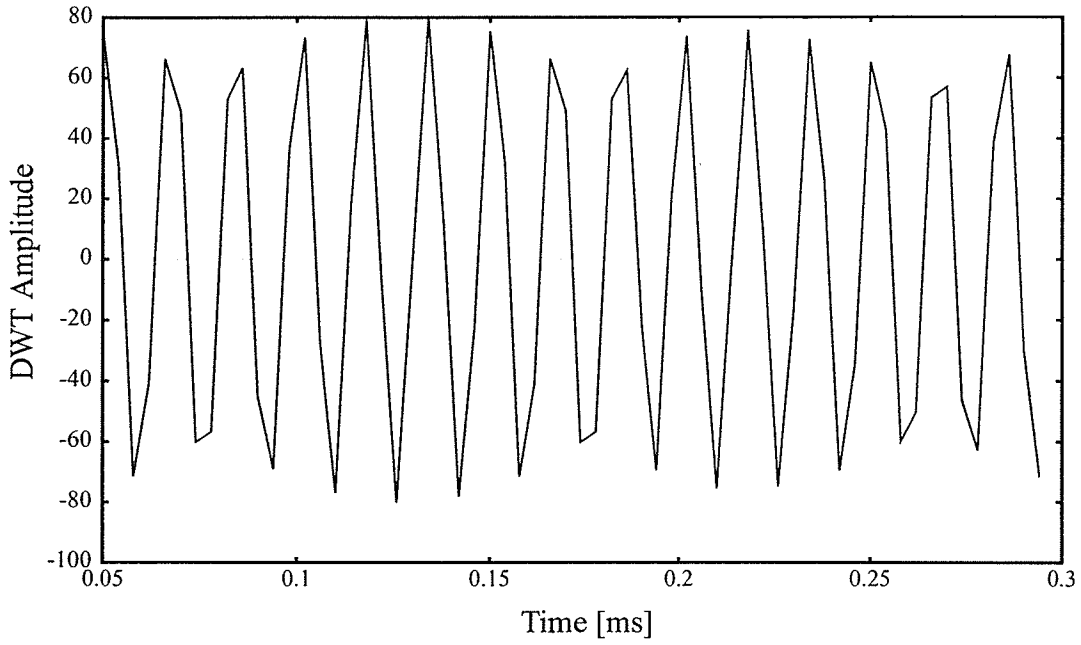

Fig. 3.4 (e) DWT of capacitor switching transient at Scale 4.

From these results at different scales, we see that the patterns from capacitor bank switching operation are totally different from the single-phase fault. But the same identifying process that is used to identify the single phase fault transient can be reused. Based on the information given in the scale 1 of DWT decomposition, the timing information of the transient event, the beginning and end, can be obtained. From the information given in scale 4, the local variance of voltage at fundamental frequency can be obtained. From the degree of the voltage variance at this fundamental frequency, we can see that this transient is not a voltage sag, nor a voltage swell. This type of transient is called oscillatory transient, with disturbance duration shorter than sags and swells. Oscillatory transients often result from the capacitor switching. Such cases may happen when utility capacitor banks are customarily switched into service early in the morning in anticipation of a higher power demand.

From the above two experimental results, we can see that wavelet transform approach is

especially suitable for the detection of system transient. For those FT based method, such short time signal may not be easy to catch on the time axis. However, due to the localization capability on both time and frequency domain in wavelets, the detection of transients and classification of transients can be efficiently solved.

3.2.5 A Two Dimensional Representation of Transient Signals in DWT

In order to give a intuitive demonstration of the time-scale representation capability of DWT, the tilling of time-scale plane by DWT from these two types of transients are visually represented in this thesis. The square values of the DWT coefficients at different scales and different time are calculated, and then mapped to different grey level of a digital image. One digital image software on Solaries, XV, is used to load the image data file, and display the image. The computation is performed on a SUN-Sparc Ultra workstation.

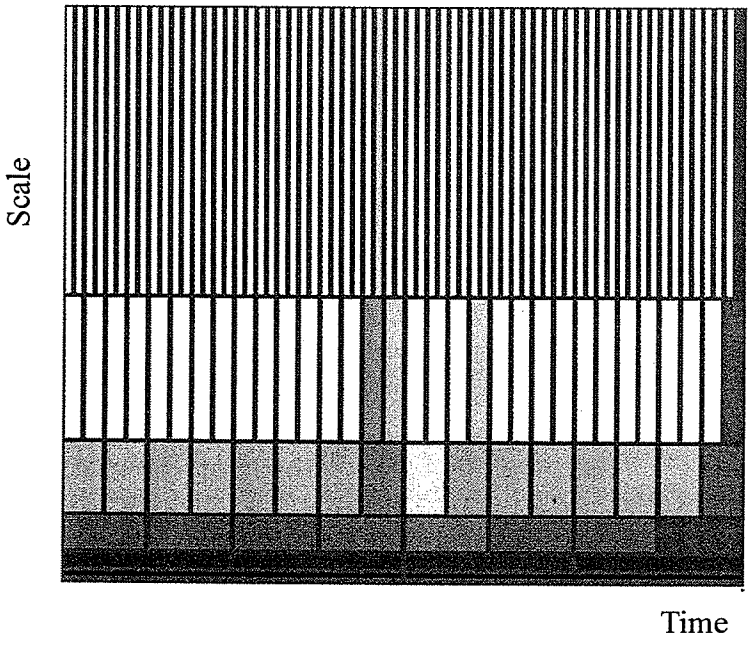

Fig. 3.5 (a) The tilling of time-scale plane by wavelets for single-phase fault.

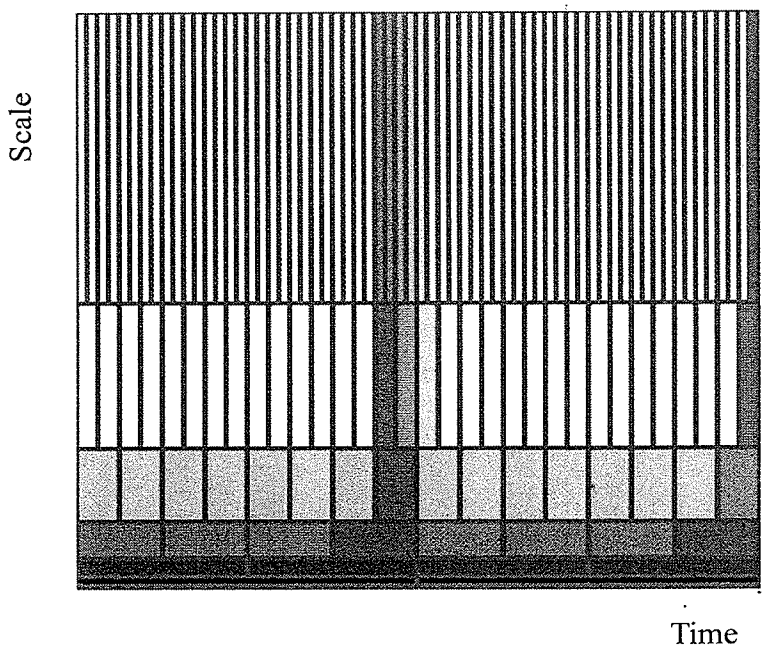

Fig. 3.5 (b) The tilling of time-scale plane by wavelets for switching operation.

The tilling of time-scale plane by DWT from these two types of transients are shown in Figs. 3.5 (a) to (b). From Figs. 3.5 (a) to (b), we can see that the energy of the transient signals is distributed in the time-scale two dimensional plane, instead just frequency domain given by FT. So the timing information of transient event is given in higher frequency band, whereas the variance of voltage at fundamental frequency is also clearly shown.

3.2.6 A More Compact Way of Representing Transient Signals by DWT

In order to classify transient signals, a more compact way of representing signals is preferred because the features of the transient signals would be more distinguished, and the input space would be much smaller. The training time of any classifier would be much faster if the input vector has lower dimension.

In order to demonstrate that DWT is a much more compact way to represent power transient, a reverse DWT is performed on the DWT coefficients to reconstruct the decomposed transient signal.

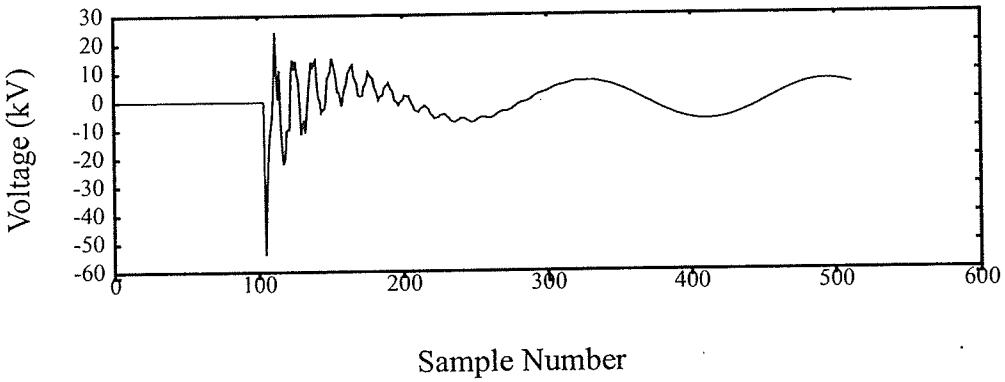

Fig. 3.6 (a) The simulated zero order voltage during a single fault.

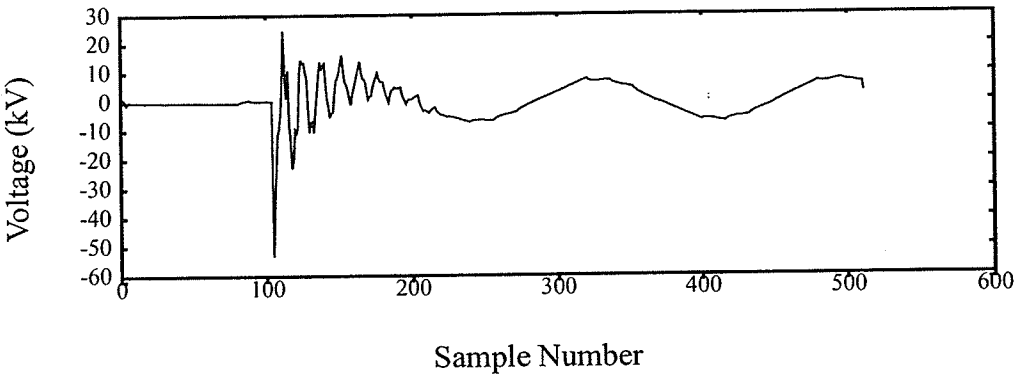

Fig. 3.6 (b) Reconstructed transient signal using 13% DWT coefficients.

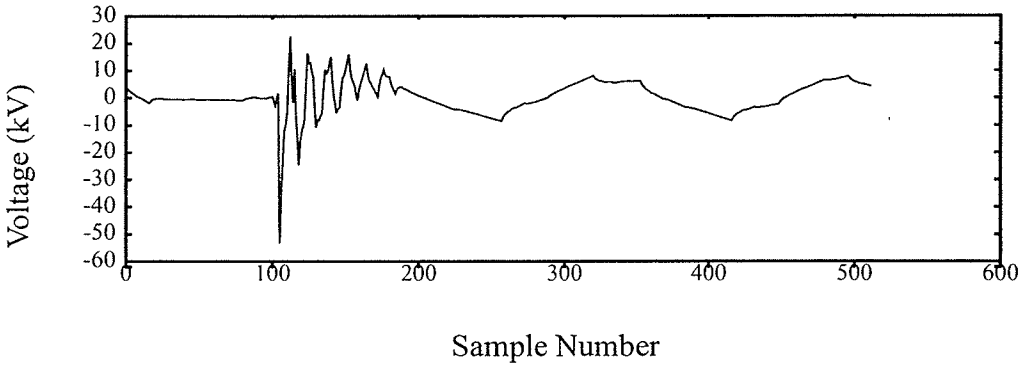

Fig. 3.6 (c) Reconstructed transient signals using 8% DWT coefficients.

Figs. 3.6 (a) shows a simulation result of the zero order voltage during a single phase fault. The total samples simulated are 512 samples.

Figs. 3.6 (b) shows the reconstructed transient signals using only 13% DWT coefficients (less than 70 coefficients). Figs. 3.6 (c) shows the reconstructed transient signals using only 8% DWT coefficients (less than 42 coefficients). From the Figs. 3.6 (c), we can see that even after a massive truncation, the major features of the transient signal are still kept.

3.3 Applications of Wavelet Modelling

One major application of wavelet modelling is to develop an intelligent transient recorder. The limitations of present transient recorders include:

(1) Manual analysis and classification;

(2) Without selective storage ability, the efficiency utilizing limited storage medium is low;

(3) Rigid trigger thresholds set for capturing transients, adaptation exclude changes in

power systems; and

(4) Poor ability to provide real-time information, difficult to be used for real-time control of power systems.

The next generation of transient recorder will overcome these shortcomings. A diagram of design of the intelligent transient recorder is shown in Fig. 3.7.

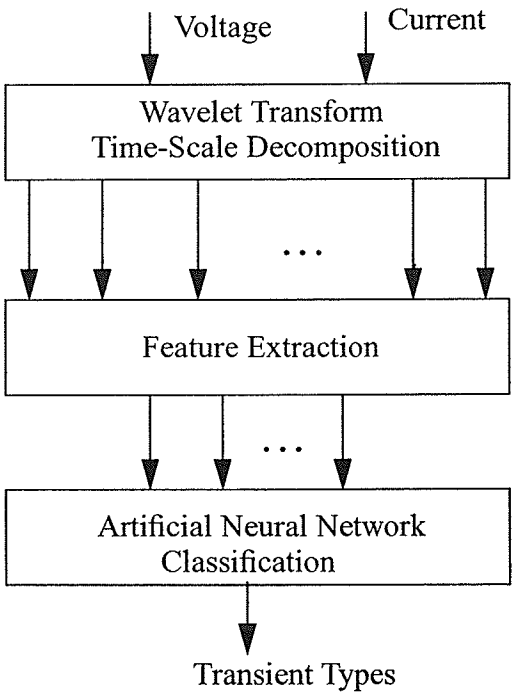

Fig. 3.7 The diagram of an intelligent transient classification system.

First, the input transient signals are decomposed using wavelet transform to obtain a time-scale representation. Then a feature modelling and extraction is applied to get a compact set of features. Finally the extracted features can be fed into an artificial neural network to perform transients classification. An experimental implementation of this system will be presented in Ch. 4.

3.4 Summary

In this chapter, the wavelet transform of two major types of transient signals are presented. The time-scale two dimensional representation capability of WT allows:

- The timing information of sharp signal changes to be detected (voltage sags, voltage swells, capacitor switching) at the high frequency with the short duration window;
- The local variance of magnitude of fundamental frequency of transient signals to be measured accurately;
- The much more compact representation of transient signals; and
- The intuitive time-scale two dimensional representation of transient signals.

The system diagram of the new power system transient recording and classification system based on WT modelling is given in this chapter. An implementation of such a system will be presented in the next chapter.

CHAPTER IV
POWER SYSTEM TRANSIENT CLASSIFICATION USING PROBABILISTIC NEURAL NETWORKS

4.1 Introduction

The use of pattern recognition for power system security analysis was first investigated in 1968. Since artificial neural networks (ANNs) can be applied for pattern recognition, it has been widely investigated for power system transient classification (including faults) [KoJo98].

Among many types of neural networks, the multilayer feed-forward neural network, also known as backpropagation (BP) network, has become the main architecture of choice. One problem with this method is that the backpropagation training algorithm is a stochastic searching algorithm to find the point with a minimum error in the error space. If the searching space is large, the training time becomes prohibitively long, without the guaranteed global minimum. Another drawback of such an architecture is that it is difficult to decide not only how many layers are needed in the neural networks but also how many neural nodes are required in each layer. All these parameters have to be tried experimentally, which is very time consuming. Furthermore, if the architecture of a neural network is decided after these tries and then fixed, the network will have difficulties to adapt itself to a new environment if the new environment is much different from the original environment. This happens when the environment is nonstationary, which means the statistical characteristics of the input from the environment to the neural networks are always changing with time. Since many power systems are expanding and new types of facilities are added to

systems, new transient types with different appearances must be added to the neural network. The BP network cannot incorporate such changes easily. One possible way is to retrain the network with all the past and new training samples, which would be very time consuming. Another way is to only use the new samples to train the neural network, but it may lead to an overall performance degradation caused by the new training samples. The basic reason for the degradation is that the BP network is a globally connected network, thus new samples affect all the weights of the networks globally. Another problem of BP network is that it is likely to be trapped to a local minima.

To overcome the shortcoming mentioned above, the probabilistic neural network (PNN) first proposed by Specht in 1988 [Spec88], was implemented in this thesis for power transients classification. The PNN can compute nonlinear decision boundaries which approach the Bayes optimal decision surface, and minimize the expected error of classification, thus improve its classification accuracy. Furthermore, due to its modular architecture and fast learning speed, the PNN is very suitable for learning systems which are always changing and evolving, and the on-line and incremental learning can also be easily achieved.

Section 4.2 gives a full description of the PNN, including the architecture, the foundation, and the training algorithms. Section 4.3 describes the application of the PNN in power system transient classification.

4.2 Probabilistic Neural Network (PNN)

In 1988, Specht from Lockheed Missiles and Space Company, first proposed a new classification scheme using a new kind of neural networks, which is far different from the traditionally used neural networks trained by backpropagation algorithm. Because this kind of neural networks

has a unique feature that under certain easily met conditions, it can asymptotically approach the Bayes optimal decision surface, Specht gave a new name to this type of neural network, probabilistic neural networks.

In the following section, the architecture of the PNN is first discussed.

4.2.1 PNN Architecture

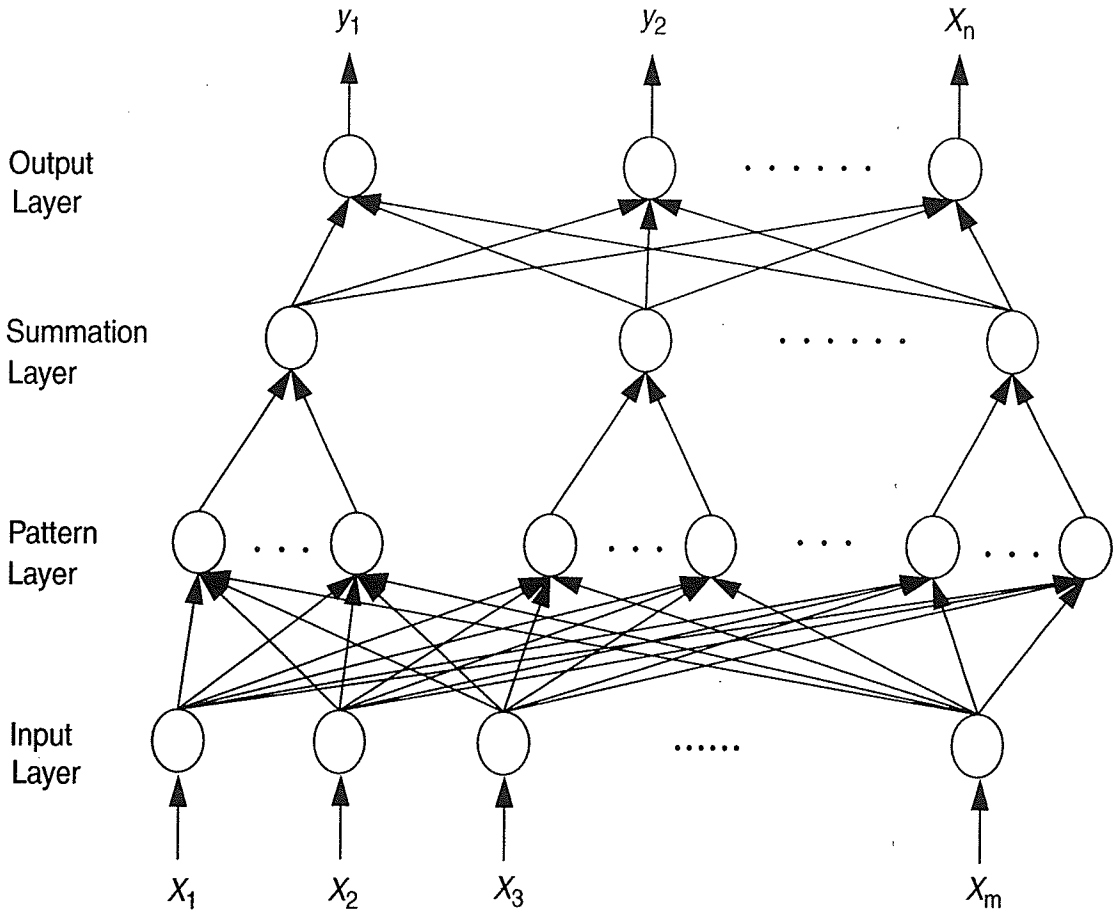

Fig. 4.1 Architecture of PNN.

The architecture of the PNN is shown in Fig. 4.1. The PNN consists of several layers of

parallel processing units, or neurons. The exact number of neurons in each layer is determined by the number of samples representing a class in a training set.

The number of input neurons is equal to the number of separable parameters used to describe the objects to be classified.

The pattern layer contains one neuron for each training case. Specifically, it represents a certain number of training samples (j, k, l,...) from each of C different classes, for a total of R training samples. In brief, each neuron in the pattern layer computes a distance measure between the unknown input and the training case represented by the neuron. An activation function, known as the Parzen window, is applied to the distance measure.

In the summation layer, there is one neuron for each of C classes. These neurons sum the values of the pattern layer neurons corresponding to that class to obtain an estimated density function of the class.

The number of neurons in the output layer is also equal to the number of classes, C. The output layer is often a simple threshold discriminator which activates a single neuron to represent the projected class of the unknown sample.

The basic idea behind the PNN is the Bayes classification rule and Parzen's method of density function estimation. The architecture and computation units of PNN are shown in Fig. 4.2.

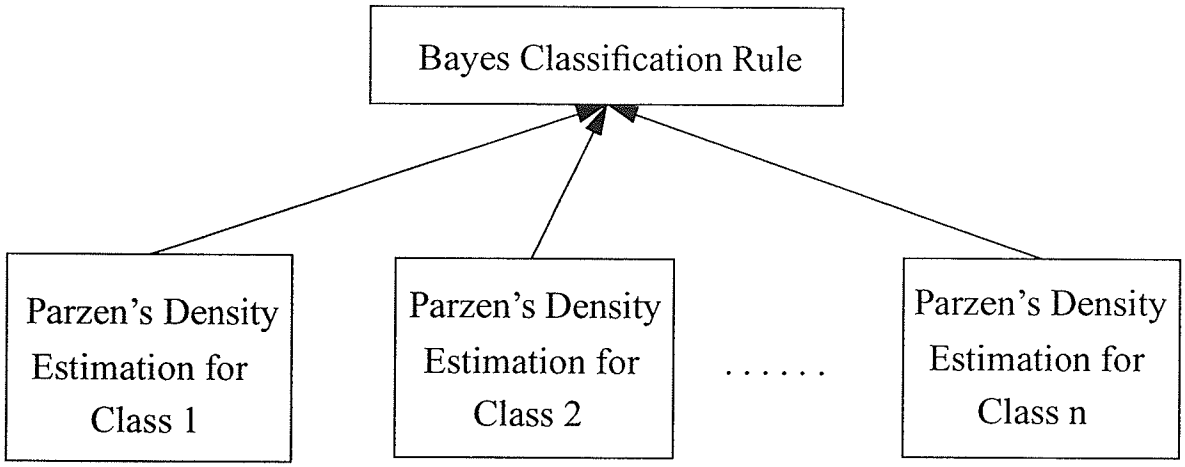

Fig. 4.2 Computational model of PNN.

