# Intelligent Monitoring and Prediction Systems

Thesis submitted in accordance with the
requirements of the University of Liverpool
for the degree of Doctor of Philosophy

in

Electrical Engineering and Electronics

by

Howard Lewis, BEng.

October 2000

This thesis is dedicated to Alison for her constant love and for never losing faith in me.

In loving memory of my father; who taught me to think dialectically.

## Acknowledgements

I would like to thank my Supervisor, Professor Henry Wu for his guidance throughout this project. I would also like to thank Dr. David Shimmin and Dr. Paul Griffiths for their occasional assistance. Thanks also go to the other members of the Intelligence Engineering and Automation research group who aided me along the way, especially Dr. Jiang Lin.

In addition, I would like to thank Shell UK and the NGC for supplying me with the research data.

I would also like to thank my family for their love and support over the years and for bringing me up with an interest in science and the world around me. They taught me to take nothing at face value and to question everything.

Finally I would like to thank all my friends for their support and encouragement when the going got tough and for helping keep me sane.

> nothing remains what, where and as it was, but everything moves, changes, comes into being and passes away
>
> merely quantitative differences beyond a certain point pass into qualitative changes
>
> Engels

## Abstract

Intelligent Monitoring

and Prediction Systems

by

Howard Lewis

The goal of the research work undertaken was to produce a novel intelligent monitoring and prediction system for real world time series prediction and classification problems. Industrial process data from three sources was used during the course of this research. *Shell UK* provided infra-red scatter data that measured oil mist. *Shell* also provided process data from a catalytic cracker. The third set of data came from the *NGC* and consisted of data used to measure the operating conditions of a normal and a faulty power transformer.

The structure of an intelligent monitoring and prediction system would include data preprocessing to extract the data's key features, a neural network for pattern classification and a genetic classifier to dynamically develop a rule base as the system develops.

Techniques examined for preprocessing the raw data include: principal component analysis and the wavelet transform. Both back propagation and radial basis function neural networks have been used to examine pattern classification of the processed data. Genetic classifiers, are also investigated as components of an intelligent monitoring and prediction system.

Extracting the wavelet coefficients of the infra-red scatter data, resulted in a large decrease in the dimensionality of the pattern space to be separated by a neural network. However, although backpropagation networks could be produced that trained properly, generalisation of the networks fell short of the accuracy required for real world time series prediction problems.

PCA of the catalytic cracker data has allowed the key variables to be identified. These variables have been used to attempt to train neural networks for time series prediction.

PCA of the transformer data allowed an index number to be introduced to measure the distribution of the variance of the principal components. The ratio between the index numbers of the normal and faulty transformers was sufficiently large that it could be used to report the variation of the transformers condition and could therefore be used as an alarm for fault monitoring.

# Contents

**List of Figures** viii

**List of Tables** xii

**1 Introduction** 1
- 1.1 Aims and Objectives . . . 1
- 1.2 Methodologies . . . 2
  - 1.2.1 Neural Networks . . . 2
  - 1.2.2 Wavelet Transforms . . . 6
  - 1.2.3 Genetics Based Machine Learning . . . 9
- 1.3 The Industrial Data . . . 12

**2 Neural Networks for Information Extraction** 15
- 2.1 Introduction . . . 15
- 2.2 Neural Networks . . . 17
  - 2.2.1 The Neuron . . . 17
  - 2.2.2 The Multilayer Perceptron Neural Network . . . 21
  - 2.2.3 Radial Basis Function Neural Networks . . . 29
  - 2.2.4 Self-Organised Learning . . . 35
  - 2.2.5 A Simplified Neuron Model as a Principal Component Analyser . . . 39
  - 2.2.6 Self-Organised Principal Component Analysis . . . 41
- 2.3 Pre-Processing the Neural Network Input Data . . . 46
  - 2.3.1 Statistical Averages . . . 46
  - 2.3.2 The Fast Fourier Transform . . . 49
  - 2.3.3 Power Spectral Density . . . 50
  - 2.3.4 Autocorrelation . . . 51
  - 2.3.5 Cross-Correlation . . . 52
  - 2.3.6 The Correlation Coefficient . . . 54
- 2.4 Information Extraction of the Oil Mist Data . . . 56
  - 2.4.1 The Oil Mist Data . . . 56

2.4.2 Training With The Raw Data . . . 58
2.4.2.1 Pre-Processing the Data with the Cross-Correlation Function . . . 61
2.4.2.2 Pre-Processing the Data with the Autocorrelation Function . . . 70
2.4.2.3 Pre-Processing the Data with the Power Spectral Density Function . . . 76
2.4.3 Training with the Signal Statistics . . . 76
2.5 Conclusion . . . 80

**3 Principal Component Analysis and it's Applications 83**
3.1 Introduction . . . 83
3.2 Principal Component Analysis . . . 86
3.2.1 The Eigen-Structure of the Principal Components . . . 87
3.2.2 Derivation of the Principle Components . . . 88
3.2.3 PCA Selection of the Key Variables . . . 92
3.2.4 Analysis of Component Scores . . . 93
3.3 Application of PCA to Industrial Data . . . 95
3.3.1 Principal Component Analysis of a Power Transformer . 95
3.3.2 Identification of Key Transformer Variables and Interdependence Analysis . . . 97
3.3.2.1 The Transformer Data . . . 97
3.3.2.2 Principal Component Analysis of the Transformer Data . . . 100
3.3.2.3 Variance Analysis of the Principal Components for Condition Monitoring . . . 103
3.3.3 Principal Component Analysis of the Catalytic Cracker Data. . . . 105
3.3.4 Eigenvalue Analysis of the Oil Mist Data . . . 120
3.4 Conclusion . . . 123

**4 The Wavelet Transform for Information Extraction 126**
4.1 Introduction . . . 126
4.2 Limitations of the Fourier Transform . . . 128
4.3 The Continuous Wavelet Transform . . . 134
4.3.1 Computing the Wavelet Transform . . . 138
4.4 The Discrete Wavelet Transform . . . 141
4.5 Analysis of the Oil Mist data . . . 150
4.5.1 Raw Data . . . 150
4.5.2 Normalised Data . . . 163
4.6 Conclusion . . . 170

**5 Expert Systems And Evolutionary Computation** **172**
5.1 Introduction . . . . . 172
5.2 Expert Systems . . . . . 176
5.2.1 Components of an Expert System . . . . . 177
5.2.2 Expert System Rules . . . . . 180
5.3 Evolutionary Computation . . . . . 184
5.3.1 Genetic Algorithms . . . . . 184
5.3.2 Genetic Classifiers . . . . . 188
5.3.3 Genetic Programming . . . . . 191
5.4 Learning Expert Systems . . . . . 196
5.4.1 Message Coding . . . . . 197
5.4.2 Training the Intelligent System . . . . . 199
5.5 Conclusion . . . . . 201

**6 Conclusion** **204**
6.1 Introduction . . . . . 204
6.2 Methodologies . . . . . 206
6.3 Results . . . . . 207
6.3.1 Infra-Red Scatter Data . . . . . 207
6.3.2 Catalytic Cracker Data . . . . . 209
6.3.3 Power Transformer Data . . . . . 210

**References** **213**

# List of Figures


2.1 The basic neuron . . . . . 18
2.2 Two dimensional pattern space . . . . . 22
2.3 A multilayer perceptron neural network . . . . . 24
2.4 Hyperbolic tangent sigmoid transfer function . . . . . 25
2.5 A radial basis function . . . . . 30
2.6 A radial basis function neural network . . . . . 32
2.7 Signal flow graph of a linear neuron . . . . . 39
2.8 Signal flow graph of equations 2.2.26 and 2.2.27 . . . . . 41
2.9 Principal component neural network . . . . . 42
2.10 A typical 20 second sample of the Shell oil mist scatter data . . 57
2.11 Choosing a spread constant . . . . . 59
2.12 The sum-squared error for a radial basis network trained using the raw data. . . . . 61
2.13 Cross-correlation coefficients for High 42 data sample . . . . . 63
2.14 Cross-correlation coefficients for Low 15 data sample . . . . . 63
2.15 Cross-correlation coefficients for Grey 24 data sample . . . . . 64
2.16 Cross-correlation coefficients for None 18 data sample . . . . . 64
2.17 Cross-correlation coefficients for High 50 data sample . . . . . 65
2.18 Cross-correlation coefficients for Low 6 data sample . . . . . 65
2.19 Cross-correlation coefficients for Grey 33 data sample . . . . . 66
2.20 Cross-correlation coefficients for None 7 data sample . . . . . 66
2.21 A radial basis function neural networks sum-squared network error for 89 epochs . . . . . 68
2.22 A radial basis function neural networks sum-squared network error for 29 epochs . . . . . 70
2.23 Plot of the training errors for a back-propagation network with 20 hidden and 4 output neurons . . . . . 71
2.24 Plot of the training errors for a back-propagation network with 10 hidden and 4 output neurons . . . . . 71
2.25 Plot of the autocorrelation (unbiased estimate) of High 42 - detector A . . . . . 72

2.26 Plot of the autocorrelation (unbiased estimate) of High 42 - detector B . . . . . 73
2.27 Plot of the autocorrelation (unbiased estimate) of High 42 - detector C . . . . . 73
2.28 Plot of the autocorrelation (unbiased estimate) of Low 15 - detector A . . . . . 74
2.29 Plot of the autocorrelation (unbiased estimate) of Low 15 - detector B . . . . . 74
2.30 Plot of the autocorrelation (unbiased estimate) of Low 15 - detector C . . . . . 75
2.31 A radial basis function neural networks sum-squared network error for 29 epochs . . . . . 75
2.32 A radial basis function neural network's sum-squared network error for 44 epochs . . . . . 77
2.33 A multilayer perceptron neural network's sum-squared network error for 201083 epochs . . . . . 79

3.1 The normalized data of the normal transformer . . . . . 99
3.2 The normalized data of the faulty transformer . . . . . 100
3.3 The principal components of the normal transformer . . . . . 100
3.4 The principal components of the faulty transformer . . . . . 101
3.5 The variance of the principal components of the normal transformer . . . . . 101
3.6 The variance of the principal components of the faulty transformer 102
3.7 The correlation coefficients matrix of the normal transformer . . 102
3.8 The correlation coefficients matrix of the faulty transformer . . 103
3.9 Comparison of the variance of the principal components for the normal and faulty transformers . . . . . 103
3.10 Catalytic cracker variables block diagram . . . . . 105
3.11 Variables: v36-v59 . . . . . 107
3.12 Normalised data: v36-v59 . . . . . 107
3.13 Principal components: v36-v59 . . . . . 108
3.14 Variability explained (%) of principal components: v36-v59 . . . 109
3.15 Component scores: v36-v59 . . . . . 110
3.16 Variables: v73-v84 . . . . . 110
3.17 Normalised data: v73-v84 . . . . . 111
3.18 Principal Components: v73-v84 . . . . . 112
3.19 Variability explained (%) of principal components: v73-v84 . . . 113
3.20 Component scores: v73-v84 . . . . . 114
3.21 Variables: v85-v95 . . . . . 114
3.22 Normalised data: v85-v95 . . . . . 116
3.23 Principal components: v85-v95 . . . . . 117
3.24 Variability explained (%) of principal components: v85-v95 . . . 118

3.25 Component scores: v85-v95 . . . . . 118
3.26 The first 5 eigenvalues of the High, Low, Grey and None oil mist data . . . . . 120
3.27 The first 7 eigenvalues of the High and Low oil mist data . . . . 121
3.28 Neural network training errors for High and Low eigenvalues . . 122

4.1 Signal 1 . . . . . 129
4.2 Signal 2 . . . . . 130
4.3 The fast Fourier transform of Signal 1 . . . . . 131
4.4 The fast Fourier transform of Signal 2 . . . . . 131
4.5 The wavelet transform of signal 1 . . . . . 132
4.6 The wavelet transform of signal 2 . . . . . 132
4.7 An idealised time-frequency plane decomposed with windowed Fourier analysis . . . . . 135
4.8 An idealised time-frequency plane decomposed with wavelets . . 136
4.9 The Daubechies D4 wavelet . . . . . 138
4.10 The Haar wavelet . . . . . 139
4.11 Signal S passes through two complimentary filters and emerges as two signals . . . . . 143
4.12 The Subband Coding algorithm . . . . . 147
4.13 High alarm data . . . . . 150
4.14 Low alarm data . . . . . 151
4.15 Grey alarm data . . . . . 152
4.16 None alarm data . . . . . 152
4.17 Scatter plot of High1 beams A and B . . . . . 153
4.18 Scatter plot of High1 beams A and C . . . . . 153
4.19 Scatter plot of High1 beams B and C . . . . . 154
4.20 Scatter plot of Low1 beams A and B . . . . . 154
4.21 Scatter plot of Low1 beams A and C . . . . . 155
4.22 Scatter plot of Low1 beams B and C . . . . . 155
4.23 Wavelet coefficients of a typical sample of High alarm data using Daubechies 5 mother wavelet . . . . . 156
4.24 Wavelet coefficients of a typical sample of Low alarm data using Daubechies 5 mother wavelet . . . . . 157
4.25 Scatter plot of Wavelet coefficients of High1 beams A and B . . 158
4.26 Scatter plot of Wavelet coefficients of High1 beams A and C . . 159
4.27 Scatter plot of Wavelet coefficients of High1 beams B and C . . 159
4.28 Scatter plot of Wavelet coefficients of Low1 beams A and B . . . 160
4.29 Scatter plot of Wavelet coefficients of Low1 beams A and C . . . 160
4.30 Scatter plot of Wavelet coefficients of Low1 beams B and C . . . 161
4.31 Training errors for backpropagation neural network with 3 hidden neurons . . . . . 161

4.32 Training errors for backpropagation neural network with 6 hidden neurons . . . . . 162
4.33 Training errors for backpropagation neural network with 9 hidden neurons . . . . . 162
4.34 Normalised High alarm data . . . . . 163
4.35 Normalised Low alarm data . . . . . 164
4.36 Normalised Grey alarm data . . . . . 165
4.37 Normalised None alarm data . . . . . 165
4.38 Training errors for backpropagation neural network with 3 hidden neurons . . . . . 166
4.39 Training errors for backpropagation neural network with 6 hidden neurons . . . . . 166
4.40 Training errors for backpropagation neural network with 9 hidden neurons . . . . . 167
4.41 Training errors for backpropagation neural network with 12 hidden neurons . . . . . 167
4.42 Training errors for backpropagation neural network with 18 hidden neurons . . . . . 168
4.43 Training errors for backpropagation neural network with 24 hidden neurons . . . . . 168

5.1 Block diagram of an ideal intelligent monitoring and prediction system . . . . . 175
5.2 Anatomy of an ideal expert system . . . . . 178
5.3 A learning expert system . . . . . 197
5.4 An intelligent monitoring and prediction system . . . . . 203

# List of Tables

2.1 Neural network target vector for the High, Low, Grey and None alarm conditions . . . . . 59

3.1 Scores on the principal components: v73-v84 . . . . . 115
3.2 Scores on the principal components: v85-v95 . . . . . 119
3.3 Neural network target vector for High and Low alarm conditions 121

4.1 Neural network prediction accuracy . . . . . 169

# Chapter 1

# Introduction

## 1.1 Aims and Objectives

The aim of this research was to develop an intelligent monitoring and prediction system. The work is aimed at developing novel methodologies for real world time series prediction and classification problems. Three sets of industrial process data are examined: infra-red scatter data (oil mist detection), catalytic cracker data (oil refinery) and power transformer data.

Many areas were investigated during the course of this work. Principal component analysis, the wavelet transform and neural networks were all examined as techniques for information processing and as methods of extracting the key features from the raw data. Neural networks were investigated for pattern classification of the time series data. Further work looked at the architecture and design of rule based expert systems and evolutionary computation for information processing and extraction.

## 1.2 Methodologies

### 1.2.1 Neural Networks

The development of artificial neural networks began with the work of McCulloch and Pitts in 1943. In their paper McCulloch and Pitts described a logical calculus of neural networks [MP43]. In 1948 Weiner's book described some important concepts for control, communications and signal processing [Wie48]. Weiner explained the linkage between statistical mechanics and learning.