4.2.2 An Intuitive Insight of the PNN

The PNN is a classifier. A good classifier should have some important properties which are demonstrated in the following examples. From these examples, some intuitive insights about a good classifier can be achieved before examining the properties of the PNN and its mathematical foundation.

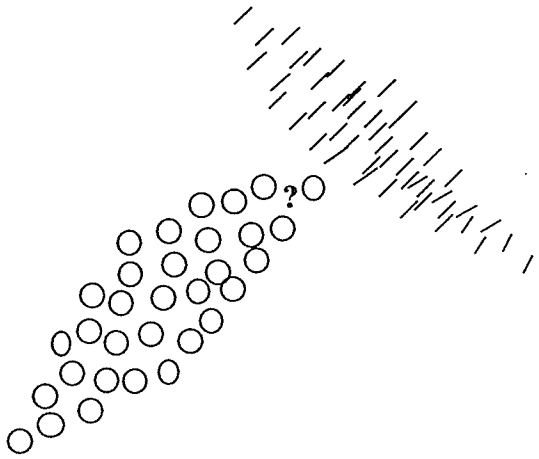

Fig. 4.3 Classification must use the overall shape of a cluster.

From Fig. 4.3, it is clear that the unknown, represented by a question mark, belongs to the class represented by the circles, inspite of the fact that it is actually closer to the centroid of the class represented by the strokes. So a good classifier examines more than just the central tendencies. It takes into account the total pattern into which class members fall.

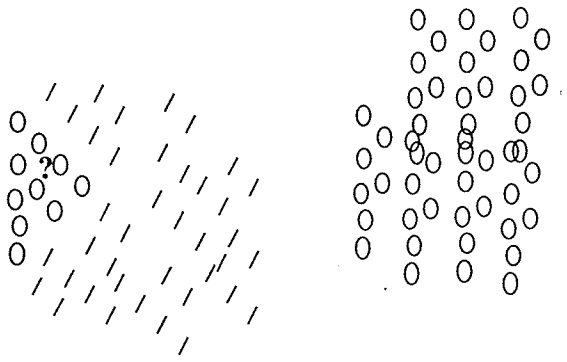

Fig. 4.4 Classes may have multiple modes.

Fig. 4.4 shows an excellent separation between the two classes, so we should be able to construct a good classifier. But many traditional classification algorithms would fail when confronted with the bimodal distribution of one of these classes. Thus, a good classifier should be able to handle multiple modes.

A good classifier should also consider the density of given samples. The nearest-neighbour classifier does not consider this. The situation that the nearest-neighbuor does not make right decision is shown in Fig. 4.5.

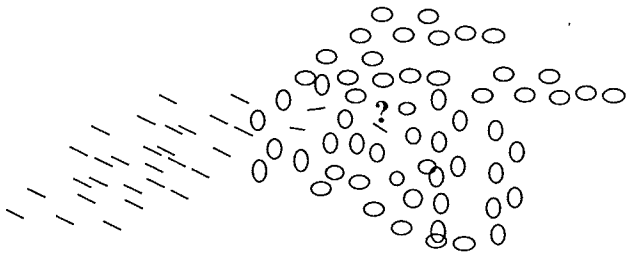

Fig. 4.5 Nearest-neighbor methods can fail.

From the above figure, it should be apparent that the unknown probably belongs to the right class, and we must not be misled by the fact that it is closer to a member of the left class than to any member of the right class. The reason is that the density of the left class is very thin where the unknown is located, while the density of the right class is much larger. Since there are more members of the right class in the vicinity, we conclude that the unknown is more likely a member of the class than of the left class, which is very sparse there. This intuition is at the core of the PNN development that follows.

4.2.3 Bayes Strategy for Pattern Classification

An accepted norm for decision rules or strategies used to classify patterns is that they do so in such a way that minimizes the "expected risk". Such strategies are called "Bayes strategies" [MoGr62] and can be applied to problems containing any number of categories.

Consider the two-category situation in which the state of nature θ is known to be either θ_A or θ_B. If it is desired to decide whether $\theta = \theta_A$ or $\theta = \theta_B$ based on a set of measurements

represented by the p-dimensional vector $X^t = [x_1 \ldots x_j \ldots x_p]$, the Bayes decision rule becomes

$$d(X) = \theta_A \quad \text{if} \quad h_A l_A f_A(X) > h_B l_B f_B(X)$$

$$d(X) = \theta_B \quad \text{if} \quad h_A l_A f_A(X) < h_B l_B f_B(X) \tag{4.1}$$

where $f_A(X)$ and $f_B(X)$ are the probability density functions for categories A and B, respectively; l_A is the loss function associated with the decision $d(X) = \theta_B$ when $\theta = \theta_A$; l_B is the loss associated with the decision $d(X) = \theta_A$ when $\theta = \theta_B$ (the losses associated with correct decisions are taken to be equal to zero); h_A is the *a prior* probability of occurrence of patterns from category A; and $h_B = 1 - h_A$ is the *a prior* probability that $\theta = \theta_B$.

Thus the boundary between the region in which the Bayes decision $d(X) = \theta_A$ and the region in which $d(X) = \theta_B$ is given by the equation

$$f_A(X) = K f_B(X) \tag{4.2}$$

where

$$K = h_B l_B / h_A l_A \tag{4.3}$$

In general, the two-category decision surface defined in equation (4.2) can be arbitrarily complex, since there is no restriction on the densities except those conditions that all *probability density functions* (PDF) must satisfy that they are everywhere non-negative, that they are integrable, and that their integrals over all space equal unity.

The key to using equation (4.2) is the ability to estimate PDFs based on training patterns accurately.

4.2.4 Parzen's Method of Density Estimation

The role of the pattern layer and summation layer in PNN is actually the realization of Parzen's method of density estimation. In 1962, Parzen presented a method of estimating a univariate PDF from a random sample [Parz62]. His PDF estimator converges asymptotically to the true density as the number of samples increases towards a full representation of the class data.

Formally, if there are n_c training cases for a given class, c, and the training sample is a scalar, then the estimated PDF for that class, $g_c(x)$, is

$$g_c(x) = \frac{1}{n_c \cdot \sigma} \cdot \sum_{i=1}^{n_c} W\left(\frac{x - x_i}{\sigma}\right) \tag{4.4}$$

where σ defines the width of the bell curve that surrounds each sample point. Proper choice of the value for σ is critical to the performance of the PNN. If σ is too small, individual training cases will be considered only in isolation and we will be left with essentially a nearest-neighbour classifier [Mast95]. However, if the value of σ is too high, details of the density will be blurred together and, unless the different classes are very well separated, confusion would certainly result. Figure 4.6 shows a density function estimation result.

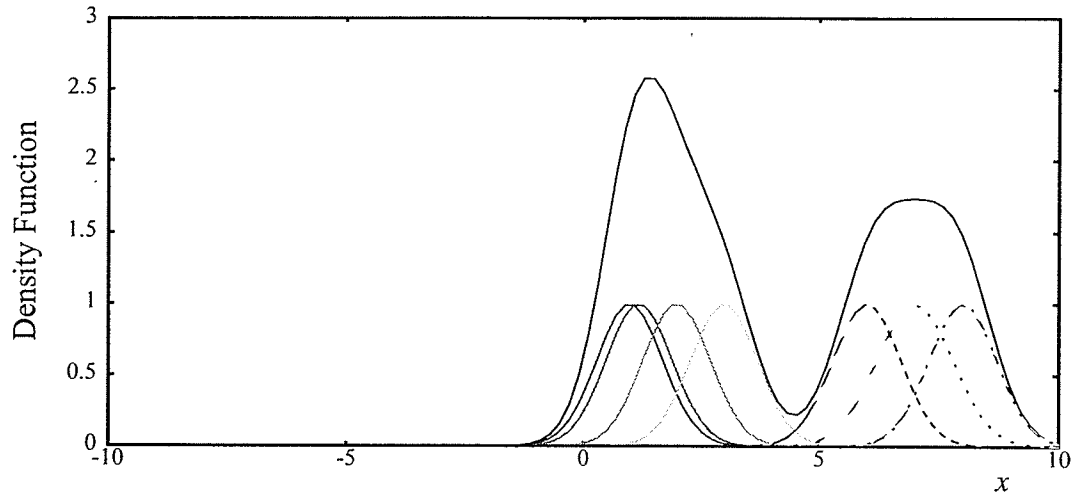

Fig. 4.6 Density function estimation.

The effects of σ are shown through Figs. 4.7 to 4.9.

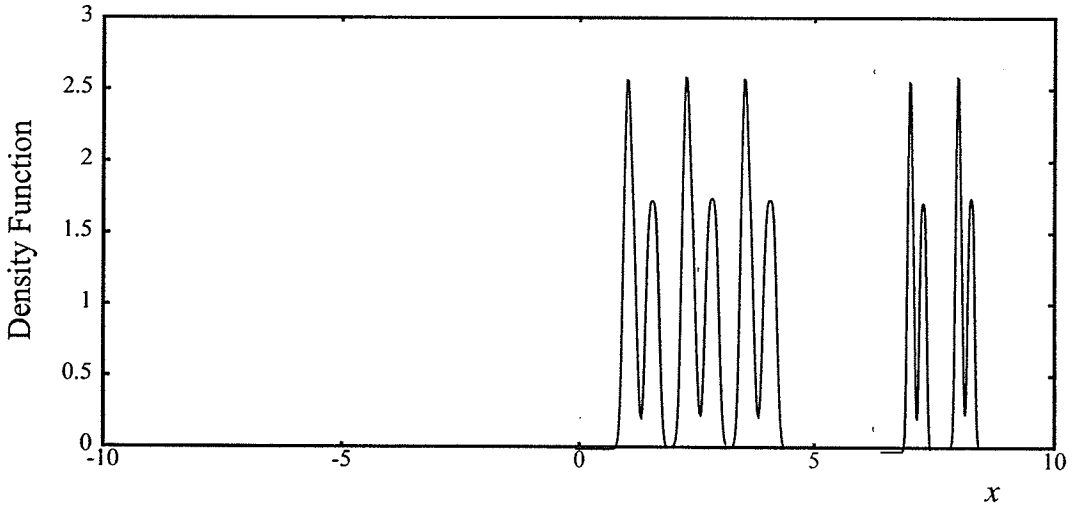

Fig. 4.7 Scaling parameter σ too small.

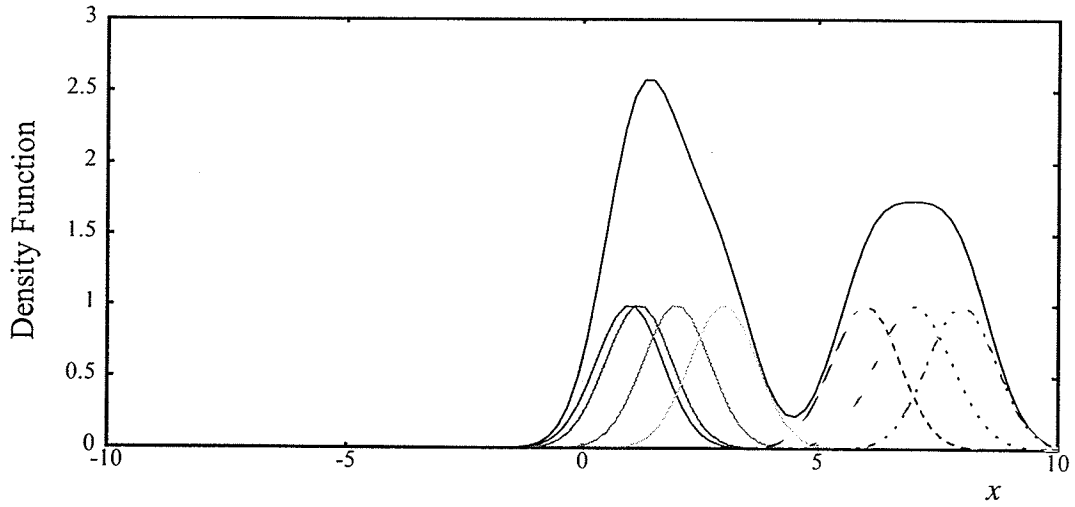

Fig. 4.8 Scaling parameter σ just right.

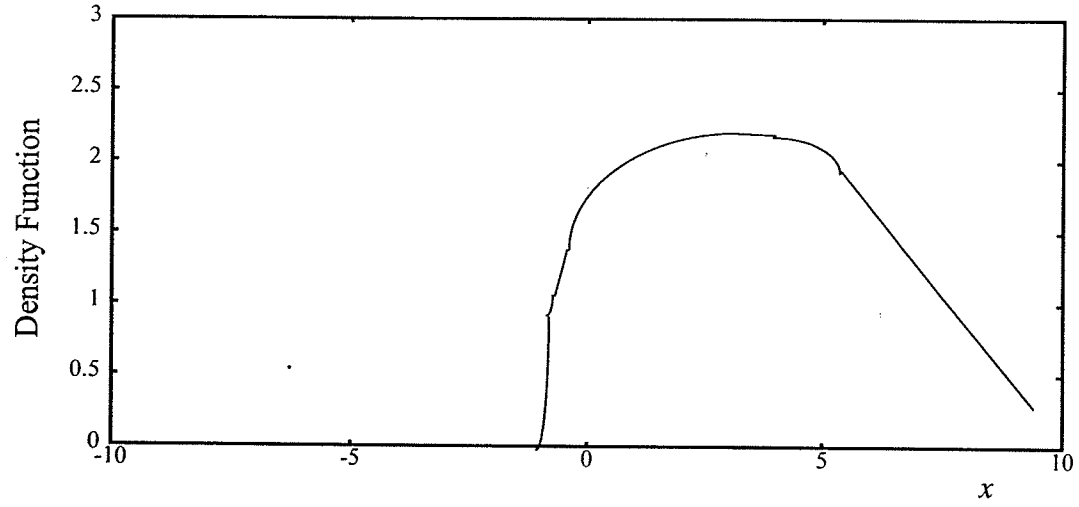

Fig. 4.9 Scaling parameter σ too large.

There are many choices for the weight functions. The candidate weight functions should satisfy the following conditions:

1) The weight function must be bounded

$$sup_d|W(d)| < \infty \tag{4.5}$$

2) The weight function must rapidly go to zero as its argument increases in absolute value

$$\int_{-\infty}^{\infty} |W(x)|dx < \infty \tag{4.6}$$

$$\lim_{x \to \infty} |xW(x)| = 0 \tag{4.7}$$

3) The weight function must be properly normalized

$$\int_{-\infty}^{\infty} W(x)dx = 1 \tag{4.8}$$

The weight function, *W(x)*, is always chosen to be the normalized Gaussian function as

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \tag{4.9}$$

where σ is the scale factor of the Gaussian function.

While other weight functions are theoretically possible, the Gaussian is well-behaved and ideally shaped for this implementation. It has been shown in practice to perform well and is almost always the function of choice for practical applications.

Since the PNN has *p* inputs, the PDF estimator must take into account all of these input variables in order to achieve an accurate density estimation. In 1966, a means to extend Parzen's method to the multivariate case was introduced [Caco66]. It is more complicated because it allows each of the input values to have its own scale factor, σ . This fully general density estimator for

a class, c, is given by

$$g_c(x_1, \ldots, x_p) = \frac{1}{n_c \sigma_1 \ldots \sigma_p} \cdot \sum_{i=1}^{n} W\left(\frac{x_1 - x_{1,i}}{\sigma_1}, \ldots, \frac{x_p - x_{p,i}}{\sigma_p}\right) \tag{4.10}$$

where n_c is the number of samples which exist for the class as in Eq. 4.4. For a simple case, all the scale factors are assumed to be equal such that $\sigma_1 = \sigma_2 = \ldots = \sigma$. Then the multivariate weight function become the product of the univariate weight functions as shown in

$$W(x_1, \ldots, x_p) = \prod_{i=1}^{n_c} W_i(x_i) \tag{4.11}$$

Now, with these simplification, the density estimator can be stated explicitly as

$$g_c(X) = \frac{1}{(2\pi)^{p/2} \sigma^p n_c} \cdot \sum_{i=1}^{n_c} e^{-d_i^2} \tag{4.12}$$

where d_i^2 is the square of the normalized distance given by

$$d_i^2 = \frac{\|X - X_i\|_2^2}{2\sigma^2} \tag{4.13}$$

and the norm $\|\bullet\|_2$ is the Euclidean norm of the distance between an unknown input sample X and a training sample X_i.

The summation neurons perform a simple summation of the weighted distance measurements for the class they represent

$$g_c(X) = \frac{1}{n_c} \cdot \sum_{i=1}^{n_c} e^{-d_i^2} \tag{4.14}$$

For the simple case $\sigma_1 = \sigma_2 = \ldots = \sigma$, the value of $(2\pi)^{p/2}\sigma^p$ is the same for all the classes considered, so it is omitted in Eq. 4.14.

The output layer of the PNN implements Eq. 4.1. All of the outputs from the neurons in the summation layer are examined and the largest one is selected. Then, only the output layer neuron which corresponds to that neuron in the summation layer is activated. The index of the activated neuron in the output layer represents the predicted class of the sample at the input layer.

4.2.5 Bayesian Confidence Measure

One important property of the PNN is that it has the ability to provide confidence estimates for its decisions. According to Bayes' theorem, when the classes are mutually exclusive and exhaustive (meaning that no case can possibly fall into more than one population, and that the training set encompasses all populations fully, then the probability that an observation X was the product of population A can be calculated by

$$P[A|X] = \frac{g_A(X)}{\sum_k g_k(X)} \tag{4.15}$$

This property of the PNN is the distinguishing feature as compared to some other neural networks. Neural networks like multiple-layer feedforward networks have been shown to be

excellent classifiers, but there is no way to calculate the probability that an observation X belong to population A. So there is no confidence measure regarding to any decisions made by these types of neural networks.

4.2.6 The Training of PNN

The training process for the PNN is very simple. The purpose of the training is to find an optimal value of σ which makes the misclassification error minimum. The process of the training involves three steps:

1) Prepare and separate the training set into 2 classes, one for training, one for calculating misclassification rate.
2) The centres of all the neurons in the pattern layer of the PNN are loaded, using the training set of samples; and
3) The optimization of σ is performed to obtain the minimum misclassification error based on the sample set used for calculate misclassification. The optimization is performed through a global search.

4.3 Application of PNN in Power System Transient Classification

4.3.1 Simulation of Power System Transients

To demonstrate the suitability of the PNN for power system transients classification, a PNN is developed to identify different transient patterns generated from a simple transmission system which is represented in Fig. 4.10.

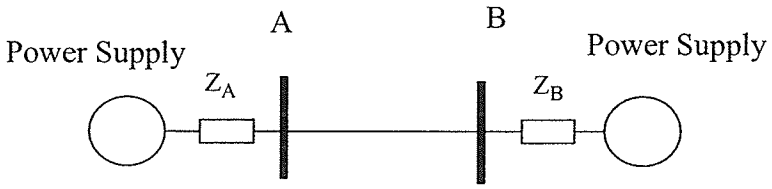

Fig. 4.10 Power system used for the PNN studies.

The training patterns are created to cover a variety of fault inception angles, fault resistance, source capacities and loads.

Three different types of disturbances are simulated on the simple power system. One type is caused by a short duration single-phase fault cleared in time. The second type is caused by a longer duration single-phase fault cleared too late. In this thesis, if the clearing time of a fault is within 3 cycles, the fault is classified as first type, otherwise classified as second type. The third type of disturbance is caused by capacitor switching, which is a frequent operation. The testing of the PNN is to be able to distinguish the three types of disturbances accurately. The waveforms of this three types of disturbances are shown in Figs. 4.11 (a) to (c).

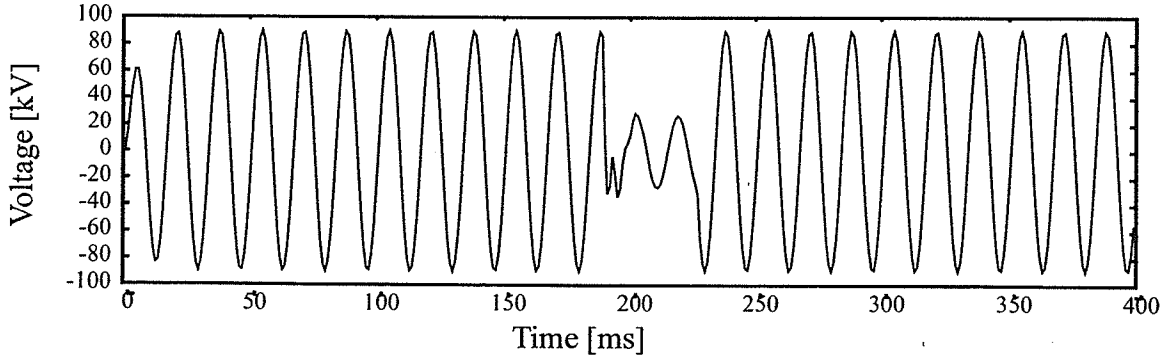

Fig. 4.11 (a) Single-phase fault, cleared in time.

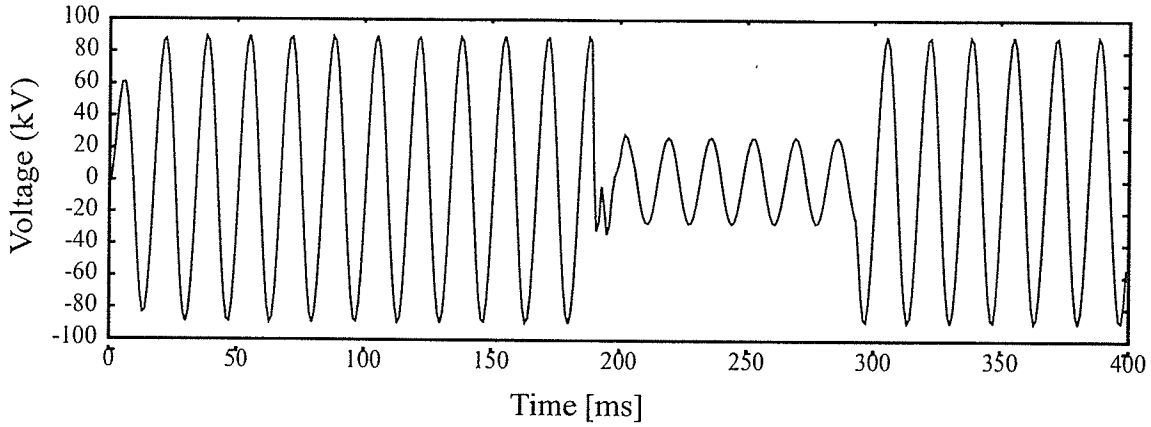

Fig. 4.11 (b) Single-phase fault, not cleared in time.

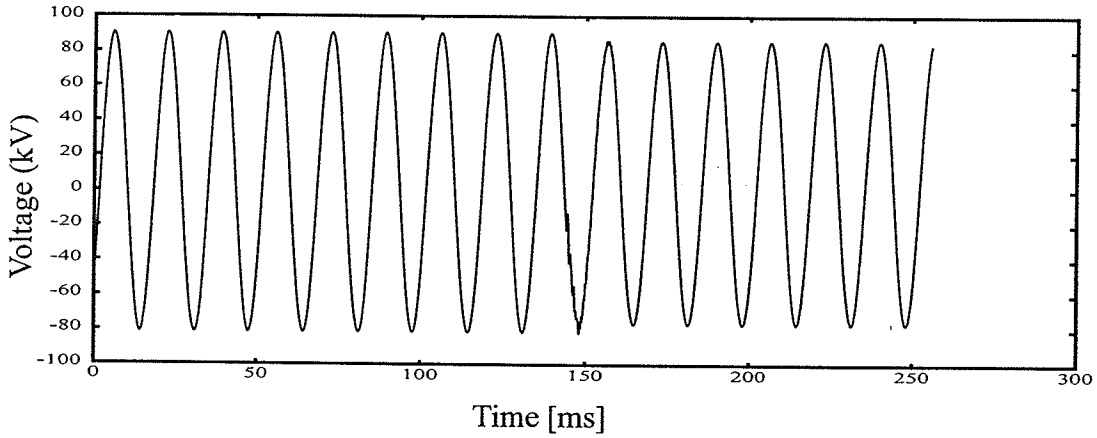

Fig. 4.11 (c) Transient caused by capacitor switching.

There are two reasons for selecting two different types of single-phase fault for classification. First, if a fault is not cleared in time, it may indicate the circuit breaker has problem, which should be fixed right away. Another reason is to test the capability of WT of extracting not only the frequency information, but also the time information.

In the real situation, there is always noise in power systems. Therefore, the effect of noise

on the transient classification system is investigated. Three types of noise are considered in the thesis, Gaussian white noise, pink noise, and black noise. These noise are added to the simulated power signals with certain *signal to noise ratio* (SNR). The following is the definition of SNR [Kins98]

$$SNR = \left(\sum_{i=1}^{N} x[i]^2 \right) / \left(\sum_{i=1}^{N} n[i]^2 \right) \tag{4.16}$$

or

$$SNR_{dB} = 10\log_{10}(SNR) \text{ [dB]} \tag{4.17}$$

where $x[i]$ is a pure transient signal, $n[i]$ is a noise, and N is the sample length of the transient signal and noise.

4.3.2 Experiment Description

There are two types of experiments are performed in the thesis. The first one is to investigate the PNN classification performance, and the performance differences between using raw data as input and using DWT pre-processed data as input. The second experiment is to investigate the effect of different noises on the classification system.

The sampling rate of the input is 1 kHz, and all the input are normalized. The signal segment containing disturbances are extracted. The extracted part is then either fed directly into the PNN, or the DWT coefficients of the extracted segment are used as input. The first 4 scales of DWT coefficients are used in the classification because most of the important information are contained in these scales. For each class of transient, there are 48 training samples and 48 testing samples, respectively. The input variable size is 64. After the training process, the testing samples

are used to evaluate the trained PNN.

Table 4.1: Summary of pure transients in experiment

Disturbance Name	Number of disturbances	Class Identifier
Single-phase fault cleared in time	96	0
Single-phase fault, not cleared in time	96	1
Capacitor bank switching transient	96	2

Table 4.2: Summary of noisy transients in experiment

Disturbance Name	Number of disturbances with white noise at 10 and 20 dB SNR	Number of disturbances with pink noise at 10 and 20 dB SNR	Number of disturbances with black noise at 10 and 20 dB SNR	Class Identifier
Single-phase fault cleared in time	32	32	32	0
Single-phase fault, not cleared in time	32	32	32	1
Capacitor bank switching transient	32	32	32	2

The PNN used in the thesis is a standard PNN, with one scaling parameter σ. The classification ability of the PNN is evaluated based on the correct classification rate and confidential level.

The training algorithms for the single sigma PNN has been described in Section 4.2.6. The training and testing of the transient classification system are performed on a SUN-Sparc Ultra workstation. The software implementation for training the PNN is accomplished in C++ language.

4.3.3 Experimental Results and Discussion

4.3.3.1 Training and testing using raw data vs. DWT coefficients

The training and testing took about a couple of seconds. This demonstrated the fast training speed of PNN comparing to other neural networks. Using BP network to do the same classification took about five minutes with lower correct classification rate (90%).

After the training on the 48 training patterns for each class of transients, the single-sigma PNN can correctly classify the testing set of disturbances with 100% correct rate.

The classification results of this experiment can be described using confusion matrix, a standard tool for testing any type of classifier. Table 4.3 shows the confusion matrix with the classification results when the raw data without noise are used as input to the PNN.

Table 4.3: Classification results of the PNN for pure raw data

Class	0	1	2
0	48		
1		48	
2			48
Unknown			
Total	48	48	48
P_c	1.0	1.0	1.0

As shown in the above table, the confusion matrix has one row and one column for each class. The row represents the original class and the column means the predicted class by the PNN. If the correct classification rate is 100%, the confusion matrix would be a pure diagonal matrix. In

this experiment, since all the testing set of disturbances are correctly classified, the matrix is a pure diagonal matrix.

The classification results using DWT pre-processed data are shown in Table 4.4.

Table 4.4: Classification results of the PNN for DWT pre-processed data without noise

Class	0	1	2
0	48		
1		48	
2			48
Unknown			
Total	48	48	48
P_c	1.0	1.0	1.0

From the above table, we can see that the PNN can also achieve 100% correct classification rate when the input are DWT coefficients of the disturbances. From the classification rate, there is no differences, but from the range of the sigma σ within which the correct classification rate are 100%, and the confidence level, we can see there is a significant difference between the two cases. The average correct classification rate and confidence results for two cases are shown in Figs. 12 and 13.

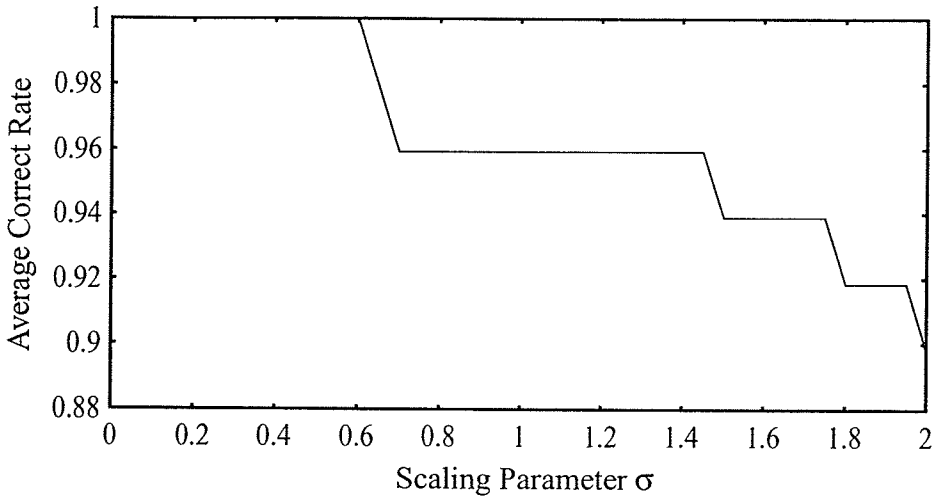

Fig. 4.12 (a) Correct classification rate vs. σ, with raw input data.

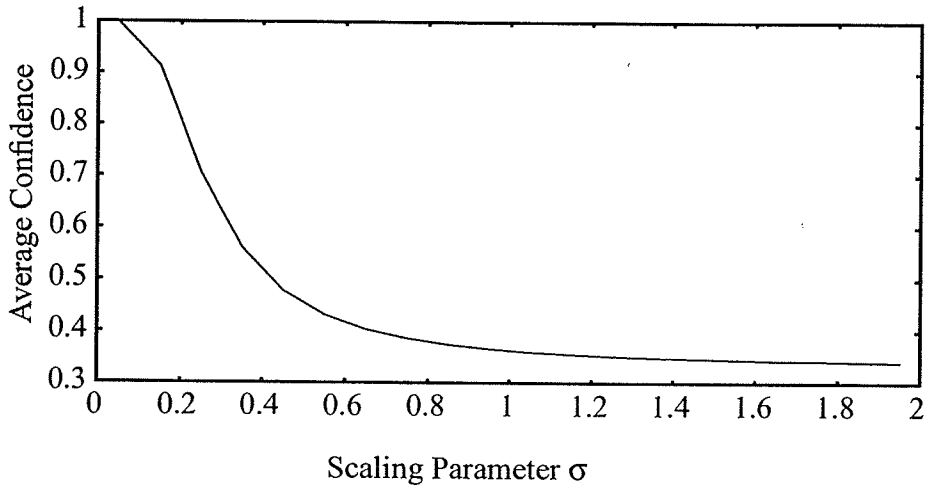

Fig. 4.12 (b) Average confidence vs. σ, with raw input data.

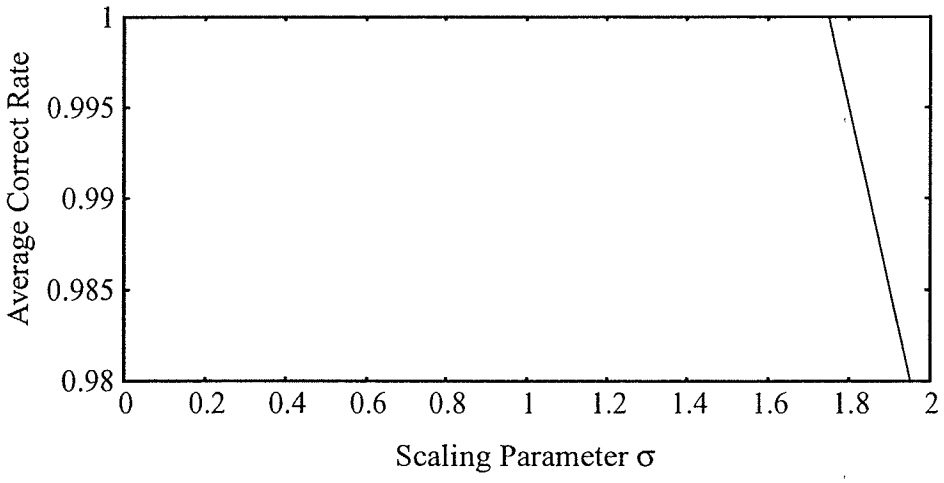

Fig. 4.13 (a) Correct classification rate vs. σ, with DWT coefficients as input data.

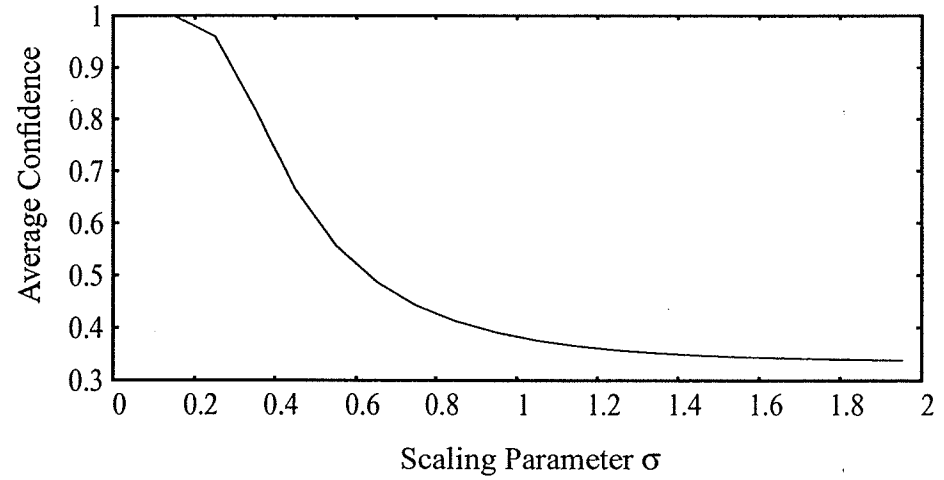

Fig. 4.13 (b) Average confidence rate vs. σ, with DWT coefficients as input data.

From Figs. 4.12 and 4.13, we can see that with the optimal σ value selected, the PNN can classify samples in testing set with 100% correct rate. In both cases, with the increasing of the value of σ, the classification accuracy drops and the average confidence also decreases. This

reflects the fact that when the value of σ is too large, the density approximation is less accurate.

Compared to the case in which raw data are applied as inputs, when using DWT coefficients as inputs, there is a wider range of σ in which both correct classification rate and average confidence are high. This demonstrated that the preprocessing of raw data by applying DWT can increase the distance between different classes, therefore enhance the differences between classes.

4.3.3.2 Effect of Noises on Classification

In this experiment, the original pure signal are contaminated with three types of noises, white Gaussian, pink, and black noise, each at 10, and 20 dB SNR, respectively. Then the noisy data are pre-processed through DWT. Finally, the output of the DWT are fed into the PNN for training and testing. The effect of different noises on classification are summarized in the following Tables.

Table 4.5: Classification result of the PNN with white noise at 20 dB SNR

Class	0	1	2
0	48		
1		48	
2			48
Unknown			
Total	48	48	48
P_c	1.0	1.0	1.0

Table 4.6: Classification results of the PNN with white noise at 10 dB SNR

Class	0	1	2
0	48		
1		48	
2			48
Unknown			
Total	48	48	48
P_c	1.0	1.0	1.0

Table 4.7: Classification results of the PNN with pink noise at 20 dB SNR

Class	0	1	2
0	47	1	
1	1	47	
2			48
Unknown			
Total	48	48	48
P_c	0.98	0.98	1.0

Table 4.8: Classification results of the PNN with pink noise at 10 dB SNR

Class	0	1	2
0	45	3	
1	2	46	
2			48
Unknown			
Total	47	49	48
P_c	0.96	0.94	1.0

Table 4.9: Classification results of the PNN with black noise at 20 dB SNR

Class	0	1	2
0	47	1	
1	2	46	
2			48
Unknown			
Total	49	47	48
P_c	0.96	0.98	1.0

Table 4.10: Classification results of the PNN with black noise at 10 dB SNR

Class	0	1	2
0	44	4	
1	2	46	
2			48
Unknown			
Total	46	50	48
P_c	0.96	0.92	1.0

From the above tables, we can see that PNN is most tolerate to white noise, and least tolerate to black noise and pink noise. When there is no noise existed, the classification system is very accurate. In a noisy environment, the classification results are quite satisfactory.

4.4 Summary

This chapter proposes a novel transient classification technique by using the combination of DWT and PNN. The PNN is based on Bayes' classification rule and Parzen's method of density

function estimation, and can achieve high accuracy of transient classification rate. Compared with neural networks trained by backpropagation algorithm, the correct classification rate is higher, and at the same time the training time required is much shorter for the same set of training samples. Another distinguishing feature of PNN is that the classification confidence can also be calculated.

A single-sigma PNN is developed for classification of three types of simulated transients. A confusion matrix is used to describe the classification results. The performance of the PNN are described when raw data and DWT coefficients are fed into the networks. Experimental results show that the DWT can increase the confidence level for a wider range of sigma, and the correct classification rate is higher within a broader range of sigma.

The effect of different noise on the classification system is also investigated in this chapter. Experimental results show that the performance of this classification system is satisfactory when white, pink, and black noise are presented. For the three classes of transients, the average classification rate can achieve 92%.

CHAPTER V
MODELLING OF NONSTATIONARY SIGNALS WITH FRACTAL CHARACTERISTICS AND POWER LOAD PREDICTION

5.1 Introduction

Nonstationary signals with fractal characteristics refer to a large family of correlated signals, including black noise, brown noise, and pink noise, whose fractal dimensions are totally different [Kins95]. Nonstationary signals originate from the complex dynamic behaviour of underlying dynamic systems, which should be best described in terms of some nonlinear differential equation. However, in most circumstances, the particular differential equation is either completely unknown, or very difficult to estimate.

The chapter gives a description of a new approach to the modelling of the nonstationary signals using a special class of artificial neural network, the resource allocating network (RAN). One significant feature of a RAN is its ability to allocate resources corresponding to the complexity of nonstationary signals, thus tracking and matching the complexity of nonstationary signals can be achieved. Section 5.2 will discuss the nonstationary signals with fractal characteristics. Section 5.3 gives a description on the common problems of the traditional fix structured neural networks in modelling of nonstationary signals, and demonstrates the ability of RAN to model nonstationary signals with fractal characteristics. Section 5.4 will give a full description of RAN and its usage in modelling of nonstationary signals with fractal characteristics. Section 5.5 will show a direct application of RAN to predict short term power load.

5.2 Nonstationary Signals with Fractal Characteristics

In the physical world, nonstationary signals are dominant. Nonstationary signals refer to a class of signals whose statistical characteristics are not stationary. Inside this class of signals, signals with self-similarity properties, i.e., signals which retain many of their essential characteristics under time scaling, arise frequently in physical processes and are also important in signal generation for communications, remote sensing, and many other applications. This type of signals are also called nonstationary signals with fractal characteristics. This class of signals exhibit statistical self-similarity, e.g., the autocorrelation functions remain invariant to within an amplitude factor under arbitrary scalings of the time axis. In communication traffic, the data flow has shown the self-similarity features and characteristics of black and brown noise. Speech signals have also shown the nonstationary and fractal characteristics [LaKi95]. One important family of random processes which generate signals with fractal characteristics are $1/f$ processes. These processes are often used in modeling natural landscapes, the distribution of earthquakes, ocean waves, turbulent flow, the pattern of errors on communication channels, and many other natural phenomena. Even in music, for a piece of music to be good (pleasing and interesting), it should not be boring (brown noise) or unpredictable (white noise) or; instead, it should follow the hyperbolic law ($1/f$ process, called pink noise).

5.3 Evaluation of Fixed-Structure Neural Networks in System Modelling

One of the major problems of the traditional fixed-structure neural networks (NN) in system modeling and signal processing is overfitting and overparameterization.

"Overfitting" usually means estimating a model that fits the data so well that it ends by including some of the error randomness in its structure, and then produces poor forecasts. In neural networks with fixed structure, this may come about for two reasons: because the model was overtrained, or because it was too complex.

Overfitting however may also be a consequence of overparamerization, that is, of the excessive complexity of the model. The problem is very common in neural networks with fixed structure; since they are often (and improperly) used as "black-box" devices, the users are sometimes tempted to add to them a large number of variables and neurons, without taking into account the number of parameters to be estimated. Many methods have been suggested to "prune" the NN, i.e., to reduce the number of its weights, either by shedding some of the hidden neurons, or by eliminating some of the connections. However, the adequate rate between the number of sample points required for training and the number of weights in the network has not yet been defined clearly. It is difficult to establish, theoretically, how many parameters are too many, for a given sample size.

The underlying dynamic characteristics of the nonlinear nonstationary signals make the modelling using traditional fixed-structure NN even more difficult. The complexity of the underlying dynamic process may constantly change with time, which means even a well trained NN within one period of time may become less accurate when the underlying process evolves, either becoming overfitting or over parameterization, or reaching the computing capacity of the structure provided. When the model becomes overfitting or overparameterized, the model needs to be pruned; when the model reaches its computing capacity, it needs more computing nodes (neurons), to cover the new evolved space of the underlying dynamic system. The constant evolving

nature of dynamic systems requires a model to be able to evolve itself, not only by slight adjustment of the parameters and wights, but also by changing computer structure itself, which means altering not only the number of neurons, but also the connections in the neural network.

To model or predict these nonstationary signals, traditional statistical methods, such as Parzen window approximation, and artificial networks, such as the *radial basis function network* (RBFN), have been applied so far. The Parzen window approximation allocates a new unit for every learned sample to approximate the underlying distribution. The problem with such a method is the high computation complexity when the number of samples is very large.

Unlike Parzen window method, the RBFN techniques uses the centers of each basis function as the prototype of a local region, to memorize an abstraction of the data set explicitly. The number of hidden units could be smaller than the number of training samples. The drawback of traditional implementation of RBFN is the number of centroids of RBFN has to be determined in advance. If the number of centroids is not decided properly, the accuracy of RBFN is seriously affected.

If a neural network is simple, it cannot reflect the complexity of real-world problems, and its accuracy is low. On the other hand, if the structure of the neural network is too complex, the generalization capability could suffer if sufficient training samples are not available, and the computation is very time consuming. The best neural network should have an optimal structure which matches the complexity of the problem.

During the process of modelling and predicting of nonstationary signals, the underlying dynamics may change quickly. An idea model should have the ability to reflect such changes with

corresponding architecture changes, thus matching its architecture complexity to the problem's complexity with time.

A significant progress toward this direction was made by Platt [Plat91] through the development of an algorithm that adds hidden units to the network based on the novelty in the new observed data. The algorithm is suitable for sequential learning and is based on the idea that the number of hidden units should correspond to the complexity of the underlying function as reflected in the observed data.

5.4 The Resource Allocation Network (RAN)

5.4.1 The Basic RAN

The basic RAN was developed as a means to overcome the problem of NP-completeness in learning with fixed-size network [Plat91]. Its motivation was coming from the fact that by allocating new resources, learning could be achieved in polynomial time. Platt views the task of RAN as combining memorization with adaptation, in which memorization is achieved by storing the input-output observation similar to Parzen window estimation method. He improves on this method by storing fewer observations which makes the size of RAN grow sublinearly and eventually saturate. The architecture of RAN is shown in Fig. 5.1.

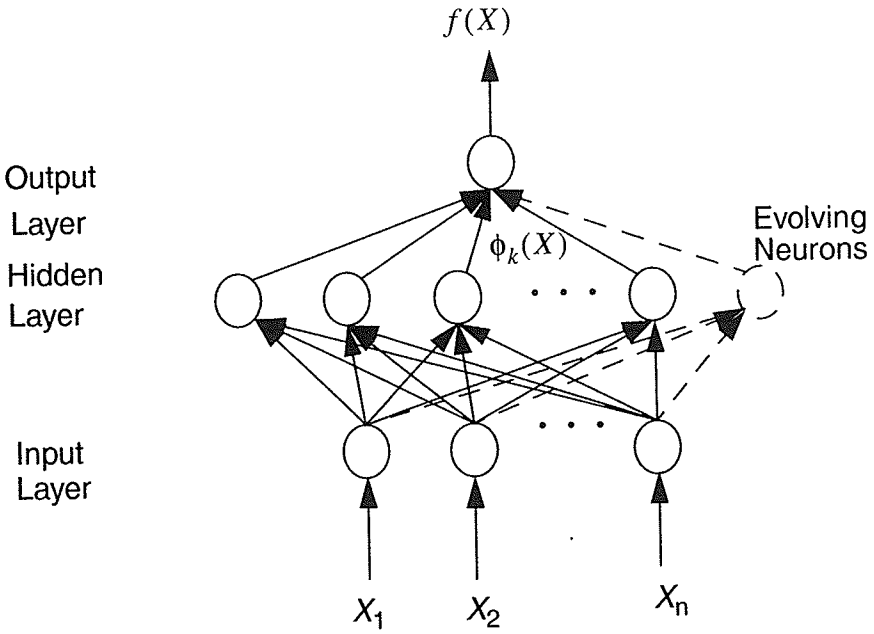

Fig. 5.1. Network architecture of RAN. (The dashed lines show the addition or deletion of hidden units.)