An explicit statement of a physiological learning rule for synaptic modification was presented for the first time by Hebb in 1949 [Heb49]. Hebb proposed that the connectivity of the brain is continually changing as an organism performs different functional tasks and that neural assemblies are created by such changes. Hebb's postulate of learning states that the effectiveness of a variable synapse between two neurons is increased by the repeated activation of one neuron by the other across that synapse.

The first paper to attempt to use a computer simulation to test a well formulated neural theory based on Hebb's postulate of learning was published in 1956 [RHHD56]. The simulation results reported in that paper clearly showed that inhibition needed to be added for the theory to actually work. In the same year Uttley demonstrated that a neural network with modifiable synapses may learn to classify simple sets of binary patterns into corresponding classes [Utt56]. Uttley introduced the so called "leaky integrate and fire neuron", which was later formally analysed by Caianiello [Cai61]. In later work, Uttley

[Utt79] hypothesised that the effectiveness of a variable synapse in the nervous system depends on the statistical relationship between the fluctuating states on either side of that synapse, thereby linking up with Shannon's information theory.

In 1954, Gabor proposed the idea of a nonlinear adaptive filter [Gab54]. He later went on to build such a machine, where learning was accomplished by feeding samples of a stochastic process into the machine, together with the target function that the machine was expected to produce. Work on associative memory followed a paper published in 1956 [Tay56]. In 1969 a paper on nonholographic associative memory proposed two network models: a simple optical system realising a correlation memory and a closely related neural network suggested by the optical memory (Willshaw et al [WBLH69]).

An issue of particular concern in the context of neural networks is that of designing a reliable network with neurons that may be viewed as unreliable components. This important problem was solved by von Neumann [vN56] using the idea of redundancy. This motivated others to suggest the use of a distributed redundant representation for the neural network [WC63].

A new approach to the pattern recognition problem was proposed by Rosenblatt in his work on the perceptron [Ros58]. The first proof of Rosenblatt's perceptron convergence theorem was published in 1960 [Ros60]. The least mean square (LMS) algorithm was used to formulate the Adaline (adaptive linear element) in 1960 [WH60]. The difference between the perceptron and the Adaline lies in its training procedure. One of the earliest trainable layered

neural networks with multiple adaptive elements was the Madaline structure proposed in 1962 [Wid62]. An exposition on linearly separable patterns in hypersurfaces was given in Nilsson [Nil65].

During the 1960s it seemed as if neural networks could do almost anything. However, a book by Minsky and Papert in 1969 [MP69] demonstrated that there are fundamental limits on what one layer perceptrons can compute. They also stated that there was no reason to believe that any of the virtues of one layer perceptrons carry over to the multilayered version.

An important problem encountered in the design of a multilayer perceptron is the credit assignment problem, that is, the problem of assigning credit to hidden neurons in the network [Min61]. By the late 1960s most of the ideas and concepts necessary to solve the credit assignment problem were already formulated. However, in part due to the paper by Minsky and Papert, interest in neural networks declined in the 1970s.

In the 1980s, major contributions to the theory and design of neural networks were made on several fronts and these led to a resurgence of interest in the field. One of these was Hopfield in 1982, who used the idea of an energy function to formulate a new way of understanding the computation performed by recurrent networks with symmetric synaptic connections [Hop82]. This particular class of neural networks, known as Hopfield networks, may not be very realistic models for neurobiological systems, however the principle they embody, namely that of storing information in dynamically stable networks, is profound.

In 1986 the development of the backpropagation algorithm was reported by Rumelhart et al [RHW86]. The same year, work by Rumelhart and McClelland became a major influence in the use of the back propagation algorithm, which has emerged as the most widely used learning algorithm for the training of multilayer perceptrons [RM86].

Linsker in 1988 described a new principle for self-organisation in a perceptual network [Lin88]. The principle is designed to preserve maximum information about input activity patterns, subject to such constraints as synaptic connections and synaptic dynamic range.

In 1988, Broomhead and Lowe described a procedure for the design of layered feedforward networks using radial basis functions, which provide an alternative to multilayer perceptrons [BL88]. This led to a great deal of research effort linking the design of neural networks to an important area in numerical analysis and also linear adaptive filters.

Today, neural networks are well established as an interdisciplinary subject with deep roots in engineering, the physical sciences, mathematics, neuroscience and psychology. Over the coming years, neural networks will continue to grow in theory, design and application.

### 1.2.2 Wavelet Transforms

Wavelet analysis shows many different origins in the history of mathematics [Mey92]. Much of the work was performed in the 1930s and at the time the separate efforts did not appear to be parts of a coherent theory. Before 1930, the main branch of mathematics leading to wavelets began with Fourier's theories of frequency analysis, now often known as Fourier synthesis. Fourier asserted that any $2\pi$-periodic function $f(x)$ is the sum of its Fourier series.

Fourier series convergence and orthogonal systems, led to the notion of scale analysis (as opposed to frequency analysis). That is, analysing $f(x)$ by creating mathematical structures that vary in scale. This is achieved by constructing a function, shifting it by some amount and changing its scale. Apply that structure in approximating a signal. Now repeat the procedure. Take that basic structure, shift it and scale it again. Apply it to the same signal to get a new approximation. And so on. Because this sort of scale analysis measures the average fluctuations of the signal at different scales it is less sensitive to noise.

The first mention of Wavelets is made by Haar in 1910. One property of the Haar wavelet is that it has compact support, which means that it vanishes outside a finite interval. The Haar wavelet, however, is not continuously differentiable which therefore limits its applications [Dau92].

In the 1930s, several groups working independently researched the representations of functions using scale-varying basis functions. By using a scale-

varying basis function called the Haar basis function Paul Levy (a 1930s physicist) investigated Brownian motion, a type of random signal. Levy found the Haar basis function superior to the Fourier basis functions for studying small complicated details in Brownian motion [Mey92].

Another 1930s research effort involved computing the energy of a function. The computation produced different results if the energy was concentrated around a few points or distributed over a larger interval. This result disturbed the scientists because it indicated that energy might not be conserved. The researchers discovered a function that can vary in scale and can conserve energy when computing the functional energy. Their work provided Marr with an effective algorithm for numerical image processing using wavelets in the early 1980s [Chu92].

Between 1960 and 1980 the mathematicians Weiss and Coifman studied the simplest elements of function space, called atoms, with the goal of finding the atoms for a common function and finding the "assembly rules" that allow the reconstruction of all the elements of the function space using these atoms [Dau92]. In 1980, Grossmann and Morlet, broadly defined wavelets in the context of quantum physics. This work provided a way of thinking for wavelets based on physical intuition [GM84].

Work on wavelets was given further impetus by the work of Mallat in digital signal processing. Mallat discovered some relationships between quadrature mirror filters, pyramid algorithms and orthonormal wavelet bases [Mal89a, Mal89b, Mal89c]. Meyer then constructed the first non-trivial wavelets. Unlike

the Haar wavelets, the Meyer wavelets are continuously differentiable; however they do not have compact support. Later Daubechies used Mallat's work to construct a set of orthonormal basis functions that are perhaps the most elegant and have become the cornerstone of wavelet applications today [Dau88].

### 1.2.3 Genetics Based Machine Learning

The theoretical foundations for Genetics Based Machine Learning (GBML) systems was laid by Holland in 1962 [Hol62]. His outline for adaptive systems theory paid special attention to the role of program replication as a method of emphasising past programs. Although subsequent applications of genetic algorithms in the 1960's largely emphasised search and optimisation, the theoretical underpinning was not so restricted.

With this theoretical foundation and the recognition of the fundamental role of recombination [Hol65], more concrete suggestions emerged for the creation of a sequence of increasingly complex schemata processors [Hol71]. In this paper Holland proposed four prototype systems. Prototype I was to be a stimulus-response processor which would link environmental schemata (conditions) with particular action effectors. Prototype II was designed to extend prototype I by adding internal effectors (internal states) and prototype III was to build on prototypes I and II by including explicit environmental state prediction (a model of the real world) and an internal evaluation mechanism. Prototype IV was to extend the other prototypes by incorporating the capability to modify its own effectors and detectors, thereby permitting a greater (or perhaps lesser) range of data detection and a larger behavioural repertoire.

Holland's pioneering book *Adaptation in Natural and Artificial Systems* proposed a broadcast language [Hol75]. The broadcast language called for the creation of broadcast units (production rules) over a 10 letter alphabet. This

alphabet added a number of wildcards both single and multiple match characters to an underlying binary alphabet. Additionally, a fundamental punctuation mark, a persistence symbol (causing continued broadcast of a message), and a quotation character (causing the next symbol to be taken literally) would have provided sufficient power for computational completeness and representational completeness. The proposal for the broadcast language was instrumental in unifying the earlier suggestions for schemata processors by theoretically permitting a consistent representation of all operators, data and rules or instructions.

Holland's book showed how the evolutionary process can be used to solve problems by means of a highly parallel technique that is now called the genetic algorithm. The genetic algorithm transforms a population of individual objects, each with an associated value of fitness, into a new generation of the population, using the Darwinian principle of survival and reproduction of the fittest and analogues of naturally occurring genetic operations such as crossover (sexual recombination) and mutation [Gol89].

The first practical application of genetics based machine learning was a classifier system in 1978 [HR78]. This system, called Cognitive System Level One (CS-1), was trained to learn two maze running tasks. It used a performance system with a message list and simple string rules called classifiers, a genetic algorithm comprised of reproduction, crossover and mutation and an epochal learning mechanism (where reward was apportioned to all classifiers active between successive payoff events). This last learning mechanism has largely been

supplanted by another mechanism, called a bucket brigade, in later systems.

The classifier system represented a considerable extension of the complexity of the structures undergoing adaptation. The genetic classifier system is a cognitive architecture that allows the adaptive modification of a set of if-then rules [Koz92, Koz94]. The architecture of the classifier system blends important features from the contemporary paradigms of artificial intelligence, connectionism and machine learning.

## 1.3 The Industrial Data

The objective of the research was to develop a novel methodology for non-linear time series prediction. The aim was to produce an alarm system for intelligent monitoring and prediction of real-world processes where limited data is available.

Initial work focused on a problem for *Shell UK* involving infra-red scatter data used for the detection of oil mist. An array of three sensors, that are spatially distributed at known locations around a common reference point, are used to detect oil mist. The sensors collect signals from sources in their field of view. Depending on the sensor characteristics and the path of propagation, the source waveforms undergo deterministic and/or random modifications. The outputs of these sensors consists of the desired source signals and unwanted noise components. The alarm system is aimed at use on unmanned offshore oil installations to warn of developing problems. If the alarm is false it is a costly waste of time to send out engineers by helicopter unnecessarily. On the other hand, not correctly determining a real alarm in time could have catastrophic consequences. The aim was to use neural networks to classify the data into one of four alarm conditions: High, Low, Grey and None.

Further work examined data, provided by *Shell UK*, from a catalytic cracker used in an oil refinery. The data consists of ninety-five variables who's physical significance has not been provided. The aim is to use the last sixty variables to predict the first thirty five variables concentrating on two variables ini-

tially. Because this is industrial process data, there are lots of "drop-outs" and "spikes" which need to be removed before any modelling is done. The data provided includes: the input feeds (a mixture of hydrocarbons), the related responses of the system, explanatory variables, related process parameters and other process parameters.

The aim was to identify the key features of the catalytic cracker data in order to reduce the dimensionality of the problem. Once the key features had been extracted from the data, the use of neural networks and learning expert systems was examined for classifying these features for time series prediction.

The third set of data was provided by the *NGC* and is used to monitor the condition of a power transformer in service. This data can be classified into two types. The first is the common variables related to the transformer operation and the second is the dissolved gas analysis. In modern industrial societies the maintenance of uninterrupted power supplies is of paramount importance. The provision of adequate monitoring systems for power systems is essential to avoid damage due to failures of apparatus' operation and degrading of their conditions. A power systems monitoring system should ensure that faulty equipment is identified and afterwards removed from service in the minimum of time so that; the faulty equipment is isolated and in order to limit the damage to equipment due to overheating, excessive mechanical forces, etc.

A method is presented, based on principal component analysis, to determine the key variables of a power transformer and the interdependence of those variables. An index number has also been defined to monitor any change in

the distribution of the variance of the principal components of the transformer, which may be used for the purpose of condition monitoring.

As described at the start of this section the aim of this research was to develop novel methodologies for time series prediction and classification problems involving real world data. This real world data included: oil mist vapour, catalytic cracker data and power transformer data. A number of techniques have been investigated to extract the key features from the real world data in order to reduce the complexity of the alarm classification problem, without at the same time, losing any important information from the data. The data, raw and preprocessed, was then used to investigate several different neural network paradigms for pattern classification, including the multilayer perceptron neural network and the radial basis function neural network. Evolutionary computation and rule based expert systems were also examined as information processing and extraction methodologies.

# Chapter 2

# Neural Networks for Information Extraction

## 2.1 Introduction

While standard digital computing is good at tasks which require quick calculations, it is bad at tasks such as pattern recognition which involve lots of information and/or noisy data. Biological neural networks on the other hand, while being poor at quick calculations are good at problems that involve lots of data or noisy data.

The silicon logic gates used in computers are several orders of magnitude faster than biological neurons. The brain however compensates for the relatively slow operation of individual neurons by having a huge number of neurons, with each neuron interconnected with many other neurons; estimates put the

number of neurons in the human brain at ten billion and the number of connections between them at ten trillion [Ros92]. The brain carries out operations in parallel, and in a distributed manner, that is, with many neurons involved in any single function and no single neuron being uniquely involved in any way. Artificial neural networks, usually referred to as *neural networks*, mimic in some way the operation of the brain to perform *parallel distributed processing* [Ros92].

One of the advantages of neural networks is their ability to *learn* and therefore *generalise*. Generalisation is the ability of the neural network to successfully classify patterns that have not been presented during training (learning).

The objective of *pattern recognition* is classification, that is, given an input signal, can it be analysed to provide a meaningful categorisation of its data content. Pattern recognition can be considered as a two stage process.

The first stage is *feature extraction*. A feature is a measurement taken on the input pattern to be classified, that will provide a definite characteristic of the input. Feature extraction is usually the most difficult part of the pattern recognition problem. The measurements taken of the input pattern form a *feature vector*. If the feature vector consists of $n$ measurements then it creates an $n$ dimensional *feature space* [Hay94, JB94].

The second stage is *classification*. The classifier is provided with the list of measured features and its task is to decide which type of class category they match most closely. Classification is usually achieved using distance metrics and probability theory.

## 2.2 Neural Networks

### 2.2.1 The Neuron

The basic building block of the neural network is the *neuron*, which is modelled on the biological neuron. A biological neuron adds up its inputs and produces an output if the sum is greater than its *threshold* value. A neuron is an information-processing unit that consists of three basic elements:

1. A set of connecting links known as *synapses*, each of which is characterised by a *synaptic weight*. A signal $x_j$ at the input of synapse $j$ connected to neuron $k$ is multiplied by the synaptic weight $w_{kj}$. The weight can be either positive or negative depending on whether it excites or inhibits the synapse.
2. An *adder* for summing the input signals, weighted by the respective synapses of the neuron.
3. An *activation function, sigmoid function*, also known as a *thresholding function*, or *squashing function* that limits the output amplitude of the neuron to a certain range. The output is normally limited to either the range [0,1] or [-1,1].

The model of a neuron also has an externally applied bias, $\theta_k$, which can be either positive or negative. The bias has the effect of raising or lowering the *activation* of the neuron [Hay94]. Figure 2.1 shows the basic model of a neuron.

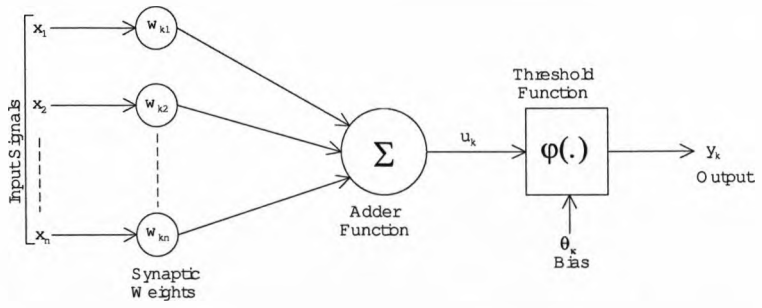


Figure 2.1: The basic neuron

The neuron, $k$, is defined by the two equations:

$$u_k = \sum_{j=1}^{n} w_{kj} x_j$$

and

$$y_k = \varphi(u_k - \theta_k)$$

where $x_1, x_2, \ldots, x_n$ are the input signals; $w_{k1}, w_{k2}, \ldots, w_{kn}$ are the synaptic weights of the neuron $k$; $u_k$ is the output of the adder; $\theta_k$ is the neurons bias; $\varphi(\cdot)$ is the *activation* function; and $y_k$ is the output signal of the neuron. The activation function defines the output of a neuron in terms of the activity level at its input.

The significance of neural networks lies in their ability to learn and to improve their performance through this learning process. Neural networks learn through an iterative process of adjustments to its synaptic weights and thresholds. Learning can be regarded as a three step process [Hay94]:

1. The neural network is stimulated by an environment.

2. The neural network undergoes changes as a result of this stimulation.

3. Because of these changes the neural network responds in a new way to the environment.

A signal $x_j$ at the input of synapse $j$ connected to neuron $k$ is multiplied by the synaptic weight $w_{kj}$. Let $w_{kj}(n)$ denote the value of $w_{kj}$ at time $(n)$. At time $n$ an adjustment $\Delta w_{kj}(n)$ is applied to the synaptic weight $w_{kj}(n)$, yielding the updated weight $w_{kj}(n+1)$. Thus:

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n). \tag{2.2.1}$$

A number of different algorithms exist for determining the adjustment $\Delta w_{kj}$ to the synaptic weight $w_{kj}$. One such algorithm is *error-correction learning.* When a input is applied to a neuron $k$ its actual output $y_k(n)$ will differ from its desired response $d_k(n)$, and an error signal may be defined as:

$$e_k(n) = d_k(n) - y_k(n). \tag{2.2.2}$$

The purpose of error-correction learning is to get the response of each output neuron to approach its desired response. This is achieved by minimising a *cost function* based on the error signal $e_k(n)$. Error correction learning then becomes an optimisation problem. The *mean square error* is a commonly used criterion for reducing the cost function, it is defined as the mean square value of the sum of squared errors:

$$J = E\left[\frac{1}{2}\sum_k e_k^2(n)\right], \tag{2.2.3}$$

where $E$ is the *statistical expectation operator* and the summation is over all output layer neurons in the network. Normally, because the statistics of the

underlying processes are unknown, an approximate solution to this optimisation problem is adopted using the *instantaneous value* of the sum of squared errors:

$$\mathcal{E}(n) = \frac{1}{2}\sum_{K} e_k^2(n). \tag{2.2.4}$$

By adjusting the synaptic weights of the network, $\mathcal{E}(n)$ can be minimised, thus optimising the network. Hence for the error correction learning rule the adjustment $\Delta w_{kj}(n)$ is given by:

$$\Delta w_{kj}(n) = \eta e_k(n) x_j(n), \tag{2.2.5}$$

where $\eta$ is the *learning rate*. Hence the adjustment made to a synaptic weight is proportional to the product of its error signal and its input signal.

Because error correction learning behaves like a *closed loop feedback system*, care has to be taken in choosing a value of $\eta$ to ensure the stability of the learning process. The learning rate affects both the rate of convergence and the convergence itself. Small learning rates provide a smooth learning process, however the system may take a long time to converge to a stable solution. Large learning rates mean that the learning process is accelerated, but the danger arises that the learning process may diverge and the system become unstable.

The error function, $\mathcal{E}$, is an energy function which represents the amount by which the output of the network differs from the expected output. Large differences correspond to large energies and small differences correspond to small energies. If the error function is plotted against the synaptic weights of

the network a multidimensional *error (energy) surface* results.

Starting from an arbitrary point on the error surface, determined by the initialisation of the networks weights, the aim of error-correction learning is to move gradually toward a global minimum. However, a problem with the error-correction learning rule is that it is possible for it to get trapped at some local minima on the error surface and therefore never reach the global minimum [Hay94, Was93].

### 2.2.2 The Multilayer Perceptron Neural Network

There are many neural network architectures. Neural networks learn through an iterative process of adjustments to its synaptic weights and thresholds. The architecture of a neural network depends upon the *learning algorithm* used to train the network. Two or more of the neurons described above can be combined together to form a layer. This forms the simplest neural network the *single-layer perceptron.* A single layer perceptron has an *input layer* of source nodes that maps onto an *output layer* of neurons. The input layer of source nodes performs no computation and is therefore not counted as a layer of the network.

A single-layer perceptron associates an output pattern with an input pattern. By modifying the synaptic weights of the network information can be stored in the network. The limitation of the single-layer perceptron is that it can only classify objects that are linearly separable in the pattern space as shown in Figure 2.2. To overcome this obstacle two or more layers can be

combined to form the *multilayer perceptron* [JB94].

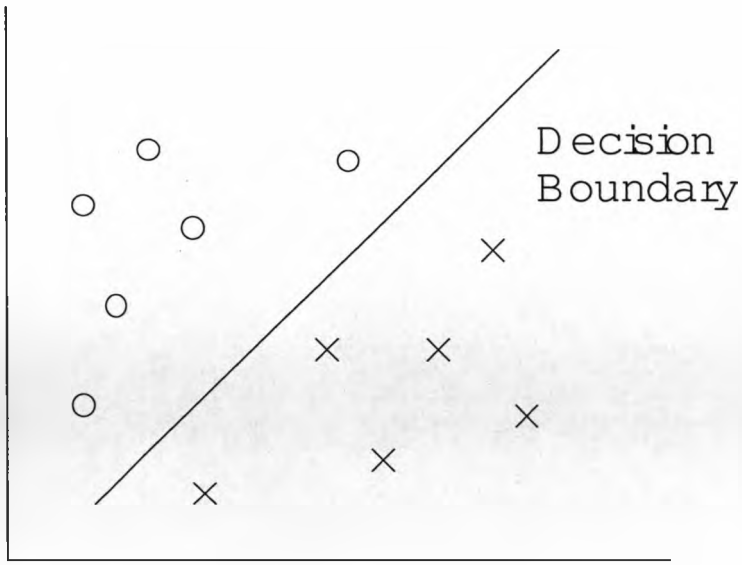


Figure 2.2: Two dimensional pattern space

The multilayer perceptron has an input layer of source nodes, an output layer of neurons and one or more *hidden layers* of neurons between the input and output layers. The hidden layer(s) of neurons allow the multilayer perceptron to learn more complex tasks. The addition of one or more hidden layers gives the network a global perspective due to the extra set of synaptic connections and the extra dimension of neural interactions, thus enabling the network to extract higher-order statistics. If the size of the input layer is large this ability to extract higher-order statistics can be extremely useful [Hay94].

Determining the size of the network, i.e. the number of neurons in the hidden layer(s) and the number of hidden layers, is critical to the performance of the network. A network that is too small may not train to acceptable accuracy. A network that is too large will be unnecessarily slow and expensive.

In addition it may require an excessively large training set to generalise well. The optimal size of a network is problem dependent and there are no hard and fast rules for determining it. Rather it is down to the experience of the engineer and trial and error to produce the smallest network that provides the desired accuracy on both the training and test data sets [Hay94].

The multilayer perceptron uses *hyperplanes*, defined by the weighted sums $\sum_n w_{kj}x_j$, to partition the pattern space, thus allowing the classification of patterns that are not linearly separable [JB94].

A multilayer perceptron network, with a single hidden layer, is shown in Figure 2.3. The source nodes in the input layer of the network supply respective elements of the input vector to the inputs of the hidden layer. The output signals of the hidden layer then provide the inputs to the output layer. The outputs of the neurons in the final layer provide the overall response of the network to the input vector.

The multilayer perceptron is trained using the *back-propagation of errors* algorithm. Back-propagation of errors requires two passes through the layers of the network. Firstly, a *forward pass*, where the input training vector is applied to the network and its effects propagate through the different layers of the network, until the network produces an output response. During the forward pass the neural networks synaptic weights are fixed. Secondly a *backward pass*, during this pass the synaptic weights are all adjusted in accordance with the error correction rule. The response of the network is compared to the desired output of the network to produce an *error signal.* The error signal is then

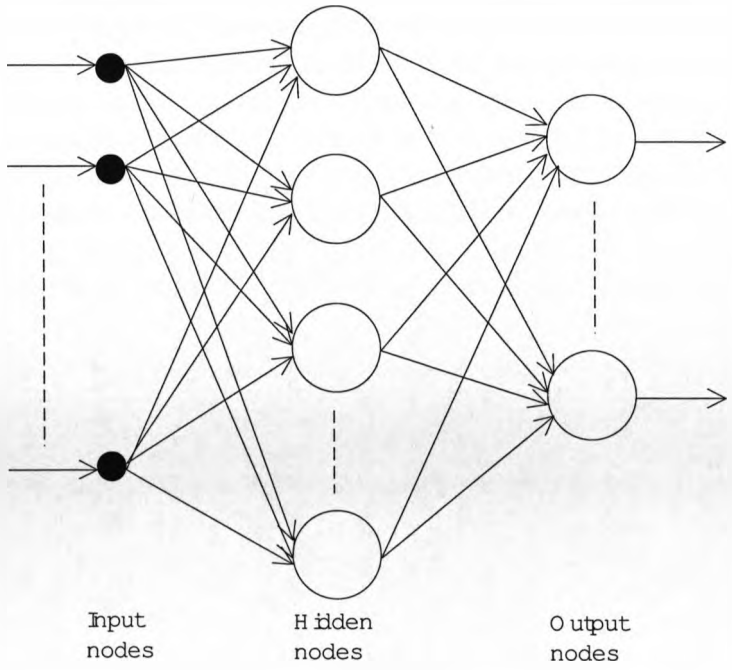


Figure 2.3: A multilayer perceptron neural network

propagated backwards through the network, that is, against the directions of the synaptic connections. The error signal is used to adjust the neurons weights in order to move the actual response of the network closer to the desired response [Hay94].

A problem arising in the training of multilayer perceptron networks is that known as *credit assignment.* For the network to train properly it is necessary to know which neurons in the hidden layer have the credit, or blame, for a particular outcome, thus allowing the correct synaptic weights to be adjusted accordingly. To allow the error signal to update the weights of the neurons in the hidden layer, it is necessary that these neurons have a nonlinear characteristic, and that it is also differentiable [Hay94]. A commonly used nonlinearity

is the *sigmoidal* nonlinearity, defined by:

$$y_j = \frac{1}{1 + e^{-v_j}}, \tag{2.2.6}$$

where $v_j$ is the net internal activity level of neuron $j$ and $y_j$ is the output of the neuron. The sigmoidal function is shown in Figure 2.4.

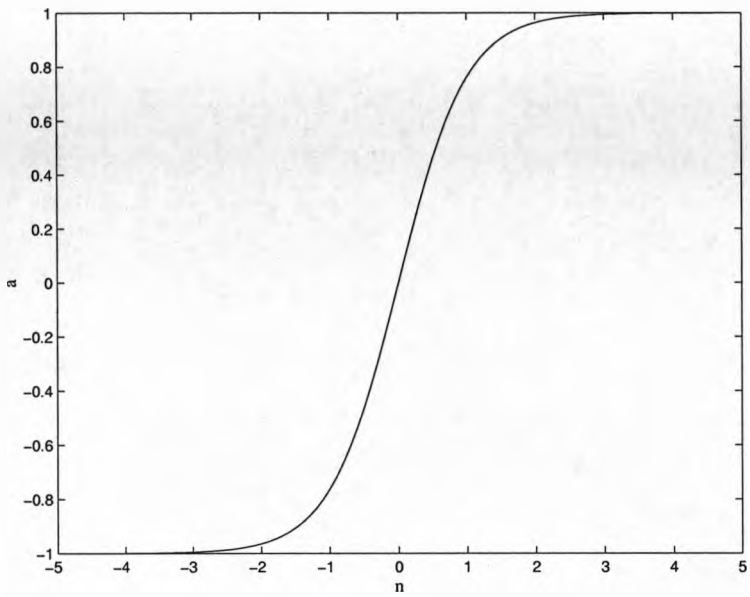


Figure 2.4: Hyperbolic tangent sigmoid transfer function

The error signal at the output of neuron $j$ at iteration $n$ is given by:

$$e_j(n) = d_j(n) - y_j(n), \tag{2.2.7}$$

where neuron $j$ is an output node and $d_j(n)$ is the desired response of neuron $j$ at iteration $n$. The *instantaneous sum squared error* of the network is:

$$\mathcal{E}(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n), \tag{2.2.8}$$

where the set $C$ includes all the neurons in the output layer of the network. If $N$ denotes the total number of training patterns then the normalised (with

respect to the training set size $N$) average squared error is:

$$\mathcal{E}_{av} = \frac{1}{N}\sum_{n=1}^{N}\mathcal{E}(n). \tag{2.2.9}$$

The sum squared error of the network is a function of its free parameters, that is its weights and biases. The task of the back-propagation algorithm is to minimise the cost function $\mathcal{E}_{av}$ by adjusting the weights and biases of the network. These adjustments to the networks free parameters are made for each training pattern in the training set.

If neuron $j$ is fed by a set of signals from the previous layer, the net internal activity level $v_j(n)$ produced at the input of the nonlinearity associated with neuron $j$ is:

$$v_j(n) = \sum_{i=0}^{p} w_{ji}(n)y_i(n), \tag{2.2.10}$$

where $p$ is the total number of inputs, excluding the bias, applied to neuron $j$. The signal $y_j(n)$ appearing at the output of neuron $j$ at iteration $n$ is:

$$y_j(n) = \varphi(v_j(n)), \tag{2.2.11}$$

where $\varphi_j(\cdot)$ is the activation function of neuron $j$.

The back-propagation algorithm applies a correction, $\Delta w_{ji}(n)$, to the synaptic weight, $w_{ji}(n)$, which is proportional to the instantaneous gradient, $\partial\mathcal{E}(n)/\partial w_{ji}(n)$. The correction to the synaptic weight is given by:

$$\Delta w_{ji}(n) = \eta\delta_j(n)y_i(n). \tag{2.2.12}$$

The gradient indicates the required changes in the synaptic weights. The gradient, $\delta_j(n)$, for output neuron $j$ is given by the product of the error signal,

$e_j(n)$, and the derivative, $\varphi'(v_j(n))$, of the activation function:

$$\delta_j(n) = e_j(n)\varphi'_j(v_j(n)). \tag{2.2.13}$$

If neuron $j$ is an output neuron then it is a simple matter to calculate its error signal because its desired response is known. Equation 2.2.7 is used to compute the error signal of the neuron and equation 2.2.13 is to calculate its gradient.

If neuron $j$ is a hidden neuron things become more difficult. Hidden neurons share responsibility for any errors at the output of the network. Because there is no known desired response for the hidden neuron, its error signal is determined recursively in terms of the error signals of all the neurons to which the hidden neuron is directly connected. Back-propagation of errors allows the hidden neurons to be penalised, or rewarded, for their contribution to this output. The gradient for a hidden neuron $j$ is given by:

$$\delta_j(n) = \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n). \tag{2.2.14}$$

The first term in equation 2.2.14, $\varphi'_j(v_j(n))$, is the derivative of the activation function of the hidden neuron. The second term, that is, the summation over $k$, is the product of the gradients of the neurons in the next layer that are connected to neuron $j$ with the synaptic weights associated with these connections.

The back-propagation algorithm then requires two passes of computation. A forward pass and a backward pass.

In the forward pass the synaptic weights remain fixed. An input vector is applied to the source nodes and propagates forward to the hidden layer and then to the output layer. The output is then compared to the desired response to produce an error signal.

The backward pass starts at the output layer by passing the error signal back through each layer of the network, recursively computing the gradient of each neuron. This recursive process allows the synaptic weights of the network to be updated.

A problem with the multilayer perceptron is that they require the input data to be presented many times, with a forward and backward pass for each training vector. Hence the repetition of the back-propagation of errors means that the network can be slow to settle to a stable solution [Hay94, Was93].

The back-propagation algorithm cycles through the training data as follows:

1. *Initialisation.* Start with a reasonable network configuration and set all the synaptic weights and threshold levels of the network to small random numbers that are uniformly distributed.
2. *Presentations of Training Examples.* Present the network with a set of training patterns and for each example in the training set perform the following forward and backward computations.
3. *Forward Computation.* With the input vector applied to the input layer and the desired response vector presented to the output layer, compute the activation potentials and function signals of the network by proceeding

through the network, layer by layer. Calculate the error signal.