The RAN is a single hidden layer network whose output response to an input pattern is a linear combination of the hidden unit responses, given by

$$f(X) = \alpha_0 + \sum_{k=1}^{K} \alpha_k \phi_k(X) \tag{5.1}$$

where $\phi_k(X)$ are the responses of the hidden units to an input X. The coefficients, α_k, are the weights between the hidden and the output layers, and α_0 is the bias term. The RAN hidden responses are given by

$$\phi_k(X) = \exp\left\{-\frac{1}{\sigma_k^2}\|X - u_k\|^2\right\} \tag{5.2}$$

where u_k is the unit center or mean of the Gaussian, and σ_k is the spread of the neighbourhood or width of the Gaussian. Hence, the u_k is viewed as stored pattern, and the weights of the hidden-output layer, α_k, define the contribution of each hidden unit to a particular output.

The network begins with no hidden units. The first observation (X_0, y_0), where y_0 is the target output, is used in initializing the coefficient $\alpha_0 = y_0$. As observations are received, the network grows by storing some of them by adding new hidden units. The decision to store an observation (X_n, y_n) depends on the novelty of the observation, for which the following two conditions must be met

$$\|X_n - u_{nr}\| > \varepsilon_n \tag{5.3}$$

$$e_n = y_n - f(X_n) > e_{min} \tag{5.4}$$

where u_{nr} is the stored pattern nearest to X_n in the input space, and ε_n, e_{min} are thresholds. The first criterion says that the input must be far away from stored patterns and the second criterion says that the error between the network output and the target must be significant. The value e_{min} is chosen to represent the desired accuracy of the network output. The distance ε_n represents the scale of resolution in the input space.

When a new hidden unit is added to the network, the parameters and weights associated with this unit are assigned as follows

$$\alpha_{k+1} = e_n \tag{5.5}$$

$$u_{k+1} = X_n \tag{5.6}$$

$$\sigma_{k+1} = \kappa \| X_n - u_{nr} \| \tag{5.7}$$

where κ is an overlap factor that determines the overlap of the responses of the hidden units in the input space and its range is (0,1). The value for the width σ_{k+1} is based on a nearest-neighbour heuristic.

When the observation (X_n, y_n) does not satisfy the novelty criteria, the least square error (LMS) algorithm is used to adapt the network parameters $W = [\alpha_0, \ldots, \alpha_k, u_1^T, \ldots, u_k^T]^T$, given by

$$W^{(n)} = W^{(n-1)} + \eta e_n \alpha_n \tag{5.8}$$

where η is the adaptation step size and $\alpha_n = \nabla_W f(X_n)$ is the gradient of the function with respect to the parameter vector W evaluated at $W^{(n-1)}$. Hence,

$$\alpha_n = \left[1, \phi_1(X_n), \ldots, \phi_k(X_n), \phi_1(X_n)\frac{2\alpha_1}{\sigma_1^2}(X_n - u_1)^T, \ldots, \phi_k(X_n)\frac{2\alpha_k}{\sigma_k^2}(X_n - u_k)^T \right]^T \tag{5.9}$$

The RAN begins with $\varepsilon_n = \varepsilon_{max}$, the largest scale of interest, typically the size of the entire input space of nonzero probability density. The distance ε_n is decreased exponentially as

$$\varepsilon_n = max\{\varepsilon_{max}\gamma^n, \varepsilon_{min}\} \tag{5.10}$$

where $0 < \gamma < 1$ is a decay constant. The value for ε_n is decreased until it reaches ε_{min}.

At first, the system creates a coarse representation of the function, then refines the representation by allocating units with smaller and smaller widths. Finally, when the system has

learned the entire function to the desired accuracy and length scale, it stops allocating new units.

Platt showed that the complexity of the RAN was smaller than that of fixed-size networks in achieving a given degree of approximation. The advantages of the RAN are that it learns quickly, accurately, and forms a compact representation.

The RAN is described mathematically as follows:

1) The network output corresponds to the input X is given by

$$f(X) = \alpha_0 + \sum_{k=1}^{K} \alpha_k \phi_k(X)$$

$$\phi_k(X) = \exp\left\{ -\frac{1}{\sigma_k^2} \| X - u_k \| \right\}$$

2) $\varepsilon_n = \varepsilon_{max} \qquad \alpha_0 = y_0$

3) For each observation (X_n, y_n),

$$\varepsilon_n = max\{\varepsilon_{max}\gamma^n, \varepsilon_{min}\}$$

$$e_n = y_n - f(X_n)$$

If

$$e_n > e_{min} \quad \text{and} \quad \| X_n - u_{nr} \| > \varepsilon_n$$

Allocate a new hidden unit with

$$\alpha_{k+1} = e_n$$

$$u_{k+1} = X_n$$

$$\sigma_{k+1} = \kappa \| X_n - u_{nr} \|$$

Else

$$W^{(n)} = W^{(n-1)} + \eta e_n \alpha_n$$

5.4.2 The Minimal RAN (M-RAN) as Enhanced RAN

One drawback of the RAN is that once a hidden unit is created, it can never be removed. Because of this, the RAN could produce networks in which some hidden units, although active initially, may subsequently end up contributing little to the network output. If such inactive hidden units can be detected and removed as learning progresses, then a more compact network topology can be realized. This lead to an enhanced RAN - Minimal RAN (M-RAN) which was proposed in [LuSu97].

5.4.2.1 Basic Growth Strategy

The basic growth strategy used in M-RAN is the same as that of RAN.

5.4.2.2 Pruning Strategy

To remove the hidden units that make little contribution to the network output, the output of each hidden unit has to be examined:

$$o_k = \alpha_k \exp\left(-\frac{\| X - u_k \|^2}{\sigma_k^2} \right) \tag{5.11}$$

If the output of a hidden unit is less than a threshold over a number of M consecutive inputs, then that hidden unit is removed from the network. The output of hidden units are normalized in the pruning criterion since the use of absolute values can cause inconsistency in pruning. The pruning strategy is expressed as follows:

1) For every observation (X_n, y_n), compute the output of all hidden units $o_k^{(n)}$ ($k = 1, \ldots K$), using equation (5-11).

2) Find the largest absolute hidden unit output value $\left\| o_{max}^{(n)} \right\|$. Compute the normalized output value $r_k^{(n)}$, ($k = 1, \ldots, K$), for the hidden units where $r_k^{(n)} = \left\| \frac{o_k^{(n)}}{o_{max}^{(n)}} \right\|$.

3) Remove the hidden units for which the normalized output is less than a threshold δ for M consecutive observations.

5.4.2.3 Sliding Window RMS Criteria for Adding Hidden Neurons

Beside the pruning strategy used in M-RAN, a new criterion is also added to the novelty criterion in M-RAN, which makes a more careful decision on whether a new hidden unit should be added or not. This new criterion uses a root mean square (RMS) value of the output error over a sliding data window before adding a hidden unit. The RMS value of network output error at nth observation e_{rmsn} is given by

$$e_{rmsn} = \sqrt{\frac{\sum_{i=n-(M-1)}^{n} e_i^2}{M}} \tag{5.12}$$

where $e_i = y_i - f(X_i)$. In Eq. 5.12, for each observation, a sliding data window is employed to include a certain number of output errors based on $y_n, y_{n-1}, \ldots, y_{n-(M-1)}$. When a new observation arrives, the data in this window is updated by including the latest data and eliminating the oldest. For example, at the nth observation, the data window includes $e_n, e_{n-1}, \ldots, e_{n-(M-1)}$;

where the *(n+1)*th observation arrives, the data window will contain $e_{n+1}, e_n, \ldots, e_{n-(M-2)}$. The size of the window is determined by the value of M - that is, the number of samples used to calculate the RMS error. Thus, the extra growth criterion to be satisfied is $e_{rmsn} > e_{min}'$, where e_{min}' is a threshold value to be selected.

5.4.3 Modelling and Prediction of Nonstationary Chaotic Signals using RAN

The performance of RAN and M-RAN are tested on a particular chaotic time series created by Mackey-Glass delay-difference equation

$$x(t+1) = (1-b)x(t) + a\frac{x(t-\tau)}{1+x(t-\tau)^{10}} \tag{5.13}$$

for $a = 0.2$, $b = 0.1$, and $\tau = 17$.

The networks are given no information about the generator of the time series, and are asked to predict the future of the time series from the samples of the history. Specifically, the two classes of networks are on-line trained to predict the value at time $T + \Delta T$, from input at time T, $T-6$, $T-12$, and $T-18$. The following parameters were used: $\varepsilon_{max} = 1.0$, $\kappa = 0.87$, $e_{min} = 0.05$, $\eta = 0.02$, $\gamma = 0.999$, and $\varepsilon_{min} = 0.01$. The prediction results are shown in Fig5.2 (a) - (d).

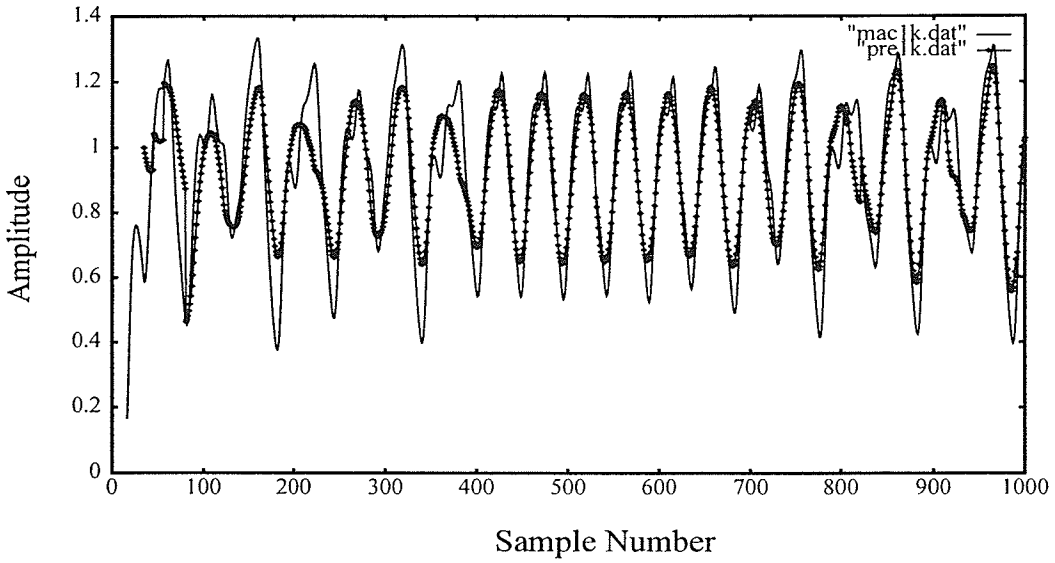

Fig. 5.2 (a) RAN 17-sample ahead prediction during the first 1000 samples period.

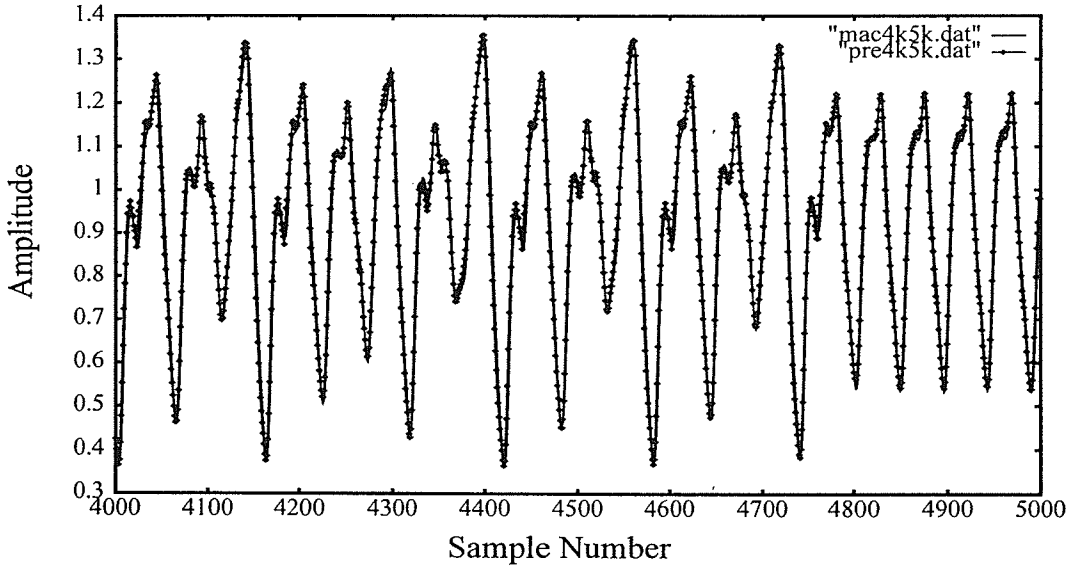

Fig. 5.2 (b) RAN 17-sample ahead prediction during the 4000th-5000th samples period.

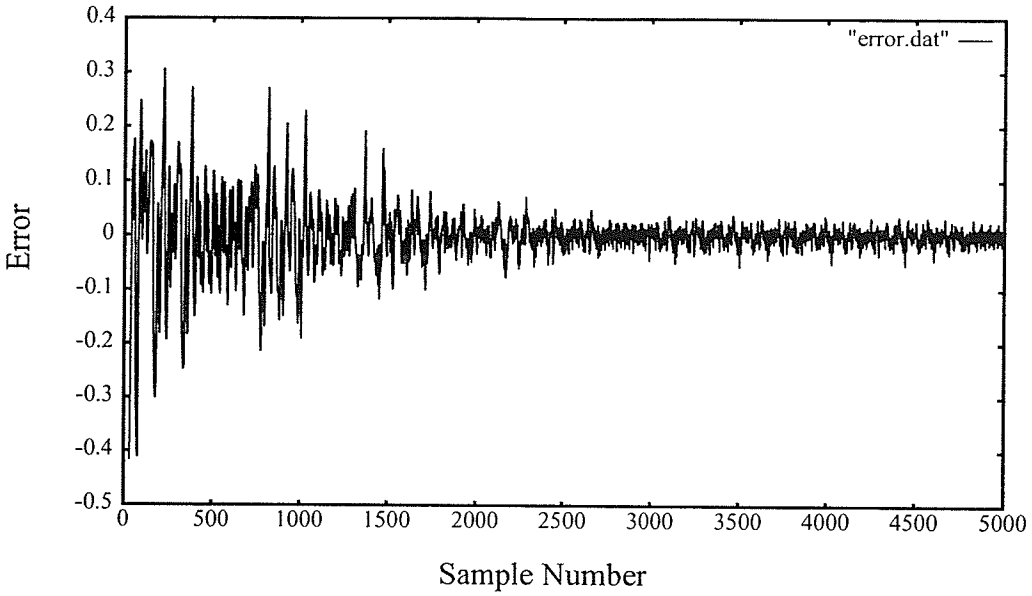

Fig. 5.2 (c) RAN 17-sample ahead prediction absolute errors.

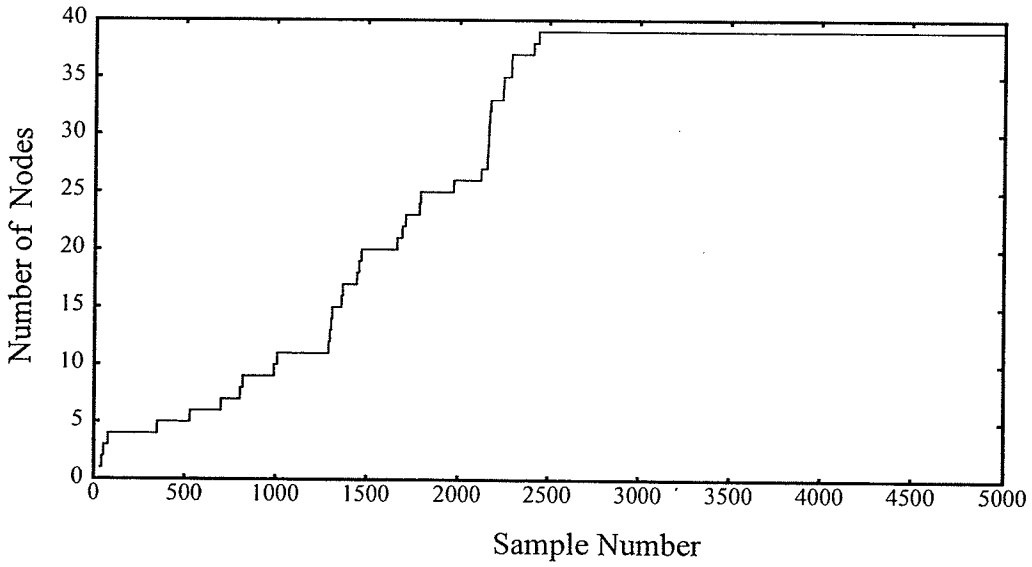

Fig. 5.2 (d) RAN hidden units number profile during learning.

Figures 5.2 (a) and (b) shows the output of the RAN after having learned the Mackey-Glass equation on-line. In the simulations, the network learns to roughly predict the time series quite rapidly. Notice in Fig. 5.2 (a) the sudden jumps in the output of the network, which show that a new unit has been allocated. At the beginning of the learning, the RAN picks up the basic

oscillatory behaviour of the Mackey-Glass equation. As more examples are shown, the network allocates more units and refines its predictions, and eventually the prediction match the actual output very well, as shown in Fig. 5.2 (b).

As shown in Fig. 5.2 (d), the number of hidden units grows and eventually approaches a constant. This demonstrates that the RAN has the ability to adjust its structure incrementally and finally to create a compact model of the underlying dynamic process.

From Fig. 5.2 (d), we can also see that the RAN can not detect those hidden units which are active at the beginning, but have no contributions at the end. So in the final neural network evolved, there are some hidden neurons left and these neurons make no contributions to the accuracy of the prediction, just a waste of the computing resource, memory and CPU processing time. The resulting neural network is not optimal in the sense of the resource and complexity. This has been improved by M-RAN. The performance of M-RAN are shown in Figs. 5.3 (a) to (d).

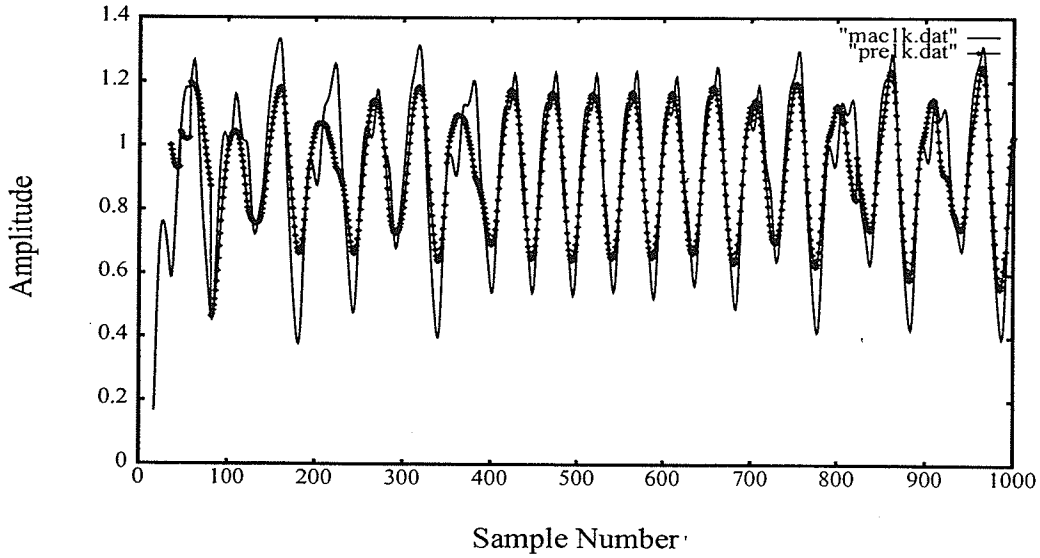

Fig. 5.3(a) M-RAN 17-samples ahead prediction during the first 1000 samples period.

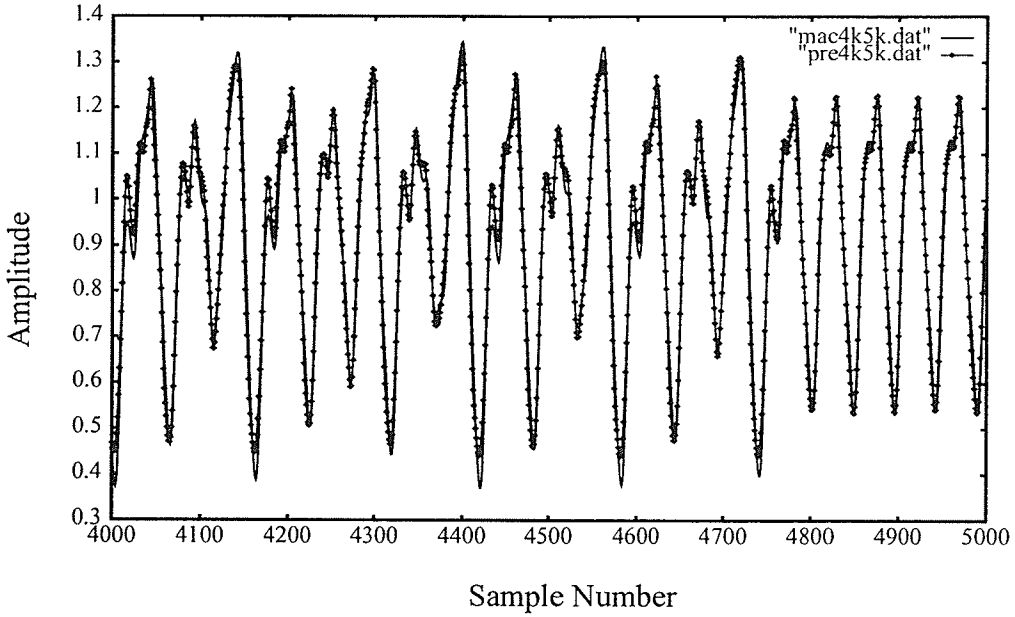

Fig. 5.3 (b) M-RAN 17-samples ahead prediction during the 4000th-5000th samples period.

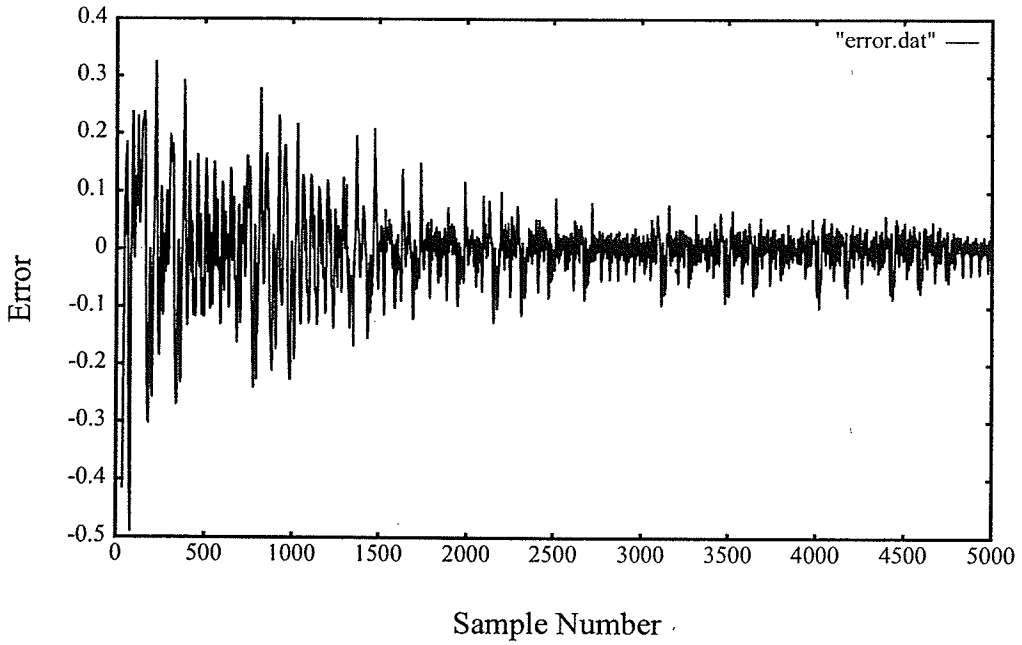

Fig. 5.3 (c) M-RAN prediction errors.