4. *Backward Computation.* Compute the required adjustments to the synaptic weights of the network by proceeding backward, layer by layer. Adjust the synaptic weights of the layer.

5. *Iteration.* Iterate the computation by presenting new sets of training examples to the network until the free parameters of the network stabilise their values and the sum squared error of the network is at an acceptably small value.

### 2.2.3 Radial Basis Function Neural Networks

An enhancement to the multilayer perceptron uses *radial basis functions.* These are a set of generally nonlinear functions that are built up into one function that can partition the pattern space successfully. Figure 2.5 shows a radial basis function, where **u** represents the hidden neurons weight vector, **x** the neurons input vector and $h$ its' output. As the distance between the input vector and the weight vector decreases, the output increases. Thus a radial basis neuron acts as a detector that outputs 1 whenever the input vector, **x**, is identical to its weight vector, **u** [Was93].

While the *radial basis function (RBF) network* may require more neurons than a multilayer perceptron, they have the advantage that they can often be designed in a much shorter time than it takes to train a multilayer perceptron. The disadvantage of the RBF network is that once trained they are slower to

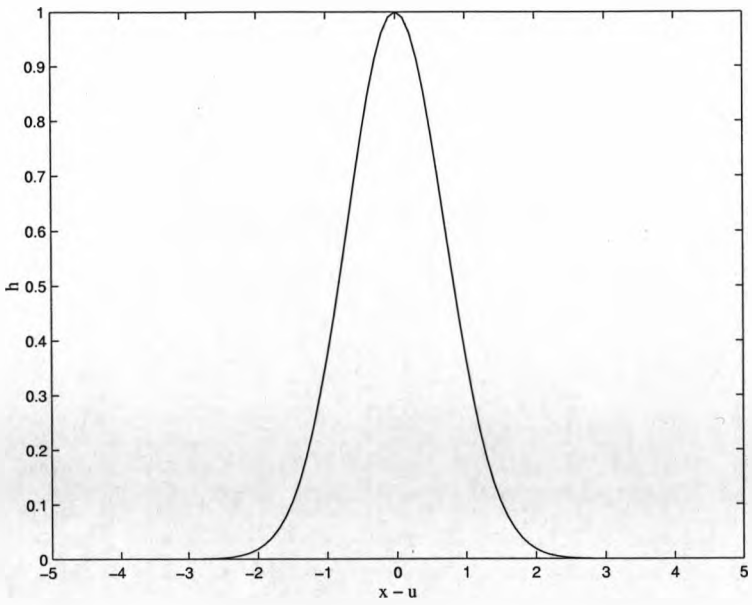


Figure 2.5: A radial basis function

use, that is, generalise, requiring more computation to perform a classification or function approximation. The reason for this slowness is due to the exponentially decaying localised nonlinearities which construct *local* approximations to nonlinear input-output mapping. Because of the local nature of each radial basis function the RBF network typically has a larger number of coefficients to compute. The multilayer perceptron, on the other hand, constructs *global* approximations to nonlinear input-output mapping, thereby encoding the characteristics of the training set into a more compact form [Hay94, Was93]. RBF networks work best when there are many training vectors available. Initially RBF networks will be used to attempt to solve the alarm problem under consideration.

RBF networks use multidimensional ellipsoids or *hyperellipsoids* to partition the pattern space. Thus learning for an RBF network is equivalent to finding

a surface in a multidimensional space that provides a best fit to the training data. The hyperellipsoids are defined by $\varphi(\|x-y)\|$. Where $\varphi$ is usually taken to be Gaussian, as defined in equation 2.2.15 and shown in Figure 2.5, and $\|\cdots\|$ denotes some distance measure, usually the Euclidean norm as defined in equation 2.2.16

$$\varphi(r) = e^{(\frac{-r^2}{2\sigma^2})}, \quad \sigma > 0, \quad r \geq 0, \tag{2.2.15}$$

where $\sigma$ determines the *range of influence*, that is, the width, of the radial basis function.

$$||x-y|| = \sum_i (x_i - y_i)^2, \tag{2.2.16}$$

where $y$ represents the centre of the hyperellipse.

The function $s$ in $k$-dimensional space, which partitions the space, has elements $s_k$ given by:

$$s_k = \sum_{j=1}^{m} w_{jk}\varphi(\|x - y_i\|). \tag{2.2.17}$$

The basic structure of the RBF network consists of three layers as shown in Figure 2.6. The input layer is made up of source nodes (the sensory units). The second layer is a hidden layer which must be of high enough dimension to partition the pattern space. The output layer supplies the response of the network to its inputs. The mapping from the input layer to the hidden layer is nonlinear, while the mapping from the hidden layer to the output layer is linear, of strength, $w_{jk}$ [JB94].

If an input is presented to the network each neuron in the hidden (radial basis) layer will output a value that depends on how close the input vector is

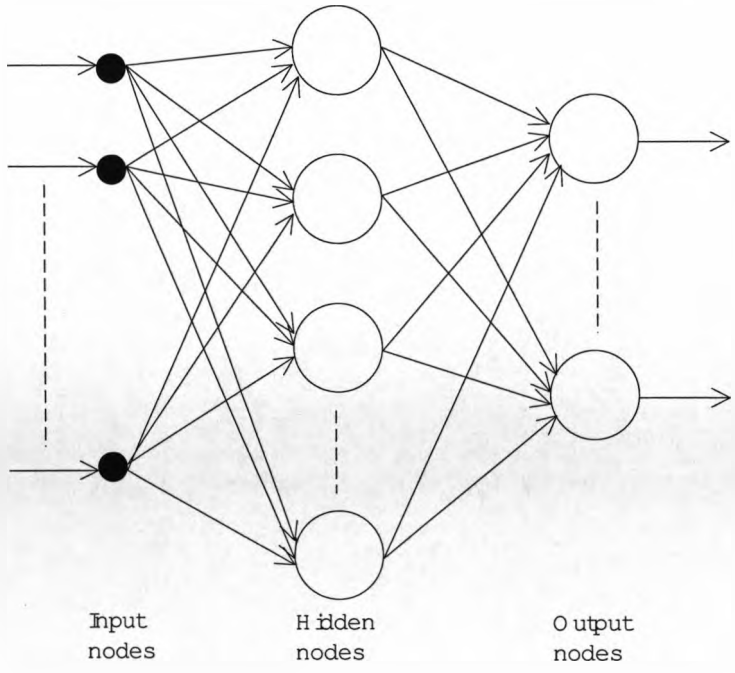


Figure 2.6: A radial basis function neural network

to its weight vector. If the input vector is not close to the weight vector of a particular neuron, then the output of that neuron will be small. These small outputs will have a negligible effect on the linear output neurons. If however, the input vector is very close to the weight vector of the neuron then that neuron will output a value close to 1. If a neuron has an output of 1 its output weights in the second layer pass their values to the linear neurons in the output layer [Hay94].

A set of radial basis functions is used to achieve the nonlinear mapping from the input layer to the hidden layer. In effect, the radial basis functions expand the inputs into a higher dimensional space where they become linearly separable. *Covers theorem on the separability of patterns* states that: a complex

pattern-classification problem cast in a high-dimensional space nonlinearly is more likely to be linearly separable than in a low-dimensional space.

If there is a radial basis function for each input to be classified then the network is guaranteed to produce a function that fits all the data points. However, this does have the disadvantage that noisy or anomalous data points will be classified and these will tend to cause distortion. The resulting distortion can result in the classification surface not being smooth and this can cause problems with generalisation. It can also lead to the network having a large number of neurons in the hidden (radial basis) layer if there are a large number of input vectors. The solution to this is to reduce the number of radial basis functions to a level at which an acceptable fit to the data is still achieved [Hay94].

The design of an RBFNN has two stages. Firstly, choose a set of basis functions that gives an acceptable fit to the data. Secondly, linear optimisation is used to set the strengths of the connections between the hidden and output layers, that is, $w_{jk}$ [nn:92].

The training of the weights, $w_{jk}$, is usually performed by linear optimisation of a cost function. Hence training pairs of input and target vectors are required. A typical cost function used is the sum squared error function as given in equation 2.2.4, though other cost functions are available. Since this is a linear optimisation problem, the radial basis function network can be trained more quickly than a multilayer perceptron [Low95].

Theoretical investigations and practical results have shown that the type of

nonlinearity, $\varphi(\cdot)$, is not vital to the performance of RBF networks. However the positioning of the centres of the radial basis functions is of critical importance and there are many alternatives for their determination [Hay94, CCG91].

A centre, and therefore a corresponding hidden layer neuron, could be located at each input vector in the training set. Because training vectors tend to occur in clusters, this method can result in more hidden layer neurons than are necessary and thus overfitting of the training data [Was93]. The result will be distortion, as described above, and longer training and generalisation times.

A second approach is to assume *fixed* radial basis functions, whose centres are chosen randomly from the training set. If there is no knowledge of the data the radial basis functions are chosen so that they fit points evenly distributed through the set of possible inputs [JB94]. If some knowledge as to the overall structure of the inputs exists, then it is better to try and mirror that structure in the choice of functions. It is often sufficient to position the basis functions at data points sampled randomly according to the distribution of the data. This ensures that the radial basis functions are concentrated in regions of higher data density [Low95].

Another method is to use the *orthogonal least squares* (OLS) *learning algorithm.* This method is based on *linear regression models.* The OLS method can be employed as a forward regression procedure to select a suitable set of centres (regressors) from a large set of candidates. The centres are determined one-by-one in a well defined manner until a network of adequate performance is built. At each step of the procedure, the increment to the explained vari-

ance (or energy) of the desired response is maximised. The OLS algorithm will generally produce a network with fewer hidden neurons than that of an RBF network with randomly selected centres [CCG91].

The advantage of the RBF network is that once the radial basis functions have been chosen, all that is necessary is to determine the coefficients $w_j$, to allow them to partition the pattern space correctly. An RBF network essentially preprocesses the data and transforms it into a higher dimensional space in which the classes are linearly separable [CCM96].

### 2.2.4 Self-Organised Learning

An important feature of neural networks is their ability to *learn* from their environment and through learning to *improve* in some sense. In *unsupervised learning* there is no teacher signal. A network has both inputs and outputs but there is no feedback from the environment to say what those outputs should be or whether they are correct. The network must discover for itself any patterns or features in the input data and map them onto its outputs. Unsupervised learning can only do anything useful when there is *redundancy* in the data. Without redundancy it would be impossible to find any patterns or features in the input data, which would seem like random noise [LB95, Buh95].

*Self-organised learning* is a process of unsupervised learning whereby significant patterns or features in the input data are *discovered.* The algorithm for self-organisation is provided with a set of rules of a *local* nature, which enable it to compute an input-output mapping with specific desirable properties.

"Local" means that the change applied to the synaptic weight of a neuron is confined to the immediate neighbourhood of that neuron. The question that arises is: can a useful network develop from self-organisation? Turing states [Tur52]:

*Global order can arise from local interactions.*

This development applies to both the brain and artificial neural networks (ANN). Many originally random local interactions between neighbouring neurons can coalesce into states of global order and ultimately lead to coherent behaviour. This process is the essence of self-organisation.

Network organisation takes place at two levels that interact with each other in a feedback loop. Firstly, *activity*, certain activity patterns are produced by a given network in response to input signals. Secondly, *connectivity*, the connection strengths (synaptic weights) of the network are modified in response to neuronal signals in the activity patterns, this modification is due to synaptic plasticity. The feedback between changes in synaptic weights and changes in activity patterns must be positive in order to achieve self-organisation, rather than stabilisation, of the neural network [Has95, HKP91].

Three principles of self-organisation can be identified [Hay94]:

- Modifications in synaptic weights tend to self amplify. The process of self-amplification is constrained by the requirement that modification of synaptic weights has to be based on locally available signals, that is, presynaptic and postsynaptic signals. A strong synapse leads to a coincidence of presynaptic and postsynaptic signals. In turn the neuron is

strengthened by such a coincidence.

- Limitation of resources leads to competition among synapses and therefore the selection of the most vigorously growing synapses (that is, the fittest) at the expense of the others. This process is made possible by synaptic plasticity.
- Modifications in synaptic weights tend to cooperate, in spite of the overall competition of the network.

A *Hebbian synapse* is a synapse that has its strength selectively increased, if the two neurons on either side of it are activated simultaneously (that is, synchronously). If the two neurons on either side of a Hebbian synapse are activated asynchronously, then that synapse is selectively weakened or eliminated. A Hebbian synapse is characterised by four key mechanisms:

- A time-dependent mechanism. Modifications depend on the exact time of occurrence of presynaptic and postsynaptic signals.
- A local mechanism. Locally available information is used by a Hebbian synapse to produce a local synaptic modification that is input specific.
- An interactive mechanism. Hebbian learning depends upon a "true interaction" between presynaptic and postsynaptic activities, that is, a Hebbian synapse cannot make a prediction from just one activity.
- A conjunctional or correlational mechanism. Correlation over time between presynaptic and postsynaptic activities is viewed as being responsible for a synaptic change.

If a Hebbian synapse has synaptic weight $w_{kj}$ with presynaptic signal $x_j$ and post synaptic signal $y_k$, then Hebbian learning updates the synaptic weight, at time $n$ according to:

$$\Delta w_{kj}(n) = F(y_k(n), x_j(n)) \tag{2.2.18}$$

where $F(\cdot, \cdot)$ is a function of both postsynaptic and presynaptic activities. Equation 2.2.18 may be written (as a special case):

$$\Delta w_{kj} = \eta y_k(n) x_j(n) \tag{2.2.19}$$

where $\eta$ is the learning rate (a small positive number).

Repeated application of the weight update rule, equation 2.2.19, leads to an *exponential growth* that finally drives $w_{kj}$ into saturation. A limit is needed on the growth of the synaptic weight. One method for doing this is to introduce a nonlinear *forgetting factor* into the synaptic weight adjustment. Hence:

$$\Delta w_{kj} = \eta y_k(n) x_j(n) - \alpha y_k(n) w_{kj}(n) \tag{2.2.20}$$

where $\alpha$ is a new positive constant and $w_{kj}(n)$ is the synaptic weight at time $n$. Equivalently:

$$\Delta w_{kj}(n) = \alpha y_k(n)[c x_j(n) - w_{kj}(n)] \tag{2.2.21}$$

where $c = \eta/\alpha$.

Equation 2.2.21 is known as the *generalised activity product rule.* The generalised activity product rule implies that for inputs $x_j(n) < w_{kj}(n)/c$ the modified synaptic weight $w_{kj}(n+1)$ will actually *decrease* by an amount proportional to the postsynaptic activity $y_k(n)$, if $x_j(n) > w_{kj}(n)/c$ then $w_{kj}(n+1)$ will *increase* by an amount proportional to $y_k(n)$ [Hay94, Has95].

### 2.2.5 A Simplified Neuron Model as a Principal Component Analyser

There is a close correspondence between the behaviour of self-organised neural networks and the statistical method of principal component analysis. A single linear neuron with a Hebbian-type adaptation rule for its synaptic weights can evolve into a filter for the first principal component of the input distribution [Oja82]. Consider the simple linear neuron model shown in Figure 2.7.