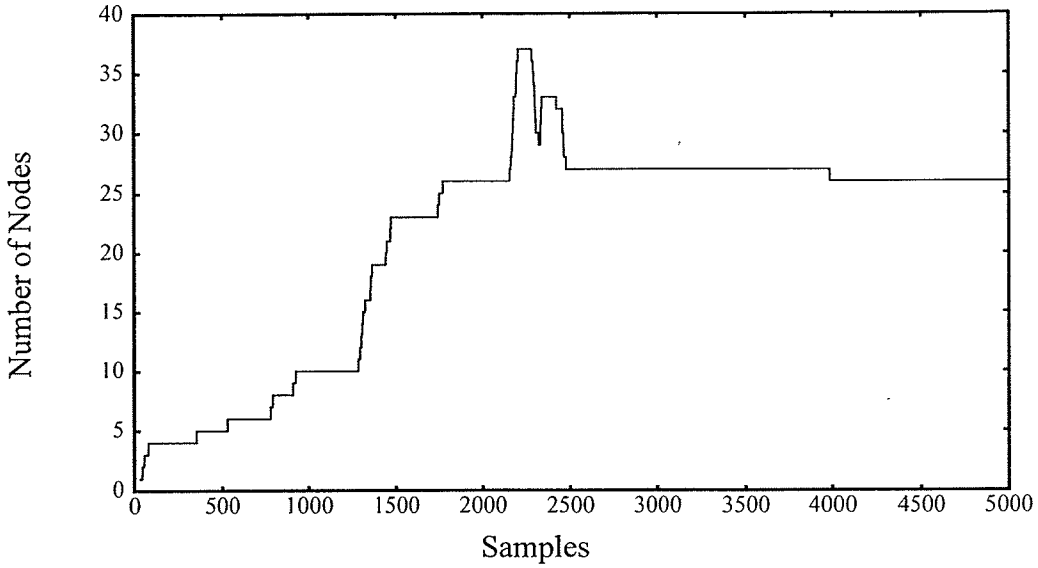

Fig. 5.3 (d) M-RAN hidden units profile during learning.

From Figs. 5.3 (a) to (d), we can see that the M-RAN achieves a comparable predicting accuracy, but with fewer hidden units. The number of hidden units grows at the beginning stage, but decreases later, which indicates that some hidden units which are active at the beginning have become inactive for some period, and have been removed from the networks. This strategy makes the structure of the network more compact comparing to the RAN. This feature of M-RAN makes M-RAN more suitable to model dynamic process, especially to model and predict nonstationary signals. The M-RAN can allocate new hidden units when new behaviour has come out, and delete old and inactive hidden units when these hidden units cannot make any contributions when the underlying dynamic process changes. There are no resource wasted and the structure of the final evolved neural network is optimized.

5.5 Short-Term Power Load Prediction using RAN

5.5.1 Short-Term Power Load Prediction

The forecasting of electricity demand has become one of the major research fields in electrical engineering. The supply industry requires forecasts with lead times that range from the short term (a few minutes, hours, or days ahead) to the long term (up to 20 years ahead). Short-term forecasts, in particular, have become increasingly important since the rise of the competitive energy markets. Many countries have recently privatized and deregulated their power systems, and electricity has turned into a commodity to be sold and bought at market prices. Since the load forecasts play a crucial role in the composition of these prices, they have become vital for the supply industry.

Short-term power load forecasting is also used to provide utility company management with future information about electric load demand in order to assist them in running more economical and reliable day-to-day operations. It is required for unit commitment, energy transfer scheduling, and load dispatch. The development of an accurate, fast and robust short-term load forecasting methodology is of importance to both the electric utility and its customers.

Load forecasting is however a difficult task. First, because the load series is complex and exhibits several levels of seasonality: the load at a given hour is dependent not only on the load at the previous hour, but also on the load at the same hour on the previous day, and on the load at the same hour on the day with the same denomination in the previous week. Secondly, because there are many important exogenous variables that must be considered, specially weather-related variables. It is relatively easy to get forecasts with about 10% *mean absolute percent error* (MAPE);

However, the costs of the error are so high that research that could help reducing it to a few percent points would be amply justified. An often quoted estimation suggests that an increase of 1% in the forecasting error would imply (in 1984) in a £10 million increase in operating costs per year.

Artificial neural networks (ANNs) have proven to be especially suited for load forecasting applications, as they do not rely on any explicit rules or predefined mathematical relationships between the model inputs and outputs. Rather, they attempt to form natural links between the inputs and outputs through a process of self-learning, based on a collection of input and desired output patterns.

The most common, by far, of these neural network structures, is the three-layer, feedforward model, trained by the error backpropagation algorithm. As discussed in the previous chapter, one very important characteristics of this structure, is that its structure is static and fixed, and number of hidden units has to be determined in advance. We have also seen that this structure lacks the ability to track the complexity of the underlying dynamic process, and to achieve an optimal structure on the fly.

In nature, complex dynamic patterns, like the power load profile, are often the result of a relatively simple underlying generating mechanism. As we have seen in the previous section, the M-RAN is suitable for sequential learning and capable to model the dynamic process and predict nonstationary signals. Hence, the M-RAN should be a good candidate for short-term power load forecasting.

5.5.2 Short-Term Power Load Prediction using RAN

Sets of power load data from Manitoba Hydro are used to test the forecasting performance of the M-RAN technique. The forecasting experiments were performed with three different lead time, 1 hour, 24 hours, and 168 hours (1 week) respectively. For 1-hour ahead forecasting, the past power load values at *T*-1, and *T*-24 were chosen as inputs to the RAN, where (*T*-*i*) represents *i* hours before the forecasting time instant *T*. For 24-hour ahead forecasting, the power load at *T*-24, *T*-48, and *T*-96 were chosen. For 168-hour ahead forecasting, the power load values at *T*-168, *T*-172, *T*-196, and *T*-220 were chosen.

To evaluate the M-RAN performance, the most commonly error metric, the *absolute percentage error* (APE) was applied, which is defined by

$$\varepsilon_{APE}{}^{(k)} = 100\frac{\left|l^{(k)} - l^{'(k)}\right|}{l^{(k)}} \qquad (5.14)$$

where $l^{(k)}$ and $l^{'k}$ is the actual and predicted power load at time instance, k, respectively. In order to get more accurate comparing results, the MAPE from 9250th hour to 9500th hour in 1994 is used to evaluate the performance.

Figures 5.4 (a) and (b) show the results of 1-hour ahead forecasting over 250 hours period. The MAPE is 2.4% and the change of hidden units number reflects the dynamic change of network structure.

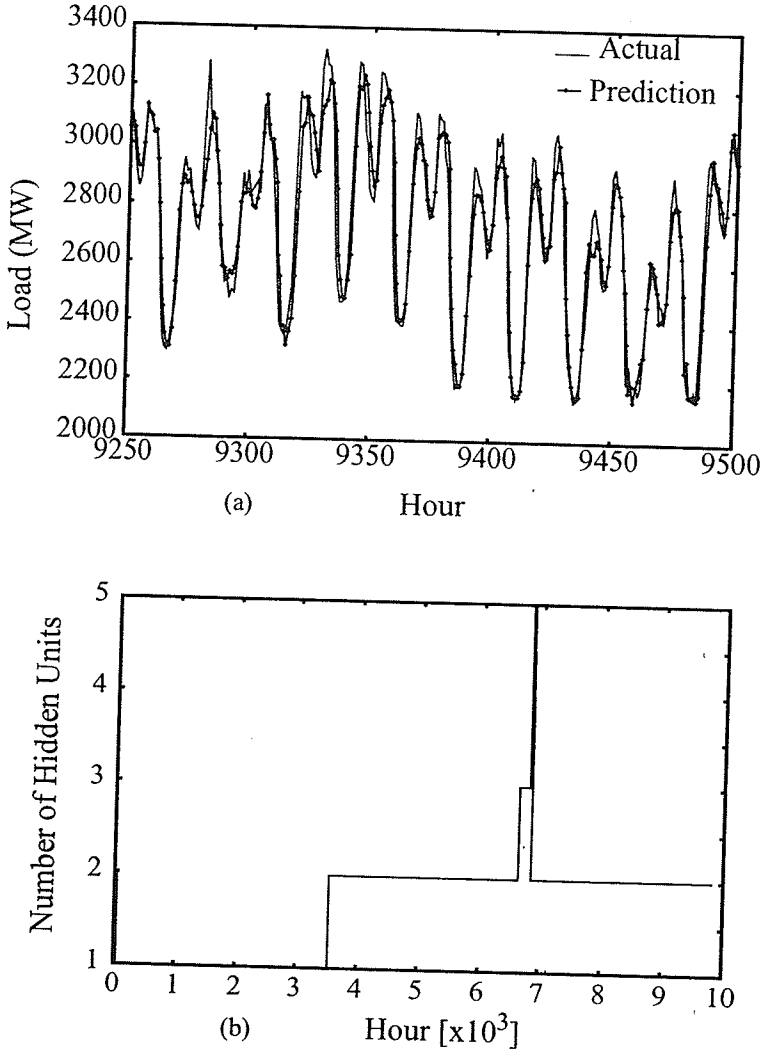

Fig. 5.4 1-hour ahead load prediction, MAPE = 2.4%.

Figure 5.5 (a) and (b) shows the results of 24-hour ahead forecasting over 250 hours period. The MAPE is 2.0% and the change of hidden units number reflects the dynamic change of network structure.

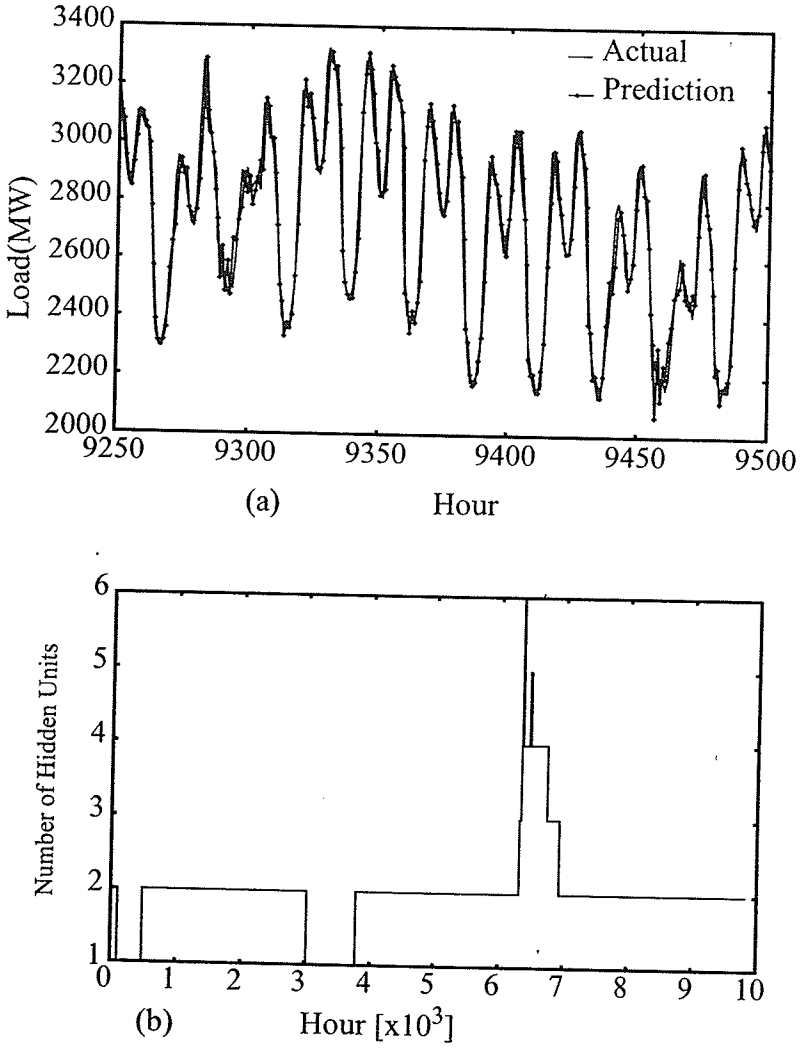

Fig. 5.5 24-hour ahead prediction, MAPE = 2.0%.

Figure 5.6 (a) and (b) shows the results of 168-hour ahead forecasting over 250 hours period. The MAPE is 2.2% and the change of hidden units number reflects the dynamic change of network structure.

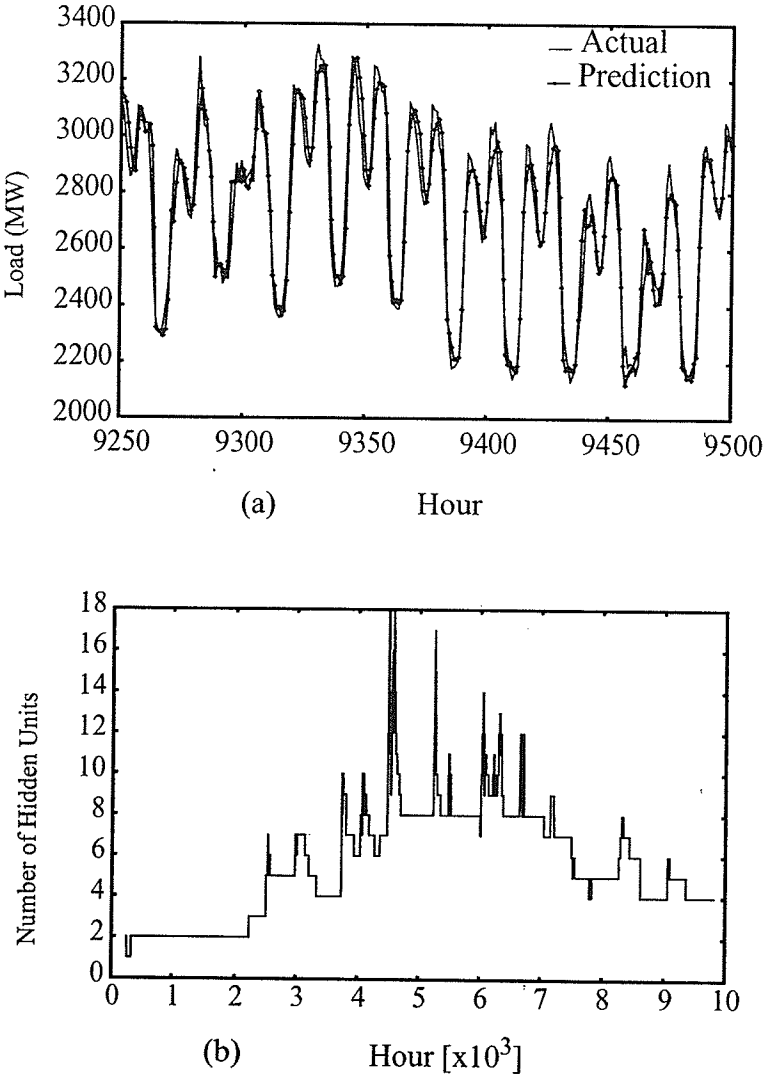

Fig. 5.6 168-hour ahead prediction, MAPE = 2.2%.

From the results of load prediction presented above, it is observed that RAN provides accurate predictions with a small network size. This is mainly due to the non-linear modelling ability of RAN and its superior tracking performance, which comes from RAN's architecture adaptation.

5.5.3 Results Comparison with Other Types of Neural Networks

Although in this thesis, only RAN is explored in short-term power load prediction, the experimental results using other types of neural networks are also collected by searching research papers. The searching results are listed here:

- As reported in [Yass95], a three-layer feed-forward neural network using back-propagation training algorithm achieved MAPE 2.924% for 24 hours ahead prediction. This network has 12 input neurons, 24 hidden neurons, and one output neuron.
- As reported in [VeBo98], a recurrent neural network achieved MAPE 2.0237% for 168 hours ahead prediction. This network has 7 input neurons, 14 hidden neurons, and one output neuron.
- As reported in [DaLa97], a functional link network achieved MAPE 2% for 24 hours ahead prediction, and 2.5% for 168 hours ahead prediction. This network has 31 input neurons, one output neuron.

From the above results, we can see that RAN has comparable or better prediction results with a much smaller structure. One thing worth to mention is in those reports, weather effect has been considered. In our experiment, due to the lack of historical weather data, only historical power load are used as input. We believe that the RAN's performance would be improved further if the weather data are used as part of the input, e.g., the peak day temperature prediction for the predicted day, the lowest temperature prediction, the peak relative humidity prediction, and the lowest relative humidity prediction. Once these weather data are used as part of the input, the RAN should be able to learn the effect of weather on the power load. The prediction accuracy should be improved.

5.6 Summary

The chapter presents a new on-line nonstationary signal prediction technique using the RAN. The RAN has one distinguishing advantage as compared to other neural networks with fixed-size architecture: it can adjust its architecture according to the novelty of data, and therefore be capable of tracking the complexity of data observed. The results of chaotic time series prediction proves that the RAN is capable of modelling and predicting nonstationary signals with fractal characteristics. Furthermore, the power load forecasting results obtained with real load data from Manitoba Hydro reveal the superior performance of the RAN in predicting short-term power load.

CHAPTER VI
CONCLUSIONS AND RECOMMENDATIONS

6.1 Summary

In this thesis, an accurate power system transient modelling and classification system has been developed, and a short-term power load prediction system using architecturally evolvable neural networks has been designed and implemented.

The wavelet transform (WT) has been employed to model different types of power system transients. A time-scale multiresolution representation of transients have been applied and used for transient classification. A probabilistic neural network (PNN) is used as a classifier due to its fast training and solid mathematical foundation. Furthermore, the PNN provides a confidence measure for each classification decision.

6.2 Conclusions

The WT has been demonstrated to be a very useful tool for nonstationary signal analysis and modelling, and its application in power system transients modelling has been shown to be applicable, practical and efficient. The time-scale two dimensional representation of transients extracts not only frequency information but also time information, which is very important for distinguishing faults with different durations. In the thesis, the first four scales of discrete wavelet transform (DWT) coefficients are used for characterizing power transients.

In our application of classifying three types of power transients, the experimental results show that the classification system can achieve 100% correct classification rate for pure data with-

out noise. In a noisy environment, the classification system also achieved over 92% correct classification rate when white Gaussian, pink, and black noises are presented. For each type of noise, the classification system is tested with SNR at 10 and 20 dB, respectively.

Experimental classification results show that the correct classification rate and confidence are higher when DWT is applied in a signal preprocessing stage, thus proving that DWT can extract features from transients, enhance features extracted, and reduce the input dimension.

In our experiment, two similar but different types of transients, single-phase fault cleared in time, and single phase fault not cleared in time are selected to further test the time-scale representation capability of the WT. The experimental results show that the PNN can make classification decision correctly due to the time information extracted by WT together with the frequency information.

The training and testing time of the PNN is about 2 seconds, while a BP network takes about 5 minutes with a lower correct classification rate. This demonstrates that the PNN is a faster and better classifier.

Unlike power transients, which are short-term nonstationary signals, power load is a long-term nonstationary signal. A better modelling methods using architecturally evolvable neural networks, such as the resource allocating network (RAN) is applied in the thesis for power load prediction.

Since hidden neurons can be added or removed in the RAN dynamically, its computing structure can be adjusted on-line, thus achieving a complexity matching between the underlying physical process and the system model.

The RAN has been studied in this thesis to predict a nonstationary chaotic signal, the Mackey-Glass signal. The experimental results show that, after learning, RAN finally becomes the model of the underlying chaotic system with a matching number of hidden neurons, and it can make accurate prediction for the nonstationary signals generated by the underlying chaotic system.

The experimental results of RAN in power system short-term load prediction have shown that RAN can achieve MAPE 2.4% for 1-hour ahead load prediction, 2.0% for 24-hour ahead load prediction, and 2.2% for 168-hour ahead load prediction. These results are comparable or better than the results achieved by other types of neural networks with fixed architectures.

From the power load prediction results, we can see that power load can be looked upon as nonstationary signals, and the methods and techniques applicable for studying nonstationary signals can be and should be used for power load prediction.

To obtain a better understanding of nonstationary signal modelling, classification, and prediction is another underlying driving force behind this thesis. From this thesis, we have seen that wavelets are suitable for nonstationary signal modelling through time-scale multi resolution representation. The WT decompositions at different scales represent the subspaces of the phase space of the underlying physical process. The RAN is also a suitable tool for modelling nonstationary signals, with the capability to find a matched computing structure. The neurons in the hidden layer consist of a neuron representation of the nonstationary signal, and each hidden neuron represents a subspace of the phase space of the underlying physical process, with different scaling parameters fit into the different subspaces. From this thesis, we can see some similarities between wavelet modelling and RAN modelling of nonstationary signals.

6.3 Contributions

This thesis has made the following contributions:

a) Wavelets have been successfully applied to power transients modelling and analysis, and demonstrated that wavelets are suitable tool for power transients modelling;

b) A standard PNN has been implemented for power transients classification;

c) Modelling of nonstationary signals with fractal characteristics using RAN has been conducted; and

d) The short-term power load prediction using RAN has been proposed and implemented.

6.4 Recommendations

Based on the research done, the following theoretical and practical work is recommended for the future:

a) Setup a large database of transients from real power systems for the transient classification system;

b) In the transient classification system, apply feature extraction methods, like multifractal measurement, to extract features and reduce the input variable size;

c) In the transient classification system, compare the Daub4 and other types of wavelets;

d) Use multi-sigma PNN for power transient classification;

e) In the short-term power load prediction system, add weather data to the input of the system to incorporate the weather effect;

f) Investigate and calculate the fractal dimension of the power load;

g) Investigate the combination usage of wavelets and the RAN in the power load prediction system; and

h) Find the theoretical relationship between the RAN and the WT in modelling of nonstationary signals.

REFERENCES

[BiKr95] R. P. Bingham, D. Kreiss, and S. Santoso, "Advances in data reduction techniques for power quality instrumentation," *Proc. 3rd European Power Quality Conf. Power Quality'95*, Bremen, Germany, Nov. 1995.

[Caco66] T. Cacoullos, "Estimation of a multivariate density," *Annals of the Institute of Statistical Mathematics (Tokyo),* 18(2), pp. 179-189, 1966.

[ChBr96] O. Chaari, F. Brouaye, etc. "Wavelets: a new tool for the resonant grouned power distribution systems relaying," *IEEE Trans. Power Delivery,* Vol. 11, No. 3, pp. 1301-1308, July 1996.

[CoJo98] D.V. Coury and D.C. Jorge, "Artificial neural network approach to distance protection of transmission lines," *IEEE Trans. Power Delivery,* Vol. 13, No. 1, pp. 102-108, Jan. 1998.

[DaKi98] R. Dansereau and W. Kinsner, "Progressive transmission of images using wavelets: Evaluation using the Renyi generalized entropy", *Proc. IEEE Canadian Conf. on Elec. and Comp. Eng.* CCECE'98, May 1998.