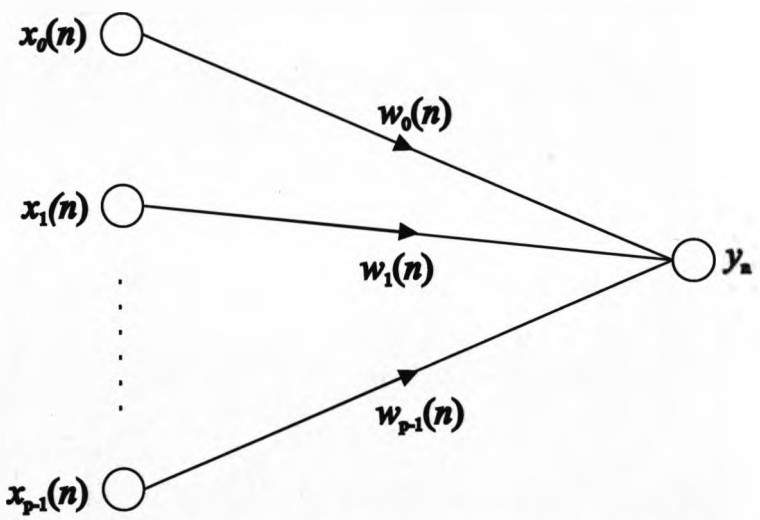


Figure 2.7: Signal flow graph of a linear neuron

The neuron receives a set of $p$ input signals $x_0, x_1, \ldots, x_{p-1}$ through a corresponding set of $p$ synapses $w_0, w_1, \ldots, w_{p-1}$, respectively. The output $y$ is

given by:

$$y = \sum_{i=0}^{p-1} w_i x_i \tag{2.2.22}$$

In accordance with Hebbian learning $w_i$ varies with time:

$$w_i(n+1) = w_i(n) + \eta y(n) x_i(n), \qquad i = 0, 1, 2, \ldots, p-1 \tag{2.2.23}$$

However, this form of learning rule leads to unlimited growth of the synaptic weight $w_i$. This can be overcome by the use of normalisation in the learning rule, for example:

$$w_i(n+1) = \frac{w_i(n) + \eta y(n) x_i(n)}{(\sum_{i=0}^{p-1} [w_i(n) + \eta y(n) x_i(n)]^2)^{\frac{1}{2}}} \tag{2.2.24}$$

where the denominator is the Euclidean vector norm. The summation in the denominator extends over the complete set of synapses associated with the neuron. Assuming a small learning rate, $\eta$, and ignoring second order and higher terms, then equation 2.2.24 can be expanded as a power series in $\eta$ as:

$$w_i(n+1) = w_i(n) + \eta y(n)[x_i(n) - y(n) w_i(n)] \tag{2.2.25}$$

The term $\eta y(n) x_i(n)$ in equation 2.2.25 provides the Hebbian modification to weight $w_i$, that is, the self-amplification effect in accordance with Principal One of self-organisation. The term $-\eta y^2(n) w_i(n)$ provides the stabilisation of the weight in accordance with Principle Two of self-organisation. Equation 2.2.25 modifies the input $x_i(n)$ into a form that is dependent on the associated synaptic weight $w_i(n)$ and the output $y(n)$. The *effective input* of the $i$th synapse is given by:

$$x_i'(n) = x_i(n) - y(n) w_i(n) \tag{2.2.26}$$

where the term $-y(n)w_i(n)$ is related to the *forgetting* or *leakage* factor and increases with stronger response $y(n)$. The learning rule becomes:

$$w_i(n+1) = w_i(n) + \eta y(n)x_i'(n) \tag{2.2.27}$$

The signal flow graph of Figure 2.8 illustrates the operation of equations 2.2.26 and 2.2.27 and exhibits two forms of internal feedback acting on the neuron [Hay94]:

- Positive feedback for self-amplification and therefore growth of the synaptic weight, $w_i(n)$, according to its external input $x_i(n)$
- Negative feedback due to $-y(n)$ for controlling the growth, thereby resulting in the stabilisation of the synaptic weight $w_i(n)$

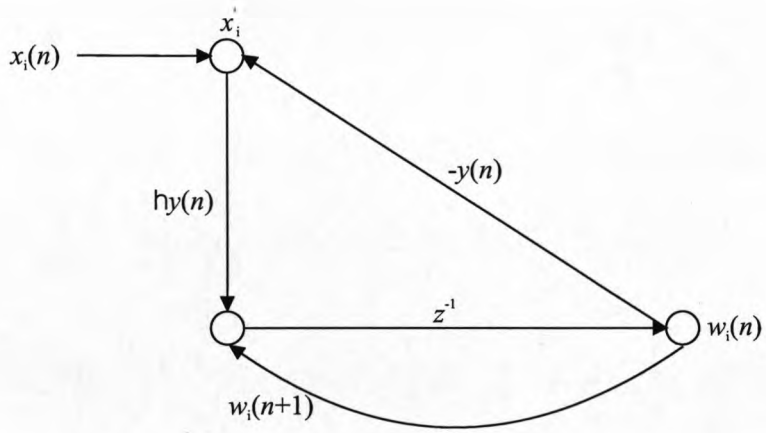


Figure 2.8: Signal flow graph of equations 2.2.26 and 2.2.27

### 2.2.6 Self-Organised Principal Component Analysis

Hebbian learning can be generalised to $m$ neurons. The number of neurons $m$ determines how many principal components the network will extract [ZL95].

Consider the neural network, consisting of a single feedforward layer of linear neurons, shown in Figure 2.9.

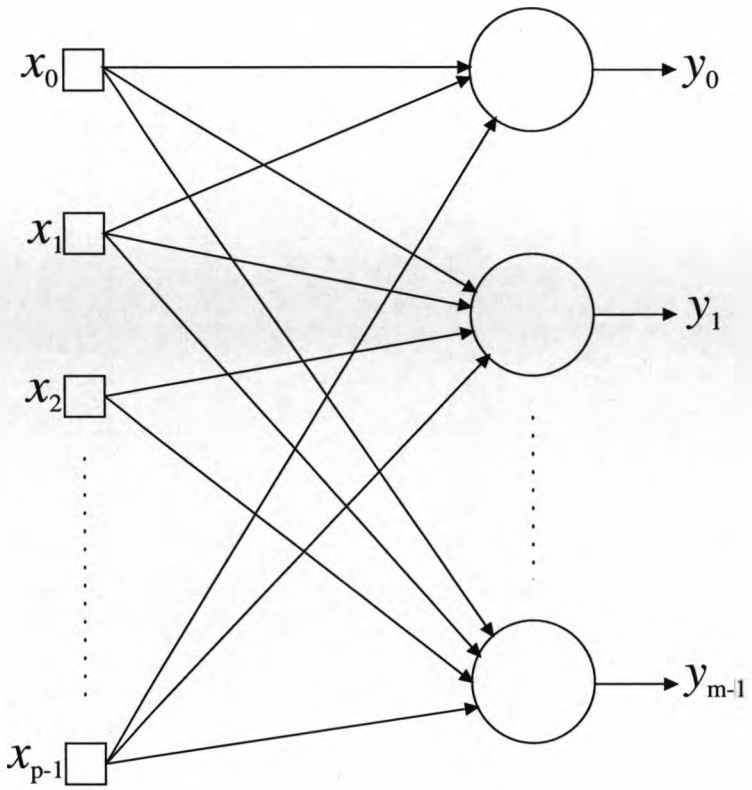


Figure 2.9: Principal component neural network

The network has $p$ inputs and $m$ outputs, with $m < p$. The output $y_j(n)$ of neuron $j$ at time $n$, produced in response to a set of inputs $\{x_i(n)|i = 0, 1, \ldots, p-1\}$, is given by:

$$y_j(n) = \sum_{i=0}^{p-1} w_{ji}(n)x_i(n), \qquad j = 0, 1, 2, \ldots, m-1 \tag{2.2.28}$$

where $w_{ji}$ is the $ith$ weight for unit $j$.

The generalised Hebbian Algorithm is given by [San89]:

$$\Delta w_{ji}(n) = \eta \left[ y_j(n)x_i(n) - y_j(n) \sum_{k=0}^{j} w_{ki}(n)y_k(n) \right], \quad \begin{matrix} i = 0, 1, 2, \ldots, p-1 \\ j = 0, 1, 2, \ldots, m-1 \end{matrix} \tag{2.2.29}$$

where $\Delta w_{ji}(n)$ is the change applied synaptic weight $w_{ji}(n)$ at time $n$. The generalised Hebbian algorithm of equation 2.2.29 for a layer of $m$ neurons includes the algorithm of equation 2.2.25 for a single neuron as a special case $j = 0$ [Oja95].

For an heuristic understanding of the generalised Hebbian algorithm and using matrix notation equation 2.2.29 can be rewritten as:

$$\Delta \mathbf{w}_j(n) = \eta y_j(n)\mathbf{x}'(n) - \eta y_j^2(n)\mathbf{w}_j(n), j = 0, 1, 2, \ldots, m-1 \tag{2.2.30}$$

where

$$\mathbf{x}'(n) = \mathbf{x}(n) - \sum_{k=0}^{j-1} \mathbf{w}_k(n)y_k(n) \tag{2.2.31}$$

The vector $\mathbf{x}'(n)$ represents a modified form of the input vector. Following equation 2.2.30 network operation is:

1. For the first neuron in the feedforward network shown in Figure 2.9:

$$j = 0: \quad \mathbf{x}'(n) = \mathbf{x}(n) \tag{2.2.32}$$

   As shown above the first neuron will discover the first principal component (that is, the largest eigenvalue and associated eigenvector) of the input vector $\mathbf{x}(n)$ [Jou97].

2. For the second neuron:

$$j = 1: \quad \mathbf{x}'(n) = \mathbf{x}(n) - \mathbf{w}_0(n)y_0(n) \tag{2.2.33}$$

Provided the first neuron has already converged to the first principal component, the second neuron sees an input vector $\mathbf{x}'(n)$ from which the first eigenvector of the correlation matrix $R$ has been removed [II97]. Therefore, the second neuron extracts the first principal component of $\mathbf{x}'$, which is equivalent to the second principal component (that is, the second largest eigenvalue and associated eigenvector) of the original input $\mathbf{x}(n)$.

3. For the third neuron:

$$j = 2: \quad \mathbf{x}'(n) = \mathbf{x}(n) - \mathbf{w}_0(n)y_0(n) - \mathbf{w}_1(n)y_1(n) \tag{2.2.34}$$

If the first two neurons have converged to the first and second principal components, the third neuron sees an input vector from which the first and second eigenvectors have been removed. The third neuron therefore extracts the first principal component from $\mathbf{x}'(n)$ which is equivalent to the third principal component of $\mathbf{x}(n)$ (that is, the third largest eigenvalue and associated eigenvector).

4. Subsequent neurons will converge to outputs of decreasing eigenvalue, that is, smaller and smaller principal components.

It should be noted however that the neuron-by-neuron description presented here is to simplify the explanation of the PCA neural network. In reality, the weight vectors, $\mathbf{w_i}$, approach their final values simultaneously, not one at a time. This simultaneous convergence of the weight vectors is of course advantageous because it leads to faster training times than if the neurons are trained one at a time [Oja82, Oja89].

The PCA network learns the principal components by unsupervised learning rules, by which the weight vectors are gradually updated until they become orthonormal and tend to the theoretically correct eigenvectors. The network also has the ability to track slowly varying statistics in the input data, maintaining its optimality when the statistical properties of the input do not stay constant. Because of their parallelism and adaptivity to input data, such learning algorithms and their implementations in neural networks are potentially useful in feature detection [Hay94, HKP91].

## 2.3 Pre-Processing the Neural Network Input Data

### 2.3.1 Statistical Averages

For random signals an important set of properties are the central moments. The two most important central moments are the first central moment, known as the mean, and the second central moment, known as the variance. Although a knowledge of these two moments does not completely describe the random data they are usually sufficient to work with [Lyn89].

The statistical average of a random variable $X$ is the numerical average of the values $X$ can assume, weighted by their probabilities. For a discrete random signal the mean is defined as:

$$\overline{x} = \lim_{N\to\infty} \sum_{i=1}^{k} x_i \frac{n_i}{N} = \sum_{i=1}^{k} x_i P_i \tag{2.3.1}$$

where $\overline{x}$ (or $\mu_x$) is the average or mean value of $X$, also known as the expected value of $X$ (the expected value $E\{X^n\}$ is referred to as the "$n$th moment" of the random variable $X$) and $P_i$ is the probability associated with the sample value $x_i$ [BP71]. The time average of $X$ is given by:

$$\overline{x} = \mu_x = \frac{1}{N} \sum_{n=1}^{N} x(n) \tag{2.3.2}$$

Similarly the second central moment, known as the variance, is defined as the expected value of the square of the functions departure from its mean. Thus variance is a measure of signal fluctuation and for a discrete random signal is

defined as [BP71]:

$$\sigma_x^2 = E\{(X - \overline{x})^2\} = \sum_{i=1}^{n} (x_i - \overline{x})^2 P_i \tag{2.3.3}$$

$$\sigma_x^2 = \frac{1}{N} \sum_{n=1}^{N} (x(n) - \overline{x})^2 \tag{2.3.4}$$

For a random signal the mean and variance cannot be determined exactly. Each time a set of $N$ samples is taken, different values are obtained. Therefore it is only possible to *estimate* the mean and the variance, based upon the $N$ samples.

If the mean is used in estimating the variance in the equation 2.3.4 the estimate is biased. An unbiased estimate is given by [BP71]:

$$\sigma_x^2 = \frac{1}{N-1} \sum_{n=1}^{N-1} (x(n) - \overline{x})^2 \tag{2.3.5}$$

The square root of the variance is called the standard deviation, $\sigma_x$, of the random variable $X$ and has the same units as $X$. The standard deviation is an indicator of the effective width of the probability density function of $X$.

Analogous to the variance of one random variable, the covariance of two random variables $X$ and $Y$ is defined as:

$$C_{xy} = E\{(X - \mu_x)(Y - \mu_y)\} \tag{2.3.6}$$

Note that $C_{xx} = \sigma_x^2$ the variance of $x(n)$ as defined in equation 2.3.4.

The covariance of $x(n)$ and $y(n)$ is related to the standard deviations of $x(n)$ and $y(n)$ by the inequality:

$$|C_{xy}| \leq \sigma_x \sigma_y \tag{2.3.7}$$

Thus the magnitude of the covariance between $x(n)$ and $y(n)$ is less than or equal to the products of the standard deviations of $x(n)$ and $y(n)$.

The normalised covariance, known as the correlation coefficient, is defined as:

$$\rho_{xy} = \frac{C_{xy}}{\sigma_x \sigma_y} \tag{2.3.8}$$

The correlation coefficient lies between $-1$ and $+1$. If the random variables have a correlation coefficient of zero, they are uncorrelated [BP71].

The covariance matrix is symmetric and hence allows the *eigenvalues* to be calculated. Eigenvalues determine the nontrivial solutions of the equation:

$$\mathbf{AX} = \lambda \mathbf{X} \tag{2.3.9}$$

which is equivalent to the homogeneous system:

$$(\mathbf{A} - \lambda \mathbf{I})\mathbf{X} = 0 \tag{2.3.10}$$

where $\mathbf{I}$ is the unit matrix.

The nontrivial solutions occur when:

$$|\mathbf{A} - \lambda \mathbf{I}| = 0 \tag{2.3.11}$$

Equation 2.3.11 is known as the *characteristic determinant.* It has $n$ roots, $\lambda_1, \lambda_2, \ldots, \lambda_n$ known as eigenvalues.

From equation 2.3.10 setting $\lambda = \lambda_i$, where $\lambda_i$ is any one of the eigenvalues, a corresponding solution vector $\mathbf{X}_i$ can be found. This vector $\mathbf{X}_i$ is called an eigenvector of $\mathbf{A}$ corresponding to $\lambda_i$.

Eigenvalues are not just the key to *differential equations*, but have a physical significance. The eigenvalues indicate the natural frequency of a system. Eigenvalues are the most important feature of practically any dynamical system [Str80].

### 2.3.2 The Fast Fourier Transform

The *fast Fourier transform* (FFT) is a powerful and widely used tool in the analysis of signals. The Fourier transform can be applied to signals to reveal information in the frequency domain that may not be readily available in the time domain. The Fourier transform is a method of expressing a given function of time in terms of a continuous set of exponential components of frequency. That is, the Fourier transform decomposes a waveform into sinusoids of different frequencies which sum to the original waveform. It identifies the different frequency sinusoids and their respective amplitudes [Bri88]. The weighting of each frequency component is given by the resulting *spectral-density function.* The *continuous Fourier transform* is defined as:

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t}dt \tag{2.3.12}$$

and the *inverse Fourier transform* by:

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)e^{j\omega t}d\omega \tag{2.3.13}$$

The *continuous Fourier transform* is not however suitable for computation by a computer because it yields an infinite number of samples of the signal,

*x(t).* The *discrete Fourier transform* (DFT) is therefore used to approximate, as closely as possible, the continuous Fourier transform. The DFT limits the number of samples, $N$, by making use of a *truncation*, or *windowing*, function. The importance of the FFT is that it offers a significant reduction in the computation time of the DFT and indeed of the inverse discrete Fourier transform (IDFT). The FFT is an algorithm that reduces the number of calculations required to compute the DFT from $N^2$ to $NlogN$. The FFT also allows for the rapid computation of the *power spectral density function* and of both the *autocorrelation* and *cross-correlation functions.*

If the signal *x(t)* is sampled every $T_0$ seconds then a sequence $\{x[k]\}$ results whose value at time, $t = kT_0$ is $x[k]$. The DFT, $X(n)$, of the finite length sequence $\{x[k]\}$ is defined as:

$$X(n) = \sum_{k=0}^{N-1} x(k) e^{\frac{-j2\pi kn}{N}}, \qquad n = 0,\ 1,\ 2,\ \ldots, N-1 \tag{2.3.14}$$

and the IDFT by:

$$x(k) = \frac{1}{N} \sum_{n=0}^{N-1} X(n) e^{\frac{j2\pi kn}{N}}, \qquad k = 0,\ 1,\ 2,\ \ldots, N-1 \tag{2.3.15}$$

### 2.3.3 Power Spectral Density

The power spectral density function describes the distribution of power versus frequency and is defined as:

$$P(\omega) = \lim_{T \to \infty} \frac{|F(\omega)|^2}{T} \tag{2.3.16}$$

The power spectral density function of a signal retains only the magnitude information, all phase information is lost. For a given signal there is a specified

power spectral density, but many signals may have the same power spectral density.

The power spectral density function of random data describes the general frequency composition of the data in terms of its mean square value. Power spectral density, $P_N$, is estimated numerically by computing the squared average of the DFT and dividing by the number of samples:

$$P_N = \frac{|X(e^{j\omega})|^2}{N}$$

### 2.3.4 Autocorrelation

The autocorrelation function is the *second joint moment* and therefore does not completely describe the random data, $x(t)$. It does however give much information about $x(t)$. The autocorrelation function of a signal is an average measure of its time domain properties and is therefore especially relevant when the signal is a random one, it identifies to what extent $x(t)$ is not random. The autocorrelation function provides the key to a random signals spectra [Str90]. The autocorrelation function is defined as:

$$R_{xx}(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)f(t+\tau)dt \qquad (2.3.17)$$

Hence the autocorrelation function is equal to the product of the signal and a time shifted version of itself, and is a function of the imposed time shift, $\tau$. It should be noted that, when the shift, $\tau$, is small the average product of the signal and the time shifted version of itself is relatively large. This is because when the shift is small, large values (positive or negative) tend to be multiplied

by similar values, hence giving a large positive product. As the shift increases there is a decreasing correlation between the signal and the time shifted version of itself, and the product therefore tends to zero. The higher the frequencies present in the signal, the smaller the shift necessary to reduce the correlation [Lyn89]. The autocorrelation function of a discrete time signal is defined as:

$$R_{xx}(k) = \frac{1}{N} \sum_{n=0}^{n=N-1-k} x(n)x(n+k) \tag{2.3.18}$$

The autocorrelation function describes the general dependence of the values of the data at one time on the values at another time, shifted by delay, k, that is, the shape of the autocorrelation function gives some idea as to how past values are related to present values of $x(t)$ and hence how fast a particular data function can change.

It should be noted that the PSD function and the autocorrelation function form a Fourier transform pair. Hence they provide similar information in the frequency and time domains respectively.

$$P(\omega) \Leftrightarrow R_{xx}(\tau)$$

### 2.3.5 Cross-Correlation

The cross-correlation function relating two functions, $f_1(t)$ and $f_2(t)$ is given by:

$$R_{xy}(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_{\frac{-T}{2}}^{\frac{T}{2}} f_1(t) f_2(t+\tau) dt \tag{2.3.19}$$

where $\tau$ is a time shift imposed upon one of the signals.

Large values of the cross-correlation function indicate the two signals are related. When the two signals being cross-correlated share a number of common frequencies, each gives a corresponding contribution to the cross-correlation function. The cross-correlation function, unlike the autocorrelation function, retains information about the relative phases of common frequency components in the two signals it is comparing. The cross-correlation function is particularly suitable for comparing random signals, since it is essentially a time averaged measure of shared signal properties. The cross-correlation function determines whether a causal relationship exists between two signals and defines it quantitatively [Lyn89]. The cross-correlation function of a random signal is defined as:

$$R_{xy}(k) = \frac{1}{N-k} \sum_{n=0}^{n=N-1-k} x(n)y(n+k), \qquad k \geq 0 \tag{2.3.20}$$

The cross-correlation function provides a measure of the similarity of one signal with another versus a relative shift by delay, k. The cross-correlation function of two sets of random data describes the general dependence of the values of one set of data on the other. A plot of the cross-correlation function, called a *cross-correlogram* will sometimes display sharp peaks, which indicate the existence of correlation between the two signals for specific time displacements [Str90].

### 2.3.6 The Correlation Coefficient

The *correlation coefficient* can be used to determine whether two random variables are interrelated and the relative strength of any interrelationship. For two random variables $x$ and $y$, the correlation coefficient is given by:

$$\rho_{xy} = \frac{C_{xy}}{\sigma_x \sigma_y} \tag{2.3.21}$$

where $\sigma$ is the *standard deviation*

$$\sigma = \left[ \int_{-\infty}^{\infty} (x-\mu)^2 f(x) dx \right]^{\frac{1}{2}}$$

and $C_{xy}$ is the *covariance* of $x$ and $y$:

$$C_{xy} = \iint_{-\infty}^{\infty} (x-\mu_x)(y-\mu_y) p(x,y) dx dy$$

where $p(x,y)$ is the *joint probability density function.*

If the two random variables $x$ and $y$ are sampled $N$ times the correlation coefficient may be estimated thus:

$$\rho_{xy} = \frac{\sum_{i=1}^{N} x_i y_i - N\bar{x}\bar{y}}{[(\sum_{i=1}^{N} x_i^2 - N\bar{x}^2)(\sum_{i=1}^{N} y_i^2 - N\bar{y}^2)]^{\frac{1}{2}}} \tag{2.3.22}$$

The correlation coefficient $\rho_{xy}$ lies between $+1$ and $-1$. The sign of $\rho_{xy}$ indicates whether the relationship between the two variables is positive or negative. The magnitude of $\rho_{xy}$, ignoring the sign, gives a measure of the strength of the relationship between the two variables. The further $\rho_{xy}$ is from zero, the stronger the relationship. The correlation coefficient takes the value $+1$ only if the plotted bivariate data displays a perfect linear relationship with

a positive slope, and $-1$ if the plotted data displays a perfect linear relationship with a negative slope. A nonlinear relationship and/or data scatter, whether it be due to data measurement errors or imperfect correlation of the variables, will force the value of the correlation coefficient $\rho_{xy}$ towards zero, that is, data where the two variables are unrelated will have a correlation coefficient of 0. It should be noted that higher values of $\rho_{xy}$ do not imply anything at all about the *slope* of the straight-line fit: they say something about the *quality* of the fit.

Care, however, needs to be exercised when using correlation coefficients. Calculating the correlation coefficient of a data set reduces it to a single number and clearly a lot of information can get lost in the process. Therefore, it is hardly ever adequate to simply look at the value of the correlation coefficient when examining the relationship between two variables.

A study of a scatter plot of the data is usually useful in determining whether or not the correlation coefficient will reveal any useful information. The correlation coefficient is most useful when the data form a more or less oval pattern on the scatter plot. It is less suitable when the data shows a curvilinear relationship. If the data contains outliers it may be more appropriate to calculate the correlation coefficient once they have been removed from the data set (whether it is appropriate to remove outliers or not is, of course, another question whose answer depends upon the nature of the outlier).

## 2.4 Information Extraction of the Oil Mist Data

### 2.4.1 The Oil Mist Data

The purpose of this work was to create a novel alarm system for use in the monitoring and prediction of random time series signals. From a knowledge of a random signals history, present value, and its statistical properties it should be possible to forecast the future of a signal. *Shell U.K.* wish to develop such a system for unmanned remote oil installations, with the aim of producing an alarm system that can anticipate problems from the presence of oil mist, and have provided the raw data for the development of this work. The scatter data is provided by three infra-red detectors used to measure oil mist. The three detectors monitor the same cell from different angles with a sampling frequency of 5Hz, and the data falls into one of four alarm categories: high, low, grey and none. Figure 2.10 shows a typical sample of the oil mist scatter data, the data is windowed into 20 second segments containing 101 samples for each of the three detector signals. It should be noted that differing quantities of data were available for each alarm condition, ranging from 380 seconds worth of data for the low alarm condition to 1140 seconds worth of data for the high alarm condition.

The aim of preprocessing the data is to extract key information in order to reduce the dimensionality of the problem and hence simplify the neural network required. However, care has to be taken in the use of transforms to preprocess the data, as problems can be caused. Firstly, although they can

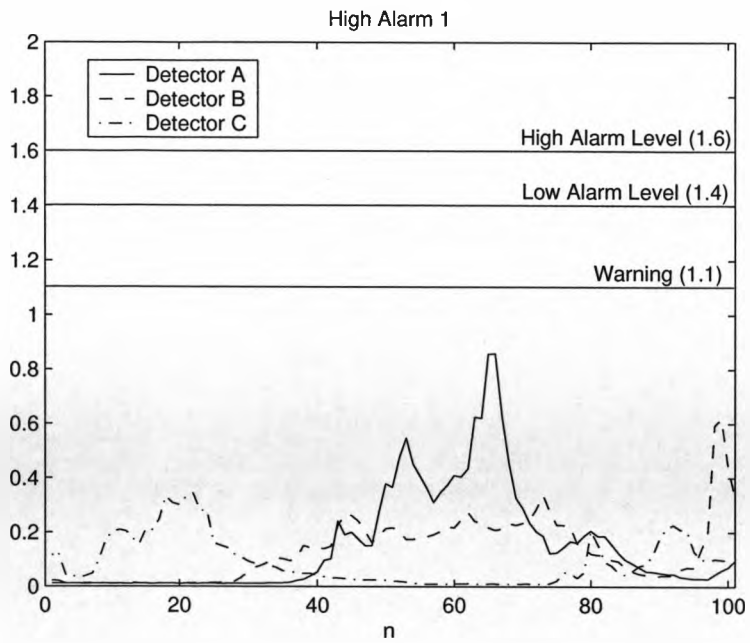


Figure 2.10: A typical 20 second sample of the Shell oil mist scatter data

represent signals to an arbitrary degree of accuracy, there is no guarantee that they make it easy to separate classes. It could be that in the transform space the representation of two different classes is such that a complicated decision boundary is required to separate them. Secondly, many transforms perform averaging of the input data. If an important feature is represented by a small characteristic of the input vector, its contribution to the average may be so low that it is discarded.

The first attempt to train a neural network used the raw data to train a radial basis function network. This work was undertaken using the mathematical computing package *Matlab* and for each data processing task required a Matlab function was written.

The Matlab function to train a radial basis network, *solverb* [nn:92] only

allows the user to alter two input design parameter the *spread constant,* **sc** and the error goal. The spread constant determines the width of the radial basis function shown in Figure 2.5. Each bias in the radial basis hidden layer is set to 0.8326/sc [nn:92], this determines the width of an area in the input space to which each neuron responds to. The spread constant should be large enough to allow the radial basis functions of the hidden layer to overlap so as to allow good generalisation. However the radial basis functions should not be spread out such that the radial basis neurons return outputs near 1 for all the input vectors used in the design. Ideally the spread constant should be much larger than the distance between adjacent input vectors, so as to get good generalisation, but much smaller than the distance across the whole input space. For the example shown in Figure 2.11, a spread constant greater than 0.1, the interval between inputs, and less than 2, the distance between the left most and right most inputs should be chosen [nn:92]. The second design parameter the user can choose is the error goal. This is the sum-squared error that the network must reach before training can stop.

### 2.4.2 Training With The Raw Data

The data was provided in a series of text files, high, low, grey and none, a Matlab function *raw.m* was written to convert each text file, containing three variables (one from each infra-red sensor), into a one column vector *mat* file, that is, a Matlab variable. A Matlab function *nnraw.m* was written to create a neural network input vector, *praw.* An output target vector, $t$ was also created.

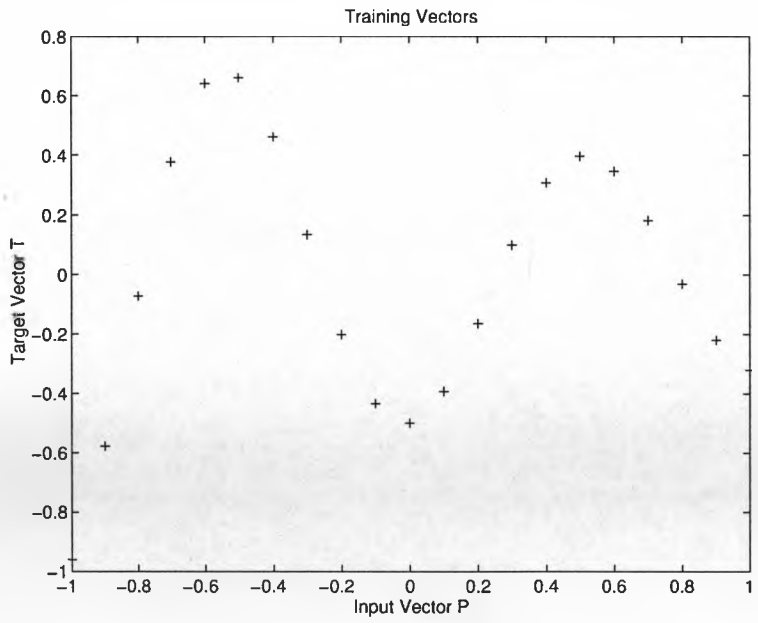


Figure 2.11: Choosing a spread constant

For the sake of simplicity the following targets were adopted:

| | |
|---|---|
| High Alarm | 4 |
| Low Alarm | 3 |
| Grey Alarm | 2 |
| None Alarm | 1 |

Table 2.1: Neural network target vector for the High, Low, Grey and None alarm conditions

An input vector of ten of each of the alarm conditions was created, that is, a 303 x 40 matrix, as was a 1 x 40 target vector. It should be noted here that a limiting factor on the size of the input vector is the fact that only 19 examples of the low alarm condition are available, their are however more available for the other alarm conditions, and in order to check the generalisation of the network after training it is necessary to reserve some of the input vectors for testing.

For the raw data provided a spread constant larger than 0.2, the time between samples, and smaller than 60, the time between the first and last samples, is required. An initial spread constant of 3.5 was chosen. This figure was derived because 3.5 is approximately 17 times larger than 0.2 and approximately 17 times smaller than 60 and is therefore midway between the two extremes. An error goal of 0.001 was chosen.

Once the network has been trained, its weights and biases calculated, the Matlab function *simurb* [nn:92] can be used to test the network designed. The neural network trained correctly using the raw data, but failed to generalise when previously unseen raw input data vectors were presented to the network. Further training then took place with a range of different spread constants. In practice it appears that the size of the spread constant makes little difference to the design of the network. Spread constants well outside this range resulted in essentially the same network being designed.

The probable source of this problem is too many radial basis neurons in the hidden layer. The function *solverb.m* iteratively creates a radial basis network one neuron at a time, adding neurons until the sum-squared error falls below the error goal. The neural network created had thirty neurons in the hidden layer, n.b. their were forty input vectors. With this large a ratio of hidden neurons to training vectors the neural network was essentially providing a one-to-one mapping between the input and target vectors. Thus creating a network with poor generalisation.

The choice of error goal also has little effect on the design of the network.

This is because, in this case, once the sum-squared error dropped below zero it fell off extremely rapidly to an error of the order of $10^{-16}$ as can be seen in Figure 2.12.