[DaLi97] P. K. Dash, A. C. Liew, etc., "A real-time short-term load forcasting system using functional link network," *IEEE Trans. Power Systems*, Vol. 12, No. 2, pp. 675-681, May. 1997.

[Daub92] I. Daubechies, *Ten Lectures on Wavelets*, Philadelphia, Pennsylvania, Capital City Press, 1992.

[Dono95] D. L. Donoho, "De-noising by soft-thresholding," IEEE Trans. Inf. Theory, Vol. 41, pp. 613-627, May 1995.

[Hayk94] S. Haykin, *Neural Networks - a comprehensive foundation*, Prentice Hall, pp.1, 1994.

[HeGa97] G. T. Heydt and A. W. Galli, "Transient power quality problems analyzed using wavelets," *IEEE Trans. Power Delivery,* Vol. 12, No. 2, pp. 908-915, Apr. 1997.

[HuHu99] S. Huang, C. Hsieh, and C. Huang, "Application of Morlet wavelets to supervise power system disturbances," *IEEE Trans. Power Delivery*, Vol. 14, No.1, pp. 235-242, 1999.

[HuYa01] C. M. Huang and H. T. Yang, "Evolving wavelet-based networks for short-term load forecasting," *IEE Proc. Gener. Transm. Distrib.*, Vol. 148, No. 3, pp. 222-228, May 2001

[KaMo00] M. Karimi, H. Mokhtari, and M. R. Iravani, "Wavelet based on-line disturbance detectin for power quality applications," *IEEE Trans. Power Delivery*, Vol. 15, No. 4, pp. 1212-1220, Oct. 2000.

[KaSo92] N. Kandil, V.K. Sood, etc., "Fault identification in an AC-DC transmission system using neural networks," *IEEE Trans. Power Systems*, Vol. 7, No. 2, pp. 812-819, May. 1992.

[KiLa93] W. Kinsner and A. Langi, "Speech and image signal compression with wavelets," *Proc. IEEE Communications, Computer & Power Conf.*, WESCANEX'93 (Saskatoon: SK: May 17-18, 1993), IEEE Cat. No. 93CH3317-5;pp. 368-375.

[Kins95] W. Kinsner, *Fractal and chaos engineering*, Lecture Notes, Dept. Electrical and Computer Eng., University of Manitoba, 1995, Ch. 4, pp. 62-66.

[Koho90] T. Kohonen, "The self-organizing map," *Proc. of IEEE*, Vol. 78, No. 9, pp. 1464-1480, 1990.

[LiYa01] W. Lin, C. Yang, etc., "A fault classification method by RBF neural network with OSL learning procedure," *IEEE Trans. Power Delivery*, Vol. 16, No. 4, pp. 473-477, Oct. 2001.

[LaKi95] A.Langi and W.Kinsner, "Consonant characterization using correlation fractal dimension for speech recognition", *Proc. on IEEE WESCANEX'95,* Winnipeg, Canada, IEEE 95CH3581-6, pp. 208-213, 15-16 May 1995.

[Mall89] S. Mallat, "A theory for multiresolution signal decomposition," *IEEE Trans. Pattern Anal. Machine Intell.*, pp. 674-693, Nov. 1989.

[Mast95] T. Masters, *Advanced algorithms for neural networks - A C++ sourcebook*, John Wiley & Sons, Inc. 1995.

[MoGr62] A.M. Mood and F.A.Graybill, *Introduction to the theory of statistics*, New York: Macmillan, 1962.

[MoKi96] F. Mo and W. Kinsner, "Review of diagnostic techniques of power system transients," *Proc. IEEE 1996 Can. Conf. Electrical and Computer Eng.*, CCECE'96 (Calgary, AB; May 26-29, 1996), ISBN 0-7803-3143-5; pp. 323-326.

[MoKi97] F. Mo and W. Kinsner, "Wavelet modelling of transients in power systems," *Proc. IEEE Communications, Computer & Power Conf., WESCANEX'97* (Winnipeg: MB: May 22-23, 1997), IEEE Cat. No. 97CH36117-5; pp. 132-137.

[MoKi98] F. Mo and W. Kinsner, "Probabilistic neural networks for power line fault classification," *Proc. IEEE 1998 Can. Conf. Elec. Comp. Eng., CCECE'98* (Waterloo, ON; May 26-28, 1998).

[MoKi98] F. Mo and W. Kinsner, "Predicting and modelling of nonstationary temporary signals with fractal characteristics," *Proc. IEEE 1998 Can. Conf. Elec. Comp. Eng., CCECE'98* (Waterloo, May 26-28, 1998).

[PaSa98] S. Pandey and L. Satish, "Multiresolution signal decomposition: a new tool for fault detection in power transformers during impulse tests," *IEEE Trans. Power Delivery*, Vol. 13, No. 4, pp. 1194-1200, Oct. 1998.

[Parz62] E. Parzen, "On estimation of a probability density function and mode," *Annals of Mathematical Statistics*, Vol. 33, pp. 1065-1076, 1962.

[Plat91] J. Platt, "A resource-allocating network for function interpolation", *Neural Computation*, vol. 3, pp. 213-225, 1991.

[PSCAD] "Power System Simulation Software Installation & Administration Manual", *PSCAD/RTDSTM Manual Set*, RTDS Technologies, Manitoba HVDC Research Centre.

[Shap93] J. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. Signal Processing*, Vol. 41, pp. 3445-3462, Dec. 1993.

[SnPo00] S. Santoso, E. Powers, etc., "Power quality disturbance waveform recognition using wavelet-based neural classifier - part 1: theoretical foundation," *IEEE Trans. Power Delevery*, Vol. 15, No. 1, pp. 222-228, Jan. 2000.

[SiMo00] A. P. Silva and L. S. Moulin, "Confidence intervals for neural network based short-term load forecasting," *IEEE Trans. Power Delivery*, Vol. 15, No. 4, pp. 1191-1196, Nov. 2000.

[Spec88] D. F. Specht, "Probabilistic neural networks for classification, mapping, or associative memory," *Proc. IEEE Int. Conf. Neural Networks*, San Diego, CA, vol. 1, pp. 525-532, July 1988.

[Spec90] D. F. Specht, "Probabilistic neural networks," *Neural Networks*, Vol. 3, pp. 109-118, 1990.

[Yass95] Y.A.Rashid, "Short-term electrical load forecasting using neural network models", *MS thesis,* Wichita, KS: Wichita State University, 1995.

[YiSu97] L. Yingwei, N. Sundararajan, etc., "A sequential learning scheme for function approximation using minimal radial basis function neural networks," *Neural Computation*, Vol. 9, pp. 461- 476, 1997.

[VeBo98] J. Vermaak and E.C. Botha, "Recurrent neural networks for short-term load forecasting," *IEEE Trans. Power Systems*, Vol. 13, No. 1, pp. 126-132, Feb. 1998.

APPENDIX A

PARAMETER SETTINGS

A-1 Parameter Settings for Transient Classification System

Transient File Parameters
- Sampling Rate: 1,024 Hz
- Transient File Size : 18,282
- Transient Size: 64
- Number of Transient Classes: 3
- Number of Transients: 288

Parameters in Wavelet Analysis
- Wavelet Levels: 4

Nueral Network Training Parameters
- Sigma Threshold : 0.1

A-2 Parameter Settings for Short-Term Power Load Prediction System

- Maximum Scale: 2.0
- Mimimum Scale: 0.03
- Mimimum Error: 0.01
- Learning Rate: 0.05
- Error Threshold: 0.01