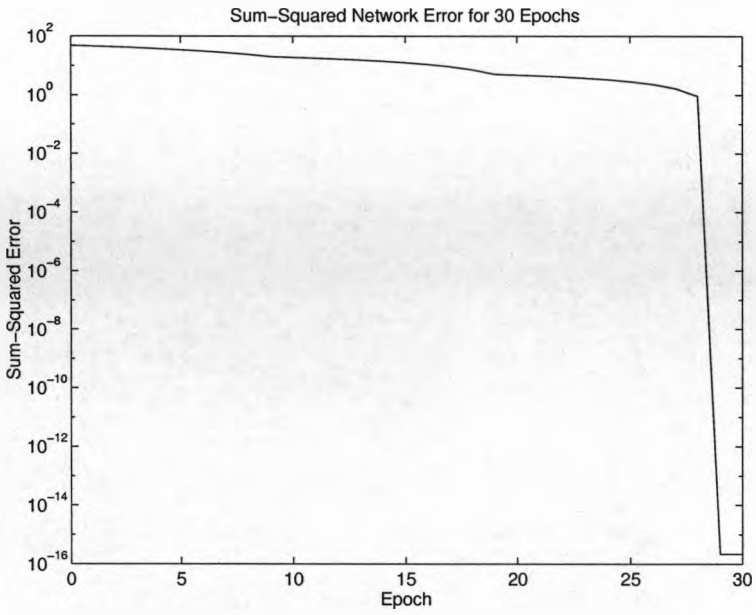


Figure 2.12: The sum-squared error for a radial basis network trained using the raw data.

#### 2.4.2.1 Pre-Processing the Data with the Cross-Correlation Function

*Shell U.K.* believe that the phase information between the three detectors scatter beams is of significance. The cross-correlation function, which retains phase information, was therefore initially chosen for the preprocessing of the input data prior to its being used to train a neural network. As noted above, differing quantities of data were available for each alarm condition and the lack of data, for the low alarm condition in particular, limited the amount of data

available for training purposes as it was necessary to reserve some data for neural network generalisation purposes once training was complete.

The function *shelxcor.m* calculates for each raw data file the unbiased cross-correlation of each of the three possible combinations of input signals. Figures 2.13 to 2.20 show plots of the cross-correlation functions of a selection of randomly chosen samples of the oil mist scatter data. From an examination of Figures 2.13 to 2.20 it is not immediately obvious, in the intuitive sense, that the patterns are separable by a neural network. The scale of the cross-correlation functions differs markedly between different data sets of the same alarm conditions, for example, Grey24 (Figure 2.15) and Grey33 (Figure 2.19). In addition, with the exception of low alarm conditions, the scale of the cross-correlation functions for different alarm conditions tend to overlap. Although these data samples were chosen randomly, comparison with other data samples suggests that they are not untypical.

An unbiased cross-correlation of the oil mist data was performed for each window of data available. The results of this cross-correlation of each segment of data was then concatenated to provide a single input vector for neural network training or later network generalisation. *Shelxcor* uses the Matlab function *xcorr* [sp:92], if the input vectors are of length $M$ then *xcorr* returns the cross-correlation sequence in a vector of length $2M-1$. Each raw data input vector contains 101 samples, therefore concatenating the three cross-correlation functions creates a neural network input vector containing 603 samples. For a neural network this represents a 603 dimensional space. This high dimension-

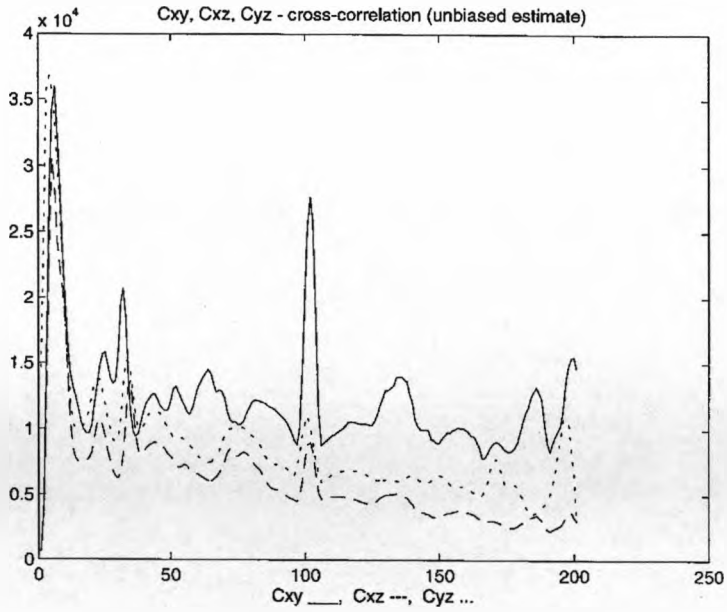


Figure 2.13: Cross-correlation coefficients for High 42 data sample

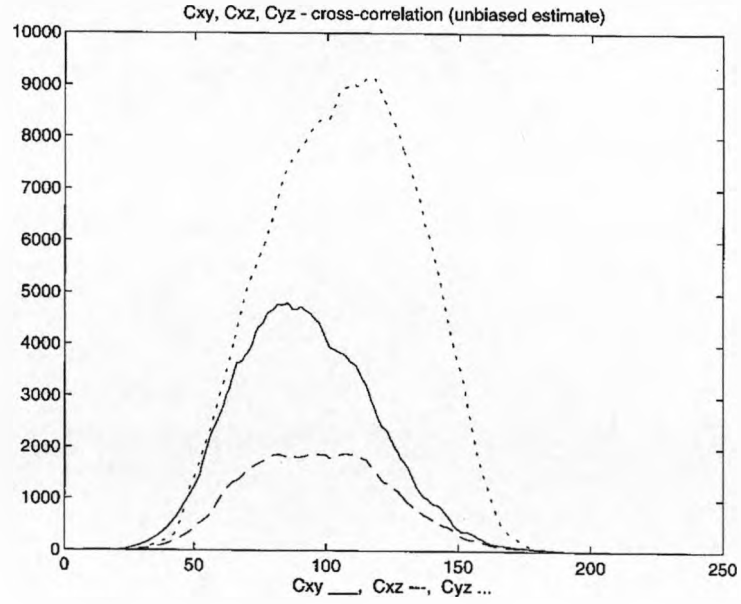


Figure 2.14: Cross-correlation coefficients for Low 15 data sample

ality makes it more difficult to design a neural network to produce the desired input-output mapping. This has resulted in an approximate doubling of the

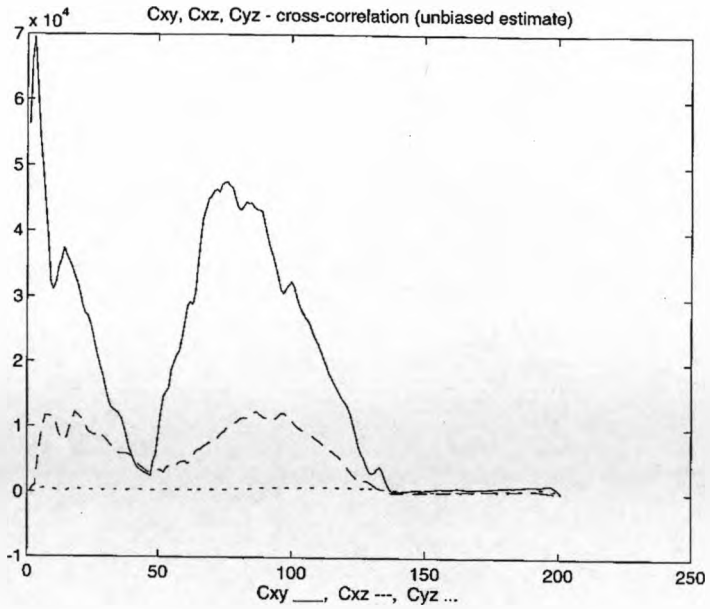


Figure 2.15: Cross-correlation coefficients for Grey 24 data sample

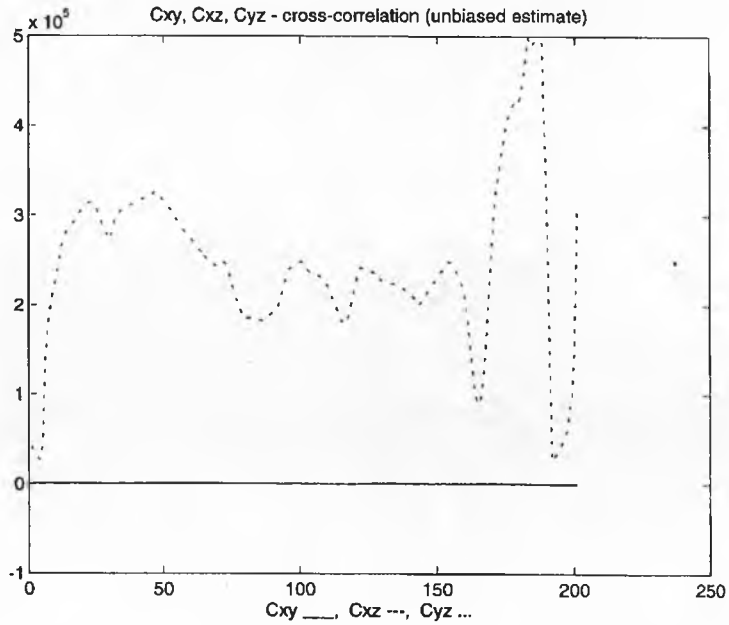


Figure 2.16: Cross-correlation coefficients for None 18 data sample

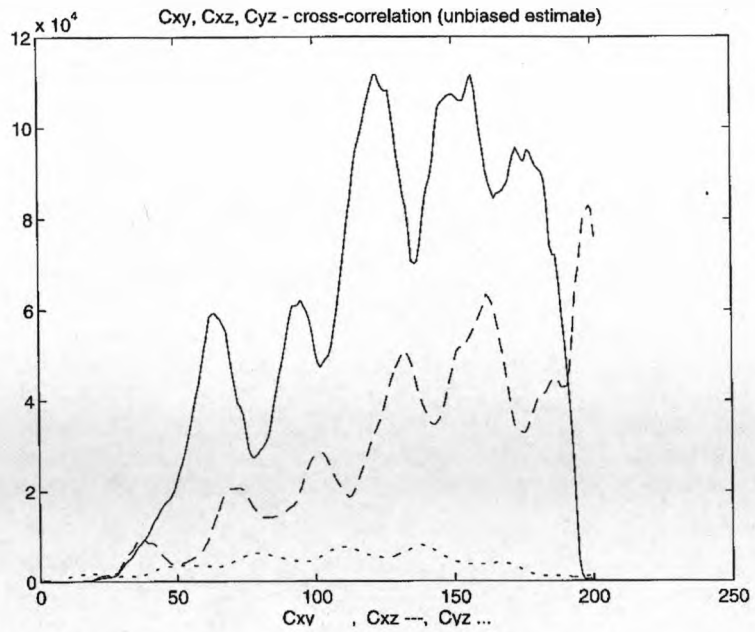


Figure 2.17: Cross-correlation coefficients for High 50 data sample

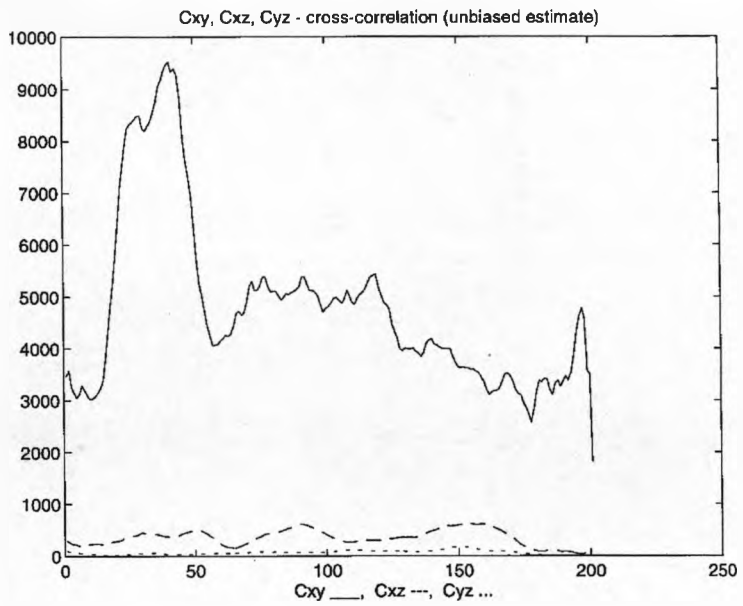


Figure 2.18: Cross-correlation coefficients for Low 6 data sample

size of the neural network input data.

The aim of pre-processing the data prior to neural network training, is to

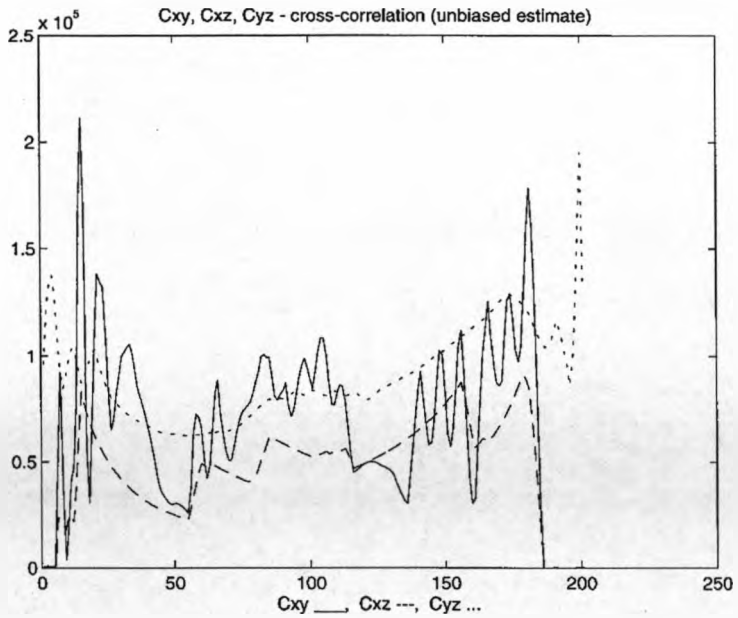


Figure 2.19: Cross-correlation coefficients for Grey 33 data sample

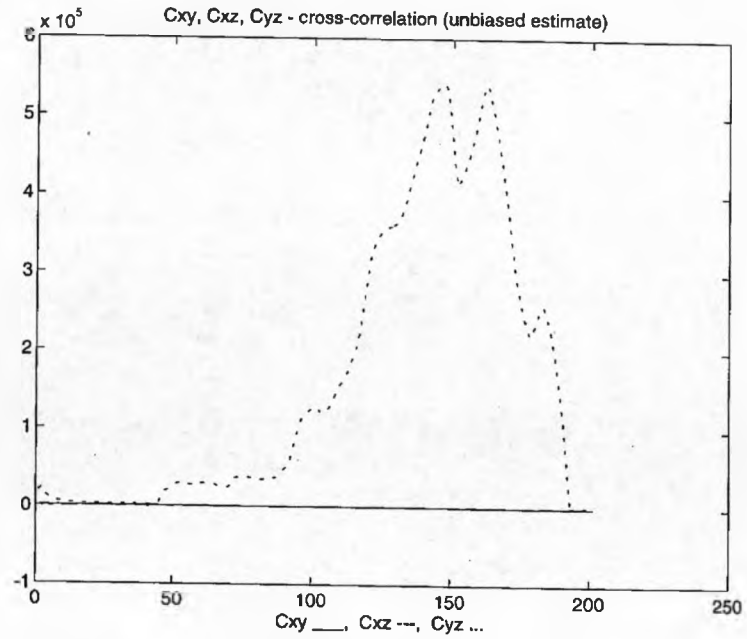


Figure 2.20: Cross-correlation coefficients for None 7 data sample

extract the key features from the raw input data without loss of information and thus simplify the neural network required. The cross-correlation function in approximately doubling the dimensionality of the problem has thus failed in that respect. Therefore, because the cross-correlation function has doubled the dimensionality this has not simplified the neural network requirements. However, this has provided a key stage in the development of this problem.

A $603 \times 40$ input matrix, consisting of 10 low, 10 high, 10 none and 10 grey alarm signals, was created for the purpose of training an RBF neural network. Matlab was used to train a radial basis function neural network using the input vector described above and neural network target outputs as shown above in Table 2.1. These values were chosen for convenience and ten such vectors were concatenated to form $40 \times 1$ target vector. A sum squared error goal for the neural network of 0.0001 was set and a *spread constant*, SC, of 4.9 chosen. The spread constant must be in the range:

$$0.2 < SC < 120.6$$

where $0.2s$ is the distance between adjacent inputs and $120.6s$ is the distance across the whole input vector.

The radial basis function neural network trained successfully, reaching an error goal of $10^{-15}$ after 89 epochs, as shown in Figure 2.21. However, although the network trained successfully it did not generalise when presented with previously unseen test vectors. On testing the generalisation of the neural network, all the test vectors returned a value of 1, that is, the none-alarm

condition.