APPENDIX B

SOURCE CODE IN THE THESIS

Table B-1: List of source files

```
//*******************************************************
// rad.h
//
// Description:  This is the header file for the class which encapsulates
//               the resource allocation networks (RAN)
//
// Programmer:  Fan Mo
//              Dept. of Electrical & Computer Engineering
//              University of Manitoba
//              Winnipeg, Manitoba
//
// Last Update:  Aug. 6, 2002
//*******************************************************
#ifndef _RAD_H
#define _RAD_H

#define DIMENSION 4

// struct nodeData;
// the struct defination of neual node data
struct nodeData {
   int position;
   double center[DIMENSION];
   double weight;
   double width;
   double response;
};

struct neuralNode;                    // the struct defination of neural node
typedef neuralNode* ptrType;          //pointer to neural node

class networkClass
{
public:
    // constructor and destructor:
    networkClass();
    networkClass(int size, ptrType head, double gama, double learningRate, double scaleMax,
                 double scaleMin, double errorMin);
    networkClass(const networkClass& L);
    ~networkClass();

   // network operations:
   int NetworkIsEmpty();
   int NetworkLength();
   void NeuralNodeInsert(int NewPosition, const struct nodeData &NewNodeData, int& Success);

   void NeuralNodeDelete(int Position, int& Success);
   void NeuralNodeRetrieve(int Position, struct nodeData& NodeRecord, int& Success);

   double EuclidDistance(double * Vect1, double * Vect2);
   ptrType  FindNearestCenter(double * input, int& Position);

   void Start();
   void AfterNthInput(int n, double * input, double DesiredValue);
```

```
    void UpdateWeightCenter(double * input);

    int Condition1();
    int Condition2(double * input);

    int AllocateNewNode(double * input);

    double & SetBias(double bias);
    double & SetGama(double gama);
    double & SetKai(double kai);
    double & SetLearningRate(double learningRate);
    double & SetScaleMax(double scaleMax);
    double & SetScaleMin(double scaleMin);
    double & SetErrorMin(double errorMin);

    double & GetBias() {return Bias;}
    double & GetOutput() {return Output;}
    double & GetPresentError() {return PresentError;}
    double & GetPresentScaleError() {return PresentScaleError;}

    void TotalOutput(double * input);
    void OutputHiddenNode(double * input);

 private:
    ptrType PtrTo(int Position);

    void CalculatePresentError(double DesiredValue);
    void CalculatePresentScaleError(int n);

    double Bias;
    double Output;
    double PresentError;
    double PresentScaleError;

    int Size;
    double Gama;
    double Kai;
    double LearningRate;
    double ScaleMax;
    double ScaleMin;
    double ErrorMin;
    ptrType Head;
};

void ErrorStop(char * Sentence);

#endif
```

```
//*******************************************************
// rad.cc
//
// Description:  This is the implementation file of the resource
//               allocation networks (RAN)
//
// Programmer:  Fan Mo
//              Dept. of Electrical & Computer Engineering
//              University of Manitoba
//              Winnipeg, Manitoba
//
// Last Update:  Aug. 6, 2002
//*******************************************************
#include "rad.h"
#include <stddef.h>          // for NULL
#include <assert.h>          // for assert()
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <math.h>

void ErrorStop(char * Sentence)
{
  printf("%s\n", Sentence);
  exit(-1);
}

// the struct definition of neural node
struct neuralNode {
  nodeData   DataSet;
  ptrType    Next;
};

// Default constructor
networkClass :: networkClass() : Size(0), Head(NULL),Gama(0.9988), LearningRate(0.01), ScaleMax(10.0), Scale-
Min(0.1), ErrorMin(0.0001) {}

// Second constructor
networkClass :: networkClass(int size, ptrType head, double gama, double learningRate, double scaleMax, double
scaleMin, double errorMin)
{
  Size = size;
  Head = head;
  Gama = gama;
  LearningRate = learningRate;
  ScaleMax = scaleMax;
  ScaleMin = scaleMin;
  ErrorMin = errorMin;
}

// Copy constructor
networkClass :: networkClass(const networkClass& L):
  Size(L.Size),
```

```
    Gama(L.Gama),
    LearningRate(L.LearningRate),
    ScaleMax(L.ScaleMax),
    ScaleMin(L.ScaleMin),
    ErrorMin(L.ErrorMin)
{
    int i;

    if (L.Head == NULL)
    {
      Head = NULL;          // original network is empty
    }
    else
    {
      // copy first node
      Head = new neuralNode;
      assert(Head != NULL);       //check allocation

      // copy DataSet
      Head->DataSet.position = L.Head->DataSet.position;
      for( i = 0; i < DIMENSION; i++)
        Head->DataSet.center[i] = L.Head->DataSet.center[i];

      Head->DataSet.weight   = L.Head->DataSet.weight;
      Head->DataSet.width    = L.Head->DataSet.width;
      Head->DataSet.response = L.Head->DataSet.response;

      // copy rest of network
      ptrType NewPrev = Head;  //new network pointer
      for(ptrType OrigCur = L.Head->Next;
          OrigCur != NULL;
          OrigCur = OrigCur->Next)
      {
        NewPrev->Next = new neuralNode;
        assert(NewPrev->Next != NULL);
        NewPrev = NewPrev->Next;

        // copyy DataSet
        NewPrev->DataSet.position = OrigCur->DataSet.position;
        for( i = 0; i < DIMENSION; i++)
          NewPrev->DataSet.center[i] = OrigCur->DataSet.center[i];

          NewPrev->DataSet.weight = OrigCur->DataSet.weight;
          NewPrev->DataSet.width  = OrigCur->DataSet.width;
          NewPrev->DataSet.response = OrigCur->DataSet.response;
          NewPrev->Next = NULL;
      }

networkClass::~networkClass()
{
  int Success;

  // delete the network
  while(!NetworkIsEmpty())
```

```
    {
      NeuralNodeDelete(1, Success);
    }
  }

double & networkClass::SetBias(double bias)
{
  Bias = bias;
  return Bias;
}

double & networkClass::SetGama(double gama)
{
  Gama = gama;
  return Gama;
}

double & networkClass::SetKai(double kai)
{
  Kai = kai;
  return Kai;
}

double & networkClass::SetLearningRate(double learningRate)
{
  LearningRate = learningRate;
  return LearningRate;
}

double & networkClass::SetScaleMax(double scaleMax)
{
  ScaleMax = scaleMax;
  return ScaleMax;
}

double & networkClass::SetScaleMin(double scaleMin)
{
  ScaleMin = scaleMin;
  return ScaleMin;
}

double & networkClass::SetErrorMin(double errorMin)
{
  ErrorMin = errorMin;
  return ErrorMin;
}

int networkClass::NetworkIsEmpty()
{
  return (Size == 0);
}
```

```
int networkClass::NetworkLength()
{
   return Size;

}

ptrType networkClass :: PtrTo(int Position)
{
   if((Position < 1) || (Position > NetworkLength()))
   {
      return NULL;
   }
   else
   {
      ptrType Trav = Head;
      for(int Skip = 1; Skip < Position; Skip++)
        Trav = Trav->Next;

      return Trav;
   }
}

void networkClass :: CalculatePresentError(double DesiredValue)
{
   PresentError = DesiredValue - Output;
}

void networkClass:: CalculatePresentScaleError(int n)
{
   double max = ScaleMax * pow(Gama, (double)n);
   if (max < ScaleMin)
      max = ScaleMin;

   PresentScaleError = max;
}

void networkClass::NeuralNodeRetrieve(int Position, struct nodeData& NodeRecord, int& Success)
{
   ptrType Cur;

   Success = ((Position >= 1) && (Position <= NetworkLength()));
   if(Success)
   {
      Cur = PtrTo(Position);
   }

   NodeRecord.position = Cur->DataSet.position;
   for(int i = 0; i < DIMENSION; i++)
      NodeRecord.center[i] = Cur->DataSet.center[i];

   NodeRecord.weight   = Cur->DataSet.weight;
   NodeRecord.width    = Cur->DataSet.width;
   NodeRecord.response = Cur->DataSet.response;
}
```

```
void networkClass::NeuralNodeInsert(int NewPosition, const struct nodeData &NewNodeData, int& Success)
{
  int NewLength = NetworkLength() + 1;
  printf("Insert a new node at position = %d\n", NewNodeData.position);
  Success = ((NewPosition >= 1) && (NewPosition <= NewLength));

  if(Success)
  {
    Size = NewLength;

    // creat new node and place NewNode in it
    ptrType NewPtr = new neuralNode;
    Success = (NewPtr != NULL);
    if(Success)
    {
      NewPtr->DataSet.position = NewNodeData.position;
      for( int j = 0; j < DIMENSION; j++)
        NewPtr->DataSet.center[j] = NewNodeData.center[j];

      NewPtr->DataSet.weight   = NewNodeData.weight;
      NewPtr->DataSet.width    = NewNodeData.width;
      NewPtr->DataSet.response = NewNodeData.response;

      if(NewPosition == 1)
      {
        // insert new node at beginning of network
        NewPtr->Next = Head;
        Head = NewPtr;
      }
      else
      {
        ptrType Prev  = PtrTo(NewPosition - 1);

        // insert new node after node to which Prev points
        NewPtr->Next = Prev->Next;
        Prev->Next   = NewPtr;
      }
    }
  }
}

void networkClass::NeuralNodeDelete(int Position, int& Success)
{
  ptrType Cur;

  Success = ((Position >= 1) && (Position <= NetworkLength()));
  if(Success)
  {
    --Size;
    if(Position == 1)
    {
      // delete the first node from the network
      Cur = Head;
      Head = Head->Next;
```

```
    }
    else
    {
      ptrType Prev = PtrTo(Position - 1);
      Cur = Prev->Next;
      Prev->Next = Cur->Next;
    }

    // return node to system
    Cur->Next = NULL;
    delete Cur;
    Cur = NULL;
  }
}

ptrType networkClass::FindNearestCenter(double * input, int& Position)
{
  double NearestDistance = 100000000.0;
  double Distance[100];
  ptrType ptr, ptr1;

  int Success;
  for( int i = 0; i < NetworkLength(); i++)
  {
    ptr = PtrTo(i+1);
    Distance[i+1] = EuclidDistance(input, ptr->DataSet.center);

    printf("distance[%d] = %lf ", i+1, Distance[i+1]);
    if(NearestDistance > Distance[i+1])
    {
      NearestDistance = Distance[i+1];
      ptr1 = ptr;
      Position = ptr->DataSet.position;
    }
  }

  //printf("\n the nearest distance is %lf between node %d\n", NearestDistance, //Position);
  return ptr1;
}

// Calculate the outputs of the hidden nodes
void networkClass::OutputHiddenNode(double * input)
{
  double temp;
  ptrType ptr;

  printf("In ***OutputHiddenNode***, NetworkLength = %d\n", NetworkLength());
  for(int i = 0; i < NetworkLength(); i++)
  {
    ptr = PtrTo(i+1);

    printf("ptr->DataSet.center:\n");
    for(int j = 0; j < DIMENSION; j++)
      printf("%lf   ", ptr->DataSet.center[j]);
```

```
        temp = EuclidDistance(input, ptr->DataSet.center);
        temp = -temp * temp/(ptr->DataSet.width * ptr->DataSet.width);
        ptr->DataSet.response = exp(temp);
    }
}

// Calculate the TotalOutput
void networkClass::TotalOutput(double * input)
{
    ptrType ptr;
    Output = Bias;
    OutputHiddenNode(input);

    for(int i = 0; i < NetworkLength(); i++)
    {
        ptr = PtrTo(i+1);
        Output += ptr->DataSet.weight * ptr->DataSet.response;
    }
}

// Update weight, the error information has been stored in the networks
void networkClass:: UpdateWeightCenter(double * input)
{
    ptrType ptr;
    double temp;

    Bias = Bias + LearningRate * PresentError;
    for( int i = 0; i < NetworkLength(); i++)
    {
        ptr = PtrTo(i+1);
        ptr->DataSet.weight +=LearningRate * PresentError * ptr->DataSet.response;

        for( int j = 0; j < DIMENSION; j++)
        {
            temp = ptr->DataSet.response * 2 *
            (ptr->DataSet.weight)/(ptr->DataSet.width *
            ptr->DataSet.width)*(input[j] - ptr->DataSet.center[j]);

            ptr->DataSet.center[j] += LearningRate * PresentError * temp;
        }

    }
}

// If PresentError satisfied ?
int networkClass:: Condition1()
{
    return (fabs(PresentError) > ErrorMin);
}
```

```
// If PresentScaleError satisfied ?
int networkClass:: Condition2(double * input)
{
   ptrType ptr;
   int Position;
   double distance;

   ptr = FindNearestCenter(input, Position);
   //printf("ptr->DataSet.width = %lf\n", ptr->DataSet.width);
   distance = EuclidDistance(input, ptr->DataSet.center);

   return (distance > PresentScaleError);
}

double networkClass:: EuclidDistance(double *Vector1, double *Vector2)
{
   double distance = 0.0;

   for(int i = 0; i < DIMENSION; i++)
      distance += (Vector1[i]-Vector2[i])*(Vector1[i]-Vector2[i]);

   return (sqrt(distance));
}

// if condition 1 and condition 2 satisfied allocate new node]
int networkClass:: AllocateNewNode(double * input)
{
   int Success;
   nodeData  NewNodeData;
   ptrType    ptr;

   printf(" Allocating a new node...\n");
   NewNodeData.position = 1;
   ptr = FindNearestCenter(input, Success);

   NewNodeData.width  = Kai * EuclidDistance(input, ptr->DataSet.center);
   NewNodeData.weight = PresentError;

   for(int j = 0; j < DIMENSION; j++)
   {
      NewNodeData.center[j] = input[j];
   }
   NeuralNodeInsert(1, NewNodeData, Success);
   if(Success)
     return (Success);
   else
    return 0;
}



void networkClass::Start()
{
   int Success;
```

```
    nodeData node;

    node.position = 1;
    for(int i = 0; i < DIMENSION; i++)
    {
       node.center[i] = 0.0;
    }

    node.weight = 0.0;
    node.width = 10.0;
    node.response = 0.0;

    NeuralNodeInsert(1, node, Success);
}



// after each input, calculate the outputs of the hidden nodes,
// total output, PresentError, and PresentScaleError
void networkClass::AfterNthInput(int n, double * input, double DesiredValue)
{
    OutputHiddenNode(input);
    TotalOutput(input);
    CalculatePresentError(DesiredValue);
    CalculatePresentScaleError(n);
}
```

```
//*********************************************************
// train.cc
//
// Description:  This is the main driver to use the RAN to predict
//               short-term power load.
//
//
// Programmer:  Fan Mo
//              Dept. of Electrical & Computer Engineering
//              University of Manitoba
//              Winnipeg, Manitoba
//
//
// Last Update:  Aug. 6, 2002
//*********************************************************
#include "rad.h"
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <iostream.h>


#define Ahead 1
#define TotalSample 20000

void main()
{
  int Success, temp;
  FILE *fp0, *fp1, *fp2, *fp3, *fp4;
  double input[DIMENSION], s[TotalSample];
  double desiredValue;

  if((fp0=fopen("hour.dat", "r"))==NULL)
  {
    printf("Can't open this file for reading!\n");
    exit(-1);
  }

  if((fp1=fopen("hourtraining.dat", "w+"))==NULL)
  {
    printf("Cannot open this file!\n");
    exit(-1);
  }

  if((fp2 = fopen("pred.dat", "w+"))==NULL)
  {
    printf("Cannot write to this file!\n");
    exit(-1);
  }

  if((fp3 = fopen("nodenumber.dat", "w+"))==NULL)
  {
    printf("Cannot write to this file!\n");
    exit(-1);
```

```
}

if((fp4 = fopen("error.dat", "w+"))==NULL)
{
   printf("Cannot write to this file!\n");
   exit(-1);
}

int Counter1 = 0;
while(!feof(fp0))
{
   fscanf(fp0, "%d %lf", &temp, &s[Counter1]);
   Counter1++;
}

for(int i1 = Ahead+24; i1 < Counter1; i1++)
{
   fprintf(fp1, "%lf %lf %lf\n", s[i1-Ahead],s[i1-24],s[i1]);
}
rewind(fp1);

networkClass N;
N.Start();

N.SetScaleMax(1.0);
N.SetScaleMin(0.01);
N.SetGama(0.999);
N.SetErrorMin(0.2);  // Condition1
N.SetKai(0.999);     // Condition 2
N.SetLearningRate(0.2);
N.SetErrorThreshold(0.05);  //Condition3
N.SetBias(1.0);
N.SetDelta(0.01);

int counter = 0;

while(!feof(fp1))
{
   //printf(" Read the %dth samples!\n", counter+1);
   for(int j = 0; j < DIMENSION ; j++)
   {
      fscanf(fp1, "%lf", &input[j]);
      //printf("%lf    ", input[j]);
   }

   fscanf(fp1, "%lf", &desiredValue);
   //printf("     %lf\n", desiredValue);

   counter++;
   //printf("%d\n", counter);
   N.AfterNthInput(counter, input,  desiredValue);

   int Success1 = N.Condition1(desiredValue);
   //printf("Success1 = %d\n", Success1);
```

```
        int Success2 = N.Condition2(input);
        //printf("Success2 = %d\n", Success2);
        int Success3 = N.Condition3(desiredValue);

        //printf("Success3 = %d\n", Success3);
        if(Success1 && Success2 && Success3)
        {
          N.AllocateNewNode(input);
        }
        else
        {
          N.UpdateWeightCenter(input);
          //printf("up data weight\n");
        }

        //printf("number of nodes = %d\n", N.NetworkLength());
        //printf("  PresentError = %lf,   PresentScaleError = %lf,
        //Output = %lf,  DesiredValue = %lf,  number of nodes = %d\n", //N.GetPresentError(), N.GetPresentScaleEr-
ror(), N.GetOutput(), desiredValue, //N.NetworkLength());

        fprintf(fp2, "%d  %lf\n", counter+Ahead+24,  N.GetOutput());
        fprintf(fp3, "%d    %d\n", counter+Ahead+23, N.NetworkLength());
        fprintf(fp4, "%d   %lf\n", counter+Ahead+23, fabs(N.GetPresentError()));

        N.FindBad();
        N.DeleteBad();
    }

    printf("number of nodes = %d\n", N.NetworkLength());

    fclose(fp0);
    fclose(fp1);
    fclose(fp2);
    fclose(fp3);
    fclose(fp4);
}
```

```
//*******************************************************
// train.cc
//
// Description:  This is the main driver to use the RAN to predict
//               Mackey-Glass signal.
//
//
// Programmer:  Fan Mo
//              Dept. of Electrical & Computer Engineering
//              University of Manitoba
//              Winnipeg, Manitoba
//
//
// Last Update:  Aug. 6, 2002
//*******************************************************
#include "rad.h"
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>

void main()
{
  int Success;
  FILE *fp1, *fp2, *fp3, *fp4;
  double input[DIMENSION];
  double desiredValue;

  if((fp1=fopen("chao.dat", "r"))==NULL)
  {
    printf("Cannot open this file!\n");
    exit(-1);
  }

  if((fp2 = fopen("pred.dat", "w+"))==NULL)
  {
    printf("Cannot write to this file!\n");
    exit(-1);
  }

  if((fp3 = fopen("nodenumber.dat", "w+"))==NULL)
  {
    printf("Cannot write to this file!\n");
    exit(-1);
  }

  if((fp4 = fopen("error.dat", "w+"))==NULL)
  {
    printf("Can't write to this file!\n");
    exit(-1);
  }

  networkClass N;
  N.Start();
```

```
    N.SetScaleMax(1.0);
    N.SetScaleMin(0.07);
    N.SetGama(0.999);
    N.SetErrorMin(0.05);
    N.SetKai(0.87);
    N.SetLearningRate(0.02);
    N.SetBias(1.0);

    int counter = 0;
    while(!feof(fp1))
    {
      //printf(" Read the %dth samples!\n", counter+1);

      for(int j = 0; j < DIMENSION ; j++)
      {
        fscanf(fp1, "%lf", &input[j]);
        //printf("%lf    ", input[j]);
      }

      fscanf(fp1, "%lf", &desiredValue);
      counter++;
      N.AfterNthInput(counter, input,  desiredValue);
      int Success1 = N.Condition1();
      int Success2 = N.Condition2(input);

      if(N.Condition1() && N.Condition2(input))
      {
        N.AllocateNewNode(input);
      }
      else
      {
        N.UpdateWeightCenter(input);
      }

      fprintf(fp2, "%d  %lf\n", counter+34,  N.GetOutput());
      fprintf(fp3, "%d    %d\n", counter+34, N.NetworkLength());
      fprintf(fp4, "%d   %lf\n", counter+34, N.GetPresentError());
    }

    printf("number of nodes = %d\n", N.NetworkLength());

    fclose(fp1);
    fclose(fp2);
    fclose(fp3);
    fclose(fp4);
}
```

```
//*******************************************************
// mac.c
//
// Description:  This is the utility to generate Mackey-Glass signals,
//               which is then used for testing of RAN.
//
// Programmer:  Fan Mo
//              Dept. of Electrical & Computer Engineering
//              University of Manitoba
//              Winnipeg, Manitoba
//
//
// Last Update:  Aug. 6, 2002
//*******************************************************
#include<stdlib.h>
#include<stdio.h>
#include<math.h>

#define Tao 17
void main()
{
   FILE *fp;
   double a = 0.2;
   double b = 0.1;
   int i;
   double x[10000];

   fp = fopen("Mac.dat", "w+");
   if(fp == NULL)
   {
      printf("Can't write to this file!\n");
      exit(-1);
   }

   x[0] = 1.0;
   for(i = 0; i < Tao; i++)
   {
      x[i+1] = (1-b)*x[i];
   }
   for(i = Tao; i < 5000; i++)
   {
      x[i+1] = (1-b)*x[i] + a*x[i-Tao]/(1+pow(x[i-Tao],10));
      fprintf(fp, "%d    %lf\n", i, x[i]);
   }
}
```

```
//*******************************************************
// cut.c
//
// Description:  This is the utility to cut a specific part from a
//               time series.
//
// Programmer:  Fan Mo
//              Dept. of Electrical & Computer Engineering
//              University of Manitoba
//              Winnipeg, Manitoba
//
//
// Last Update:  Aug. 6, 2002
//*******************************************************
#include<stdlib.h>
#include<stdio.h>

void main(int argc, char *argv[])
{
  FILE *fp1, *fp2;
  int start, end, cur, stop = 0;
  double value;
  int counter = 0;

  if(argc!=3)
  {
    printf("You forgot to enter the file names <two file names>\n");
    exit(-1);
  }
  if((fp1=fopen(argv[1], "r"))==NULL)
  {
    printf("Cannot open %s\n", argv[1]);
    exit(-1);
  }
  if((fp2=fopen(argv[2], "w+"))==NULL)
  {
    printf("Cannot write to %s\n", argv[2]);
    exit(-1);
  }
  printf("Enter the start and end position...");
  scanf("%d  %d", &start, &end);

  while(!feof(fp1) && !stop)
  {
    fscanf(fp1, "%d  %lf", &cur, &value);
    if(cur<start)
      counter++;
    else
    {
      if(cur<=end)
        fprintf(fp2, "%d  %lf\n", cur, value);
      else
      {
        stop = 1;
```

```
            }
        }
    }
    fclose(fp1);
    fclose(fp2);
}
```

```
#*******************************************************
# makefile
#
# Description:  This is the makefile used to compile the RAN system.
#
# Programmer:  Fan Mo
#              Dept. of Electrical & Computer Engineering
#              University of Manitoba
#              Winnipeg, Manitoba
#
# Last Update:  Aug. 6, 2002
//*******************************************************
# *translator Defination*

CC = g++ -g
CLINK = g++

.cc.o:; $(CC) -c  $*.cc


NN_dep = \
        rad.o\
        train.o

# *Explicit Rules*

#*Individual File Dependencies*

go: $(NN_dep) rad.h
   $(CLINK) -o  go  $(NN_dep) -lm

clean:
   rm -fr *.o
```

```
//*******************************************************
// DWT.c
//
// Description:  This program computs the one-dimensional
//               orthogonormal wavelet transform of signals
//               stored in a data input file and reconstruct the
//               original signal from its wavelet decomposition
//               using the Mallat's pyramidal algorithm.
//
// Programmer:  Fan Mo
//               Dept. of Electrical & Computer Engineering
//               University of Manitoba
//               Winnipeg, Manitoba
//
// Input parameters:
//
//    N - number of data samples to be processed
//    M - number of scales
//    file - data file name
//
// Output parameters:
//
//   a[i][j] - the approximation of the signal at the resolution
//             2^i and time j. (note: a[0][j] corresponds to the
//             original signal at time j.)
//
//   b[i][j] - the wavelet decomposition result at resolution 2^i
//            and time j.(note:j>=1)
//
//   aa[i][j] - the reconstructed signal at the resolution 2^i
//              and time j. (note: i>=0)
//
// Note: the maximum number of scales is not great than 6!
//
// Last Update:  Aug. 6, 2002
//*******************************************************
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

/* define variables of the input and output parameters */
float a[7][2048],b[7][2048],aa[7][2048],time[2048];
int N,M;

/*the impulse respons coefficients of quadrature mirror filter H and G */
float c1[]={0.0078125,0.046875,0.1171875,0.65625,0.1171875,0.046875,0.0078125};
float h[]={0.125,0.375,0.375,0.125};
float g[]={-2.0,2.0};
float k1[]={0.0078125,0.054685,0.171875,-0.171875,-0.054685,-0.0078125};

/* define the functions used throught program */
void dwt();          /* function for orthogonomal wavelet decomposition */
void recon_dwt();    /* function for reconstructing the original signal
                        from the wavelet decomposition */
```

```
main()
{
  int i,j,nuli;
  int driver, mode;
  char file[50];
  FILE *fp;

  /* Read the signal data file */
  do
  {
    printf("please input the name of the data file:");
    scanf("%s",file);
  }
  while((fp=fopen(file,"rb"))==NULL);
  printf("please input the length of data:");
  scanf("%d",&N);
  printf("input the numger of scales:");
  scanf("%d",&M);

  for(j=0;j<N;j++)
  {
    fscanf(fp,"%f%f\n", &time[j], &a[0][j]);
  }

  fclose(fp);
  printf("**********processing dwt*************\n");
  dwt();          /* find the wavelet decomposition of signal */
  for(i=0;i<=N+10;i++)
    aa[M][i]=a[M][i];
  printf("**********processing recon_dwt***********\n");
  recon_dwt();    /* Reconstruct the signal from the wavelet
                     decomposition */
  if((fp=fopen("source.dat","wb"))==NULL)
  {
    printf("cannot open file source1.dat\n");
    exit(1);
  }
  for(j=0;j<N;j++)
    fprintf(fp,"%f  %f\n",time[j],a[0][j]);
  fclose(fp);

  if((fp=fopen("apps1.dat","wb"))==NULL)
  {
    printf("cannot open file apps1.dat\n");
    exit(1);
  }
  for(j=0;j<N;j++)
    fprintf(fp, "%f %f\n",time[j],a[1][j]);
  fclose(fp);

  if((fp=fopen("apps2.dat","wb"))==NULL)
  {
    printf("cannot open file apps2.dat\n");
    exit(1);
```

```
}
for(j=0;j<N;j++)
   fprintf( fp,"%f  %f\n",time[j],a[2][j]);
fclose(fp);

if((fp=fopen("apps3.dat","wb"))==NULL)
{
   printf("cannot open file apps3.dat\n");
   exit(1);
}
for(j=0;j<N;j++)
   fprintf( fp,"%f  %f\n",time[j],a[3][j]);
fclose(fp);

 if((fp=fopen("recon0.dat","wb"))==NULL)
{
  printf("cannot open file recon0.dat\n");
  exit(1);
}
for(j=0;j<N;j++)
   fprintf(fp, "%f %f\n",time[j],aa[0][j]);
fclose(fp);

if((fp=fopen("recon1.dat","wb"))==NULL)
{
   printf("cannot open file recon1.dat\n");
   exit(1);
}
for(j=0;j<N;j++)
   fprintf(fp,"%f  %f\n", time[j],aa[1][j]);
fclose(fp);

if((fp=fopen("recon2.dat","wb"))==NULL)
{
   printf("cannot open file recon2.dat\n");
   exit(1);
}
for(j=0;j<N;j++)
   fprintf(fp,"%f  %f\n", time[j],aa[2][j]);
 fclose(fp);

 if((fp=fopen("recon3.dat","wb"))==NULL)
 {
    printf("cannot open file recon3.dat\n");
    exit(1);
 }
for(j=0;j<N;j++)
   fprintf(fp,"%f  %f\n", time[j],aa[3][j]);
fclose(fp);

if((fp=fopen("decom1.dat","wb"))==NULL)
{
  printf("cannot open file decom1.dat\n");
  exit(1);
```

```
    }
    for(j=0;j<N;j++)
      fprintf(fp,"%f  %f\n",time[j],b[1][j]);
    fclose(fp);

    if((fp=fopen("decom2.dat","w"))==NULL)
    {
      printf("cannot open file decom2.dat\n");
      exit(1);
    }
    for(j=0;j<N;j++)
      fprintf(fp,"%f  %f\n", time[j],b[2][j]);
    fclose(fp);

    if((fp=fopen("decom3.dat","wb"))==NULL)
    {
      printf("cannot open file decom3.dat\n");
      exit(1);
    }
    for(j=0;j<N;j++)
      fprintf(fp, "%f  %f\n",time[j],b[3][j]);
    fclose(fp);

    if((fp=fopen("decom4.dat","wb"))==NULL)
    {
      printf("cannot open file decom4.dat\n");
      exit(1);
    }
    for(j=0;j<N;j++)
      fprintf(fp, "%f  %f\n",time[j],b[4][j]);
    fclose(fp);

    if((fp=fopen("decom5.dat","wb"))==NULL)
    {
      printf("cannot open file decom5.dat\n");
      exit(1);
    }
    for(j=0;j<N;j++)
      fprintf(fp, "%f  %f\n",time[j],b[5][j]);
    fclose(fp);

    if((fp=fopen("decom6.dat","wb"))==NULL)
    {
      printf("cannot open file decom6.dat\n");
      exit(1);
    }
    for(j=0;j<N;j++)
      fprintf(fp, "%f  %f\n",time[j],b[6][j]);
    fclose(fp);
}
```

```
void dwt()
{
  float p,q;
  int i,j,k;
  float c2;
  int c=1;
  for(k=1;k<=M;k++)
  {
    c2=1/c1[k-1];
    if(k==1)
    {
      for(i=N;i<=N+2*c+1;i++)
        a[k-1][i]=a[k-1][2*N-1-i];
      for(i=0;i<=2;i++);
        {
          p=0;q=0;
            for(j=-1;j<=2;j++)
              p=p+h[j+1]*(*(a[0]+(int)(fabs(i-j+0.5)-0.5)));
            for(j=0;j<=1;j++)
              q=q+g[j]*(*(a[0]+(int)(fabs(i-j+0.5)-0.5)));
            a[1][i]=p;b[1][i]=-c2*q;
        }
       for(i=3;i<=N;i++)
          {
           p=0;q=0;
             for(j=-1;j<=2;j++)
               p=p+h[j+1]*(*(a[0]+i-j));
             for(j=0;j<=1;j++)
               q=q+g[j]*(*(a[0]+i-j));
             a[1][i]=p;b[1][i]=-c2*q;
          }
    }
    else
     {
      for(i=N+1;i<=N+2*c+1;i++)
        a[k-1][i]=a[k-1][2*N-i];
      for(i=c/2;i<=2*c;i++)
      {
        p=0;q=0;
        for(j=-1;j<=2;j++)
          p=p+h[j+1]*(*(a[k-1]+(int)(fabs(i-c*j))));
        for(j=0;j<=1;j++)
          q=q+g[j]*(*(a[k-1]+(int)(fabs(i-c*j))));
        a[k][i-c/2]=p;b[k][i-c/2]=-c2*q;
      }
      for(i=2*c+1;i<=N+c/2;i++)
      {
        p=0;q=0;
        for(j=-1;j<=2;j++)
          p=p+h[j+1]*(*(a[k-1]+i-c*j));
        for(j=0;j<=1;j++)
          q=q+g[j]*(*(a[k-1]+i-c*j));
        a[k][i-c/2]=p;b[k][i-c/2]=-c2*q;
      }
```

```
        }
        c=2*c;
    }
}


void recon_dwt()
{
    int i,j,k;
    float p,q;
    float c2;
    int c=pow(2,M-1);
    for(k=M;k>=1;k--)
    {
        c2=c1[k-1];
        for(i=N+1;i<=N+3*c;i++)
        {
            b[k][i]=-b[k][2*N-i];
            aa[k][i]=aa[k][2*N-i];
        }
        for(i=-c/2;i<=3*c;i++)
        {
            p=0;q=0;
            for(j=-1;j<=2;j++)
                p=p+h[j+1]*(*(aa[k]+(int)(fabs(i+c*j))));
            for(j=-3;j<=2;j++)
            {
                if((i-c*j)<0)
                    q=q+k1[j+3]*(-(*(b[k]+(int)(fabs(i-c*j)))));
                else
                    q=q+k1[j+3]*(*(b[k]+i-c*j));
                aa[k-1][i+c/2]=p-c2*q;
            }
            for(i=3*c+1;i<=N-c/2;i++)
            {
                p=0;q=0;
                for(j=-1;j<=2;j++)
                    p=p+h[j+1]*(*(aa[k]+i+c*j));
                for(j=-3;j<=2;j++)
                    q=q+k1[j+3]*(*(b[k]+i-c*j));
                aa[k-1][i+c/2]=p-c2*q;
            }
            c=c/2;
        }
    }
}
```

```
//*******************************************************
// PNN.h
//
// Description:  This is the header file of the class which encapsulates
//               the probabilistic neural networks (PNN)
//
// Programmer:  Fan Mo
//              Dept. of Electrical & Computer Engineering
//              University of Manitoba
//              Winnipeg, Manitoba
//
// Last Update:  Aug. 6, 2002
//*******************************************************
#ifndef _PNN_H
#define _PNN_H

#define DIMENSION 64

struct patternNodeDataField{
  int      position;
  double  center[DIMENSION];
  double  width;
  double  response;
};

struct patternNode;
typedef patternNode* ptrType;

class networkClass{
public:
  //constructor and destructor:
  networkClass();
  networkClass(int size, ptrType head);
  ~networkClass();

  // networkClass interface
  int NetworkIsEmpty();
  int NetworkLength();
  void NodeInsert(int NewPosition, const struct patternNodeDataField & NewDataField, int&Success);
  void NodeDelete(int Position, int & Success);
  void NetworkClear();
  void NodeRetrieve(int Position, struct patternNodeDataField & Record, int & Success);
  double EuclidDistance(double * Vector1, double * Vector2);
  void AfterNthInput(int n, double * input, double DesiredValue);
  void OutputPatternNode(double * input);
  int AllocateNewNode(double * input, double width);
  double GetOutput() const {return Output;}
  void TotalOutput(double * input);
  void Start(double width);

private:
  ptrType PtrTo(int Position);
  double Output;
  int Size;
```

```
    double PresentError;
    ptrType Head;
};

#endif
```

```
//*******************************************************
// PNN.cc
//
// Description:  This is the implementation file of the probabilistic
//               neural networks (PNN)
//
// Programmer:  Fan Mo
//              Dept. of Electrical & Computer Engineering
//              University of Manitoba
//              Winnipeg, Manitoba
//
// Last Update:  Aug. 6, 2002
//*******************************************************
#include "PNN.h"
#include <stddef.h>          // for NULL
#include <assert.h>          //for assert()
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <math.h>

struct patternNode {
   patternNodeDataField   patternNodeData;
   patternNode *          Next;
};

// constructor and destructor
networkClass :: networkClass() : Output(0.0), Size(0), PresentError(1.0), Head(NULL) {}

networkClass :: networkClass(int size, ptrType head):
Output(0.0),
PresentError(1.0)
{
   Size = size;
   Head = head;
};

networkClass :: ~networkClass()
{
   int Success;

   while(!NetworkIsEmpty())
   {
      NodeDelete(1, Success);
   }
}

// networkClass interface
int networkClass :: NetworkIsEmpty()
{
   return (Size == 0);
}

int networkClass :: NetworkLength()
```

```
{
  return Size;
}

void networkClass :: NodeInsert(int NewPosition, const struct patternNodeDataField & NewDataField, int & Success)
{
  int NewLength = NetworkLength() + 1;

  Success = ((NewPosition >= 1) && (NewPosition <= NewLength));
  if(Success)
  {
    Size = NewLength;
    //creat new node and place new node in it
    ptrType NewPtr = new patternNode;
    Success = (NewPtr != NULL);
    if(Success)
    {
      NewPtr->patternNodeData.position = NewDataField.position;
      for(int j = 0; j < DIMENSION; j++)
        NewPtr->patternNodeData.center[j] = NewDataField.center[j];

      NewPtr->patternNodeData.width    = NewDataField.width;
      NewPtr->patternNodeData.response = NewDataField.response;
      if(NewPosition == 1)
      {
        NewPtr->Next = Head;
        Head = NewPtr;
      }
      else
      {
        ptrType Prev = PtrTo(NewPosition - 1);
        //insert new node after node to which Prev points
        NewPtr->Next = Prev->Next;
        Prev->Next = NewPtr;
      }
    }
  }
}


void networkClass :: NodeDelete(int Position, int & Success)
{
  ptrType  Cur;
  Success = ((Position >= 1) && (Position <= NetworkLength()));
  if(Success)
  {
    --Size;
     if(Position == 1)
     {
       // delete the first node from the network
       Cur = Head;
       Head = Head->Next;
```

```
        }
        else
        {
            ptrType Prev = PtrTo(Position - 1);
            Cur = Prev->Next;
            Prev->Next = Cur->Next;
        }
        // return node to system
        Cur->Next = NULL;
        delete Cur;
        Cur = NULL;
    }
}


void networkClass :: NodeRetrieve(int Position, struct patternNodeDataField &Record, int & Success)
{
    ptrType Cur;

    Success = ((Position >= 1)&&(Position <= NetworkLength()));
    if(Success)
    {
        Cur = PtrTo(Position);
    }
    Record.position = Cur->patternNodeData.position;
    for(int i = 0; i < DIMENSION; i++)
        Record.center[i] = Cur->patternNodeData.center[i];

    Record.width = Cur->patternNodeData.width;
    Record.response = Cur->patternNodeData.response;
}


double networkClass :: EuclidDistance(double * Vector1, double * Vector2)
{
    double distance = 0.0;

    for(int i = 0; i < DIMENSION; i++)
        distance += (Vector1[i]-Vector2[i])*(Vector1[i]-Vector2[i]);

    return (sqrt(distance));
}


void networkClass :: OutputPatternNode(double * input)
{
    double temp;
    ptrType ptr;
    int length = NetworkLength();

    for(int i = 0; i < length; i++)
```

```
    {
        ptr = PtrTo(i+1);
        temp = EuclidDistance(input, ptr->patternNodeData.center);
        temp = -temp*temp/(ptr->patternNodeData.width*ptr->patternNodeData.width);
        ptr->patternNodeData.response = exp(temp);
    }
}



void networkClass :: TotalOutput(double * input)
{
    ptrType ptr;
    Output = 0.0;

    OutputPatternNode(input);
    int length = NetworkLength();
    for(int i = 0; i < length; i++)
    {
        ptr = PtrTo(i+1);
        Output += ptr->patternNodeData.response;
    }
}



void networkClass :: AfterNthInput(int n, double * input, double DesiredValue)
{
    TotalOutput(input);
    PresentError = DesiredValue - Output;
}



int networkClass :: AllocateNewNode(double * input, double width)
{
    int Success;
    patternNodeDataField NewNodeData;
    ptrType ptr;

    //printf(" Allocating a new node...\n");
    int newposition = NetworkLength() + 1;
    NewNodeData.position = newposition;
    NewNodeData.width = width;
    for(int j = 0; j < DIMENSION; j++)
    {
        NewNodeData.center[j] = input[j];
    }

    NodeInsert(newposition, NewNodeData, Success);
    if(Success)
        return Success;
    else
      return 0;
```

```
}


void networkClass :: Start(double width)
{
   int Success;
   patternNodeDataField nodedata;
   nodedata.position = 1;

   for(int i = 0; i < DIMENSION; i++)
   {
      nodedata.center[i] = 0.0;
   }
   nodedata.width    = width;
   nodedata.response = 0.0;

   NodeInsert(1, nodedata, Success);
}


ptrType networkClass :: PtrTo(int Position)
{
   if((Position < 1)||(Position > NetworkLength()))
      return NULL;
   else
   {
      ptrType Trav = Head;
      for(int Skip = 1; Skip < Position; Skip++)
        Trav = Trav->Next;

     return Trav;
   }
}


void networkClass :: NetworkClear()
{
   int Success;

   while(!NetworkIsEmpty())
   {
      NodeDelete(1,Success);
   }
}
```

```
//*******************************************************
// train.cc
//
// Description:  This is the main driver to use PNN to perform
//               classification.
//
//
// Programmer:  Fan Mo
//              Dept. of Electrical & Computer Engineering
//              University of Manitoba
//              Winnipeg, Manitoba
//
// Last Update:  Aug. 6, 2002
//*******************************************************
#include "PNN.h"
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>

#define TOTALCLASS 3

void main()
{
   int Success;
   FILE *fp1, *fp2, *fp3, *fp4;
   double input[DIMENSION];
   int classMember;
   int testingErrorNumber = 0;
   double testingError;

   double WIDTH;
   double WIDTH_STEP = 0.1;
   double WIDTH_START = 0.1;
   double WIDTH_END = 2.0;
   int ArraySize = int((WIDTH_END - WIDTH_START)/WIDTH_STEP) + 1;
   double CorrectRate[ArraySize];

   for(int i = 0; i < ArraySize; i++)
      CorrectRate[ArraySize] = 0.0;

   char trainingFile[50];
   char testingFile[50];
   char resultFile[50];

   printf("Enter the name of training file...\n");
   scanf("%s", trainingFile);

   if((fp1 = fopen(trainingFile, "r"))==NULL)
   {
      printf("Cannot open this file!\n");
      exit(-1);
   }

   printf("Enter the name of testing file...\n");
```

```
scanf("%s", testingFile);
if((fp2 = fopen(testingFile, "r"))==NULL)
{
   printf("Caanot open this file!\n");
   exit(-1);
}

printf("Enter the name of result file...\n");
scanf("%s", resultFile);
if((fp3 = fopen(resultFile, "w"))==NULL)
{
   printf("Cannot open this file for writing!\n");
   exit(-1);
}

if((fp4 = fopen("conf.dat", "w"))==NULL)
{
   printf("Cannot open this file for writing!\n");
   exit(-1);
}

networkClass N[TOTALCLASS];
int Counter[TOTALCLASS];

for(int ii = 0; ii < ArraySize; ii++)
{
   WIDTH = WIDTH_START + WIDTH_STEP*ii;
   for( int i = 0; i < TOTALCLASS; i++)
   {
      N[i].NetworkClear();
      Counter[i] = 0;
   }
   double desiredValue1, desiredvalue2;

   //start training the network
   while(!feof(fp1))
   {
      for(int j = 0; j < DIMENSION; j++)
      {
         if ( fscanf(fp1, "%lf", &input[j]) == EOF )
            break;
         //printf("   %lf   ", input[j]);
      }
      if ( feof(fp1) )
         break;

      fscanf(fp1, "%d", &classMember);
      //printf("   %d  \n\n ", classMember);
      if(classMember == 1)
      {
         N[0].AllocateNewNode(input, WIDTH);
         Counter[0]++;
      }
      else if(classMember == 2)
```

```
    {
      N[1].AllocateNewNode(input, WIDTH);
      Counter[1]++;
    }
    else if(classMember == 3)
    {
      N[2].AllocateNewNode(input, WIDTH);
      Counter[2]++;
    }
    else
    {
       printf("One sample is incorrectly labled!\n");
    }
}
printf("End of training.\n");

for(int i = 0; i < TOTALCLASS; i++)
   printf("\n SubNetwork %d: number of nodes = %d\n",i+1, N[i].NetworkLength());

printf("Start testing the network...\n");
int TotalTest = 0;
int CorrectCounter = 0;
double NOutput[TOTALCLASS];
int Decision;
double Confidence = 0.0;

for(int i = 0; i < TOTALCLASS; i++)
   NOutput[i] = 0.0;

double Mean_Conf;
double Conf_Accum= 0;
int Conf_Count = 0;
while(!feof(fp2))
{
   for(int j = 0; j < DIMENSION; j++)
   {
      fscanf(fp2, "%lf", &input[j]);
      //printf("%lf   ", input[j]);
   }
   fscanf(fp2,"%d", &classMember);
   //printf("   %d", classMember);
   TotalTest++;

   for(int i = 0; i < TOTALCLASS; i++)
   {
      N[i].AfterNthInput(TotalTest, input, 1);
      NOutput[i] = N[i].GetOutput();
   }

   double Max;
   int IndexMax;
   Max = NOutput[0];
   IndexMax = 0;
```

```
        for(int i = 1; i < TOTALCLASS; i++)
        {
          if(Max < NOutput[i])
          {
             Max = NOutput[i];
             IndexMax = i;
          }
        }

        Decision = IndexMax + 1;
        double temp = 0.0;
        for(int i = 0; i < TOTALCLASS; i++)
          temp += NOutput[i];

        Confidence = (float)NOutput[IndexMax]/(float)temp;
        Conf_Accum +=Confidence;
        Conf_Count++;

        if(classMember == Decision)
        {
          CorrectCounter++;
        }

          printf("TotalTest = %d,  Decision = %d,  Confidence = %lf,  True classMember = %d,  CorrectCounter =
%d  \n\n", TotalTest, Decision, Confidence, classMember, CorrectCounter);
      }

      Mean_Conf = Conf_Accum/Conf_Count;
      printf("The Average Confidence is %lf\n", Mean_Conf);

      CorrectRate[ii] = (float)CorrectCounter/TotalTest;
      //printf("\n\n Width = %lf, CorrectRate[%d] = %lf\n",WIDTH, ii, CorrectRate[ii]);
      fprintf(fp3, "%lf  %lf\n", WIDTH, CorrectRate[ii]);
      fprintf(fp4, "%lf %lf\n", WIDTH, Mean_Conf);

      Conf_Count = 0;
      Conf_Accum = 0.0;
      rewind(fp1);
      rewind(fp2);
    }

    fclose(fp1);
    fclose(fp2);
    fclose(fp3);
    fclose(fp4);
  }
```

```
#*******************************************************
# makefile
#
# Description:  This is the makefile used to compile the PNN.
#
#Programmer:  Fan Mo
#                Dept. of Electrical & Computer Engineering
#                University of Manitoba
#                Winnipeg, Manitoba
#
#  Last Update:   Aug. 6, 2002
//*******************************************************
#  *translation Defination*

CC = g++  -g
CLINK = g++

.cc.o:; $(CC) -c $*.cc

NN_dep = \
          PNN.o\
          train.o

# *Explicit Rules*

# *Individual FILE Dependence*

go:  $(NN_dep)  PNN.h
     $(CLINK) -o go $(NN_dep) -lm

clean:
   rm -fr *.o
```

```
//*******************************************************
// BPN.h
//
// Description:  This is the header file which defines the back-propagation
//               networks (BPN)
//
// Programmer:  Fan Mo
//              Dept. of Electrical & Computer Engineering
//              University of Manitoba
//              Winnipeg, Manitoba
//
// Last Update:  Aug. 6, 2002
//*******************************************************
#ifndef _BP_H
#define _BP_H

struct layer {
   float * outputs;
   float ** weights;
   float * errors;
   float ** last_deltas;
   int  thislayersize;
};

struct BPN{
   layer * InUnits;
   layer * OutUnits;
   layer ** Layers;
   double   Alpha;
   double   Eta;
   int TotalLayerNumber;
   int InputSize;
   int OutputSize;
};

extern float * floatVector(int size);
extern int initializeBPN( BPN * NET, int layernumber, int * layersize, double alpha, double eta);
extern void set_inputs(BPN * NET, float * NET_IN, int inputsize);
extern void propagate_layer(BPN * NET,layer * LOWER, layer * UPPER);
extern void propagate_forward(BPN * NET);
extern void get_outputs(BPN * NET, float * OUTPUTS);
extern void comput_output_error(BPN * NET, float * TARGET);
extern void backpropagate_error(layer * UPPER, layer * LOWER);
extern void backpropagate(BPN * NET);
extern void adjust_weights(BPN * NET);

#endif
```

```
//*********************************************************
// BPN.c
//
// Description:  This is the implementation file of BPN.
//
//
// Programmer:  Fan Mo
//                  Dept. of Electrical & Computer Engineering
//                  University of Manitoba
//                  Winnipeg, Manitoba
//
//  Last Update:  Aug. 6, 2002
//*********************************************************
#include "BPN.h"
#include <stddef.h>  //for NULL
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


float * floatVector(int size)
{
   float * floatPtr;

   floatPtr = (float *)malloc(size * sizeof(float));
   if(floatPtr==NULL)
   {
      printf("Memory allocation error!\n");
      exit(-1);
   }
   for( int i = 0; i < size; i++)
   {
      floatPtr[i] = 0.000001;
   }
   return floatPtr;
}



int initializeBPN( BPN * NET, int layernumber, int * layersize, double alpha, double eta)
{
   layer * layerPtr;
   NET->Layers = new layer *[layernumber];
   NET->Alpha = alpha;
   NET->Eta = eta;
   NET->TotalLayerNumber = layernumber;
   NET->InputSize = layersize[0];
   NET->OutputSize = layersize[layernumber - 1];

   for(int i = 0; i < layernumber; i++)
   {
      layerPtr = new layer;
      layerPtr->thislayersize = layersize[i];
```

```
        //printf("layerPtr->thislayersize = %d\n", layerPtr->thislayersize);
        layerPtr->outputs = floatVector(layersize[i]);
        layerPtr->errors = floatVector(layersize[i]);
        NET->Layers[i] = layerPtr;
    }

    for(int j = 1; j < layernumber; j++)
    {
        //printf("NET->Layers[%d]->thislayersize = %d\n", j+1, NET->Layers[j]->thislayersize);
        int currentlayersize  = NET->Layers[j]->thislayersize;
        int previouslayersize = NET->Layers[j-1]->thislayersize;
        NET->Layers[j]->weights = new float *[currentlayersize];
        NET->Layers[j]->last_deltas = new float *[currentlayersize];

        for( int k = 0; k < currentlayersize; k++)
        {
            NET->Layers[j]->weights[k]= floatVector(previouslayersize);
            NET->Layers[j]->last_deltas[k] = floatVector(previouslayersize);
        }
    }
    NET->InUnits = new layer;
    NET->InUnits = NET->Layers[0];
    NET->OutUnits = new layer;
    NET->OutUnits = NET->Layers[layernumber - 1];

    return 0;
}



void set_inputs(BPN * NET, float * NET_IN, int inputsize)
{
    for(int i = 0; i < inputsize; i++)
    {
        NET->InUnits->outputs[i] = NET_IN[i];
    }
}



void propagate_layer(BPN * NET, layer * LOWER, layer * UPPER)
{
    float * inputs;
    float * currents;
    float * connects;
    float sum;
    inputs = LOWER->outputs;
    currents = UPPER->outputs;

    for( int i = 0; i < UPPER->thislayersize; i++)
    {
        sum = 0.0;
        connects = UPPER->weights[i];
```

```
      for( int j = 0; j < LOWER->thislayersize; j++)
      {
         printf("inputs[%d]= %f, connects[%d]= %f\n",j, inputs[j], j,connects[j]);
         sum += inputs[j] * connects[j];
      }
      if(UPPER = NET->OutUnits)
      {
          currents[i] = sum;
      }
      else
      {
          currents[i] = 1.0/(1.0 + exp(-sum));
      }
      printf("sum = %f,  currents[%d] = %f\n", sum, i, currents[i]);
   }
}



void propagate_forward(BPN * NET)
{
   layer *upper;
   layer * lower;

   for( int i = 0; i < NET->TotalLayerNumber-1; i++)
   {
      lower = NET->Layers[i];
      upper = NET->Layers[i+1];
      propagate_layer(NET,lower, upper);
   }
}



void get_outputs(BPN * NET, float * OUTPUTS)
{
   float * temp1;
   float * temp2;
   temp1 = NET->OutUnits->outputs;
   temp2 = OUTPUTS;

   for( int i = 0; i < NET->OutputSize; i++)
   {
      temp2[i] = temp1[i];
   }
}



void comput_output_error(BPN * NET, float * TARGET)
{
   float * errors;
   float * outputs;
```

```
    errors = NET->OutUnits->errors;
    outputs = NET->OutUnits->outputs;

    for( int i = 0; i < NET->OutputSize; i++)
    {
        errors[i] = outputs[i] * (1-outputs[i])*(TARGET[i]-outputs[i]);
    }
}



void backpropagate_error(layer * UPPER, layer * LOWER)
{
    float * senders;
    float * receivers;
    float * connects;
    float unit;

    senders = UPPER->errors;
    receivers = LOWER->errors;

    for( int i = 0; i < LOWER->thislayersize; i++)
    {
        receivers[i] = 0.0;
        for(int j = 0; j < UPPER->thislayersize; j++)
        {
            connects = UPPER->weights[j];
            receivers[i] += senders[j]*connects[i];
        }

        unit = LOWER->outputs[i];
        receivers[i] = receivers[i] * unit * (1-unit);
    }
}



void backpropagate(BPN * NET)
{
    layer * upper;
    layer * lower;

    for( int i = NET->TotalLayerNumber - 1; i > 0; i--)
    {
        upper = NET->Layers[i];
        lower = NET->Layers[i-1];
        backpropagate_error(upper, lower);
    }
}
```

```
void adjust_weights(BPN * NET)
{
   layer * current;
   layer * previous;
   float * inputs;
   float * units;
   float * weight;
   float * delta;
   float * error;

   for( int i = 1; i < NET->TotalLayerNumber; i++)
   {
      current  = NET->Layers[i];
      previous = NET->Layers[i-1];
      units = NET->Layers[i]->outputs;

      inputs = NET->Layers[i-1]->outputs;
      for( int j = 0; j < current->thislayersize; j++)
      {
         weight = current->weights[j];
         delta = current->last_deltas[j];
         error = NET->Layers[i]->errors;
         for(int k = 0; k < previous->thislayersize; k++)
         {
            weight[k] += (inputs[k]*NET->Eta*error[j])+(NET->Alpha*delta[k]);
            delta[k] = inputs[k]*NET->Eta*error[j];
         }
      }
   }
}
```

```
//*******************************************************
// train.cc
//
// Description:  This is the main driver to use BPN to perform
//               classification.
//
//
// Programmer:   Fan Mo
//               Dept. of Electrical & Computer Engineering
//               University of Manitoba
//               Winnipeg, Manitoba
//
// Last Update:  Aug. 6, 2002
//*******************************************************
#include "BPN.h"
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <math.h>

#define InputSize 2
#define OutputSize 1


int  main()
{
   int * layersize;
   FILE * fp1, *fp2;
   float input[InputSize];

   struct BPN  * NET = new BPN;
   layersize[0] = InputSize;
   layersize[1] = InputSize;
   layersize[2] = 1;
   double alpha     = 0.2;
   double eta       = 0.05;
   int layernumber = 3;

   NET->Alpha = alpha;
   NET->Eta = eta;
   int a = initializeBPN(NET, layernumber, layersize, alpha, eta);

   printf("Start training the network.....\n");
   if((fp1 = fopen("train.dat", "r"))==NULL)
   {
      printf("Cannot open this file!\n");
       exit(-1);
   }

   if((fp2 = fopen("test.dat", "r"))==NULL)
   {
      printf("Cannot open this file!\n");
      exit(-1);
   }
```

```
int epocNumber = 0;
int * intptr;
float * outputs;
float * targets;
outputs = floatVector(OutputSize);
targets = floatVector(OutputSize);
double error = 0.0;
int ContinueTraining = 1;
int counter;
int testingErrorNumber;
int  classMember;

while(ContinueTraining)
{
   epocNumber++;
   testingErrorNumber = 0;
   counter = 0;

   printf("epocNumber = %d\n", epocNumber);
   while(!feof(fp1))
   {
      printf("Read the %dth samples!\n", counter+1);
      for(int j = 0; j < InputSize; j++)
      {
         fscanf(fp1, "%f", &input[j]);
         printf("   %f   ", input[j]);
      }

      fscanf(fp1, "%d", &classMember);
      printf("   %d\n", classMember);
      *targets = (float)classMember;
      counter++;

      set_inputs(NET, input, InputSize);
      propagate_forward(NET);
      get_outputs(NET, outputs);
      printf("\nTotal outputs = %f\n", *outputs);

      comput_output_error(NET, targets);
      printf("errors = %f\n", *(NET->OutUnits->errors));
      backpropagate(NET);

      adjust_weights(NET);

      for( int j = 0; j < OutputSize; j++)
         error +=(outputs[j]-targets[j])*(outputs[j]-targets[j]);
   }
   error = sqrt(error);
   if(error > 0.1)
   {
      ContinueTraining = 1;
      rewind(fp1);
   }
   else
```

```
    {
      ContinueTraining = 0;
    }
  }

  printf("Starting testing...\n");

  float * Error;
  Error = floatVector(OutputSize);
  int TotalTest = 0;
  int CorrectCounter = 0;
  double CorrectRate;

  while(!feof(fp2))
  {
    for(int j = 0; j < InputSize; j++)
    {
      fscanf(fp2, "%lf", &input[j]);
      printf("%lf   ", input[j]);
    }

    fscanf(fp2, "%d", &classMember);
    printf("    %d\n", classMember);
    TotalTest++;
    set_inputs(NET, input, InputSize);
    propagate_forward(NET);
    get_outputs(NET, outputs);

    *targets = (float)classMember;
    for(int i = 0; i < OutputSize; i++)
    {
      Error[i] = fabs(targets[i]-outputs[i]);
      if(Error[i] < 0.2)
      {
        CorrectCounter++;
      }
    }
  }

  TotalTest--;
  CorrectCounter--;
  CorrectRate = (double)CorrectCounter/(double)TotalTest;
   printf("TotalTest = %d,    CorrectCounter = %d,   CorrectRate = %lf\n", TotalTest, CorrectCounter, CorrectRate);

  printf("End of programming\n");
  delete NET;

  fclose(fp1);
  fclose(fp2);

  return 0;
}
```

```
#*********************************************************
# makefile
#
# Description:  This is the makefile used to compile the BPN.
#
#Programmer:  Fan Mo
#             Dept. of Electrical & Computer Engineering
#             University of Manitoba
#             Winnipeg, Manitoba
#
#  Last Update:   Aug. 6, 2002
//*********************************************************
#  *translation Defination*

CC = g++  -g
CLINK = g++

.cc.o:; $(CC) -c $*.cc

NN_dep = \
        BPN.o\
        train.o

# *Explicit Rules*

# *Individual FILE Dependence*

go: $(NN_dep) BPN.h
   $(CLINK) -o go $(NN_dep)  -lm

clean:
   rm -fr *.o
```

```
//*******************************************************
// addnoise.cc
//
// Description:  This is the utility to generate different noises,
//               which is then added to signal.
//
// Programmer:  Fan Mo
//              Dept. of Electrical & Computer Engineering
//              University of Manitoba
//              Winnipeg, Manitoba
//
//
// Last Update:  Aug. 6, 2002
//*******************************************************
#include<stdio.h>
#include<math.h>
#include<stdlib.h>

#define InputDimension 64

double AveragePower(float input[])
{
   int i;
   double temp=0.0;

   for(i=0;i<InputDimension;i++)
   {
      temp=temp + (double)(input[i] * input[i]);
   }
   return temp/InputDimension;
}



double SNR(float Signal[],float Noise[])
{
   double temp=0.0;

   temp=AveragePower(Signal)/AveragePower(Noise);
   return 10*log(temp)/log(10.0);
 }



int main()
{
   int i, counter = 0;
   int classMember;

   float Factor;
   float SNR_SET;
   float Time[InputDimension],Signal[InputDimension], Noise[InputDimension];
   float temp=0.0,test;
   float OldSNR, NewSNR;
```

```
char SignalFile[30], NoiseFile[30], S_add_NFile[30], Single_colum[30];
FILE *fp1, *fp2, *fp3;

printf("Enter the signal filename.\n");
scanf("%s",SignalFile);

printf("Enter the noise filename.\n");
scanf("%s", NoiseFile);

printf("Enter the value of SNR_SET.\n");
scanf("%f", &SNR_SET);

printf("Enter the output filename. \n");
scanf("%s", S_add_NFile);

if((fp1=fopen(SignalFile,"r"))==NULL)
{
   printf("cannot open the %s file!\n",SignalFile);
   exit(1);
}

if((fp2=fopen(NoiseFile,"r"))==NULL)
{
   printf("cannot open the $s file!\n", NoiseFile);
   exit(1);
}

if((fp3=fopen(S_add_NFile,"w"))==NULL)
{
   printf("cannot open the %s file!\n", S_add_NFile);
   exit(1);
}

while(!feof(fp1))
{
   counter++;
   printf("Read the %dth samples...\n", counter);

   for(i = 0; i < InputDimension; i++)
   {
      fscanf(fp1, "%f", &Signal[i]);
      if(feof(fp2))
         rewind(fp2);
      fscanf(fp2, "%f %f", &temp, &Noise[i]);
   }

   fscanf(fp1, "%d", &classMember);
   OldSNR = SNR(Signal, Noise);

   printf("P=%f, N=%f, OldSNR=%f\n",AveragePower(Signal),AveragePower(Noise),
          OldSNR);

   temp=(SNR_SET-OldSNR)/10.0;
   Factor=(float)(sqrt(1.0/pow(10,temp)));
```

```
        for(i = 0; i < InputDimension; i++)
        {
           Noise[i]=(float)(Noise[i]*Factor);
           Signal[i]=Signal[i]+Noise[i];
           fprintf(fp3, "%lf  ", (double)Signal[i]);
        }

        fprintf(fp3, "%d\n", classMember);
        NewSNR=SNR(Signal,Noise);

        if(fabs((double)(NewSNR)-(double)(SNR_SET))/SNR_SET >0.05)
            printf("Sorry, try again!\n");

        printf("OK!\n");

        printf("OldSNR=%f, NewSNR=%f, Factor=%f, SNR_SET=%f\n",OldSNR, NewSNR,
          Factor, SNR_SET);
    }
    fclose(fp1);
    fclose(fp2);
    fclose(fp3);
}
```