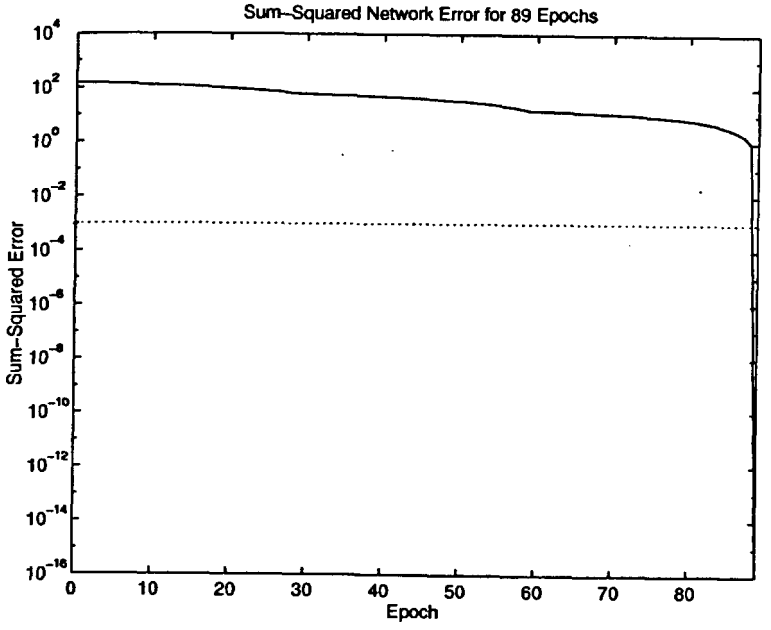


Figure 2.21: A radial basis function neural networks sum-squared network error for 89 epochs

Because of the relatively high ratio of hidden neurons to training patterns, 29:40, it appears that a radial basis function neural network is being created which has a virtually one-to-one mapping between the input training vectors and the target vectors. Thus creating a neural network that has poor generalisation. It should be noted that, Matlab creates radial basis function neural network's iteratively. One neuron is added at a time until the sum squared error falls beneath an error goal or a maximum number of neurons, specifiable by the user, has been reached.

Changing the spread constant made no difference to the performance of the neural network. For a wide range of spread constants a neural network essentially identical to the network described above was created, that is, one

that trained correctly but completely failed to generalise when presented with previously unseen training vectors.

To reduce the length of the networks input vectors and hence the dimensionality of the problem the function *xcorwin* was written. This function allows the user to window the raw data, to a length, $N$, of their choosing, before calculating the cross-correlation functions of the data. This windowing has the added advantage of creating more vectors for use in the training set. For example, windowing the data to a length $N = 50$, halves the dimensionality of the problem and doubles the number of input vectors available for training.

However this approach yielded the same results as above, the neural network trained successfully, reaching an error goal of $10^{-15}$ after 29 epochs, as shown in Figure 2.22. Again, however, although the network trained successfully it completely failed to generalise when presented with previously unseen test vectors. With an input of 120 training vectors the trained network required 90 hidden neurons, thus again essentially providing a one-to-one mapping between the input and output. The network therefore failed to generalise correctly when presented with previously unseen training vectors.

Further reducing the size of the cross-correlation window, thus further reducing the size of the input vectors and further increasing the number of training vectors available, produced similar results.

The training vector described above was then used to train a range of multilayer perceptron neural networks with differing numbers of hidden and output neurons. All these networks did not train correctly, as the neural networks

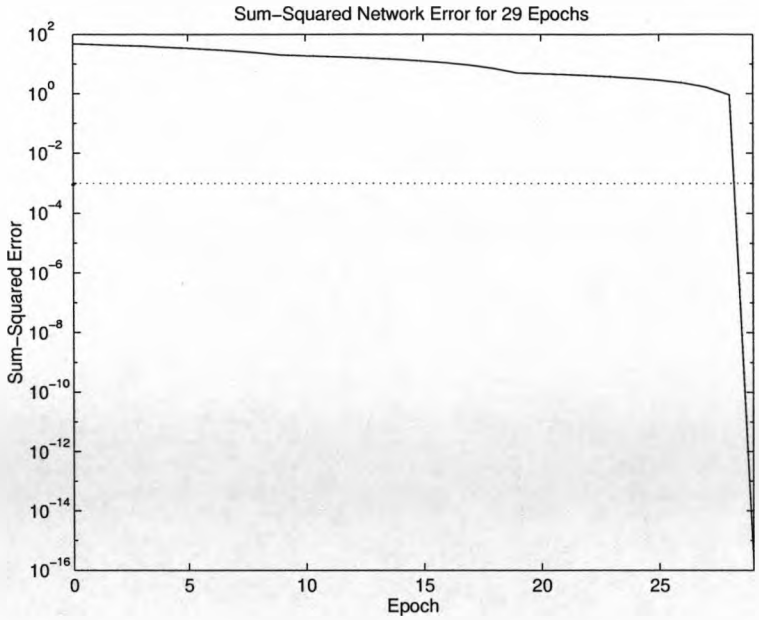


Figure 2.22: A radial basis function neural networks sum-squared network error for 29 epochs

failed to separate the training patterns in the pattern space. Plots of typical training errors are shown in Figures 2.23 and 2.24, which show the training errors for back-propagation neural networks with 10 hidden and 4 output neurons and 50 hidden and 4 output neurons respectively. As can be seen from Figures 2.23 and 2.24 both neural networks completely failed to reach the error goal during training.

#### 2.4.2.2 Pre-Processing the Data with the Autocorrelation Function

The Matlab function *shelacor* was written to calculate the autocorrelation of the raw data using *acorr* to calculate an unbiased estimate. Figures 2.25 to 2.30 show typical plots of the autocorrelation function. When examining these plots it is not possible, in an intuitive sense, to determine whether or

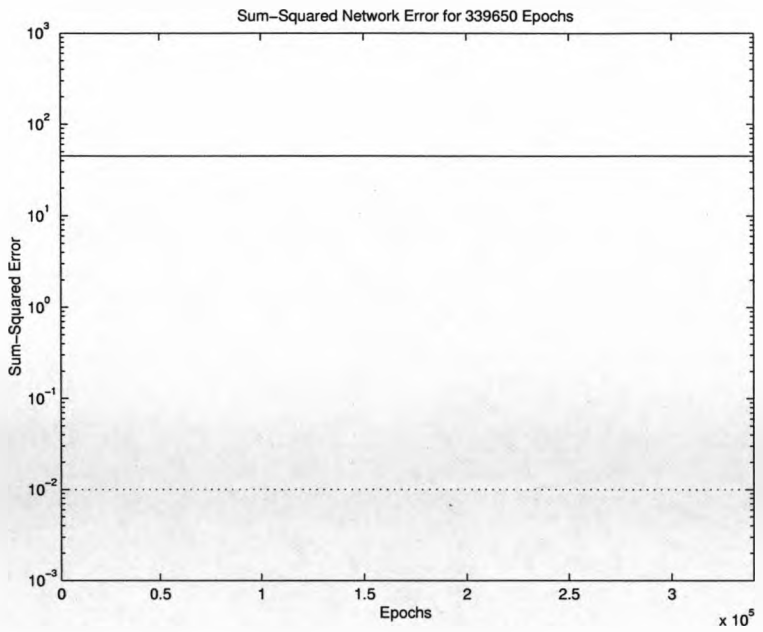


Figure 2.23: Plot of the training errors for a back-propagation network with 20 hidden and 4 output neurons

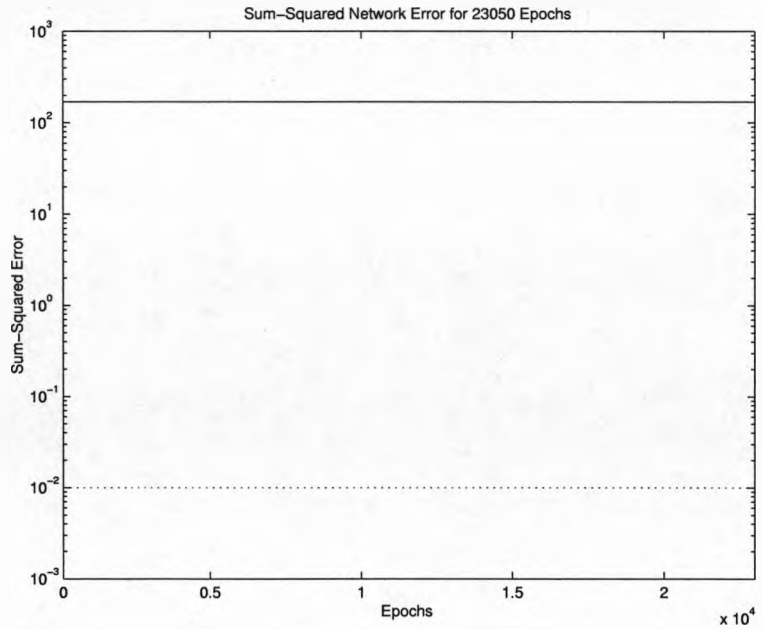


Figure 2.24: Plot of the training errors for a back-propagation network with 10 hidden and 4 output neurons

not the data sets are separable by a neural network. The scale of the autocorrelation function differs markedly between different detectors of the same data window, as can be seen from High 42 Detector A, Figure 2.25, and High 42 Detector C, Figure 2.27. In addition, the scale of the autocorrelation functions of the different data sets often overlap, for example, High 42 Detector A, Figure 2.25, and Low 15 Detector B, Figure 2.29.

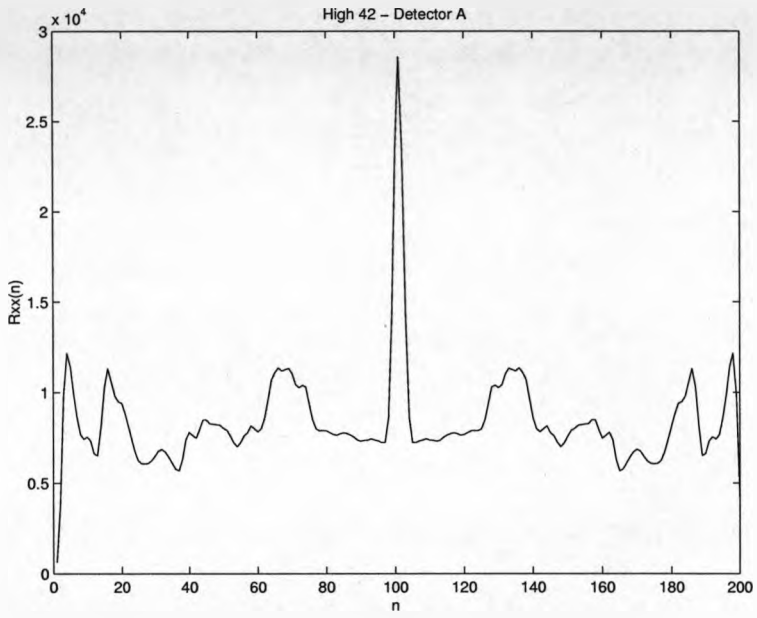


Figure 2.25: Plot of the autocorrelation (unbiased estimate) of High 42 - detector A

As with the cross-correlation function, above, calculating the autocorrelation coefficients of the data resulted in a doubling of the size of the neural network input vector. A $603 \times 40$ input matrix, consisting of 10 low, 10 high, 10 none and 10 grey alarm signals, was created for the purpose of training a multilayer perceptron neural network. Matlab was used to train a radial basis function neural network using the input vector described above and neural

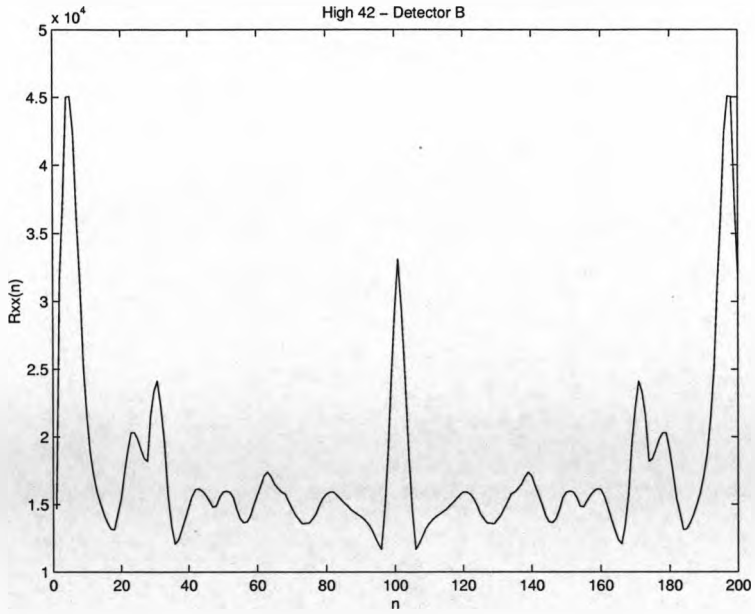


Figure 2.26: Plot of the autocorrelation (unbiased estimate) of High 42 - detector B

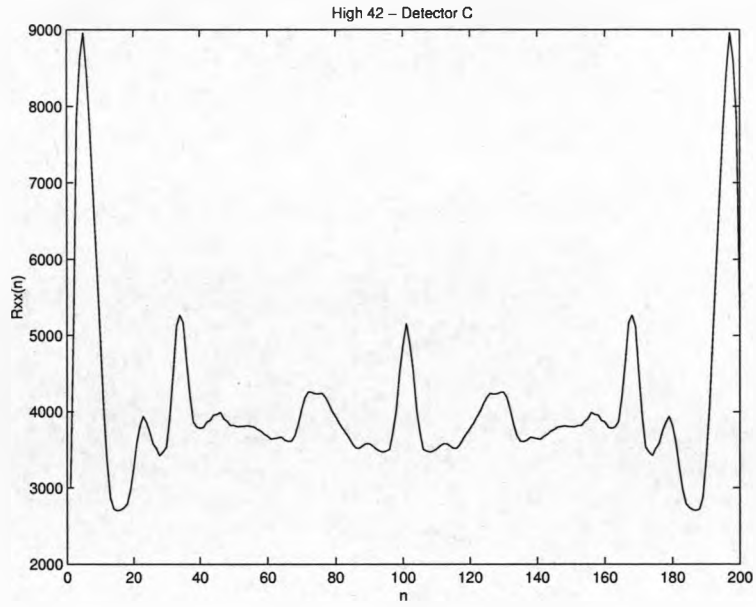


Figure 2.27: Plot of the autocorrelation (unbiased estimate) of High 42 - detector C

network target outputs as shown above in Table 2.1. These values were chosen for convenience and ten such vectors were concatenated to form $40 \times 1$ target

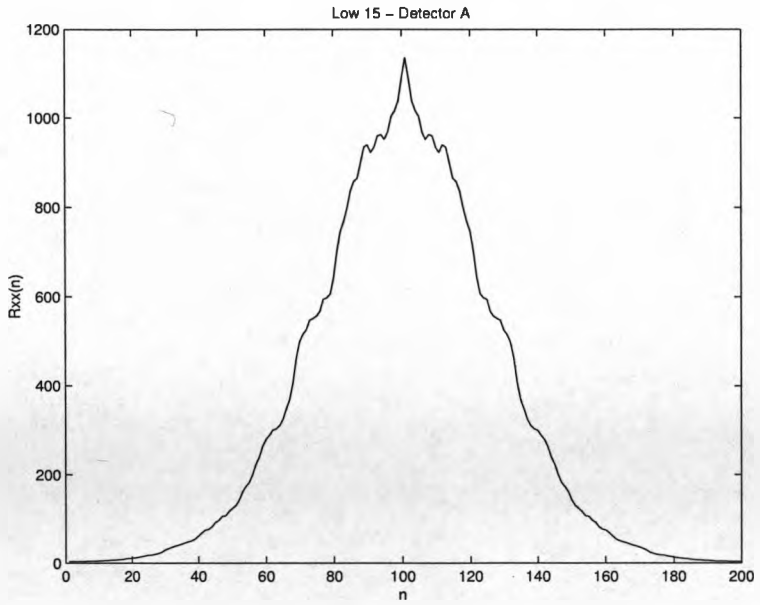


Figure 2.28: Plot of the autocorrelation (unbiased estimate) of Low 15 - detector A

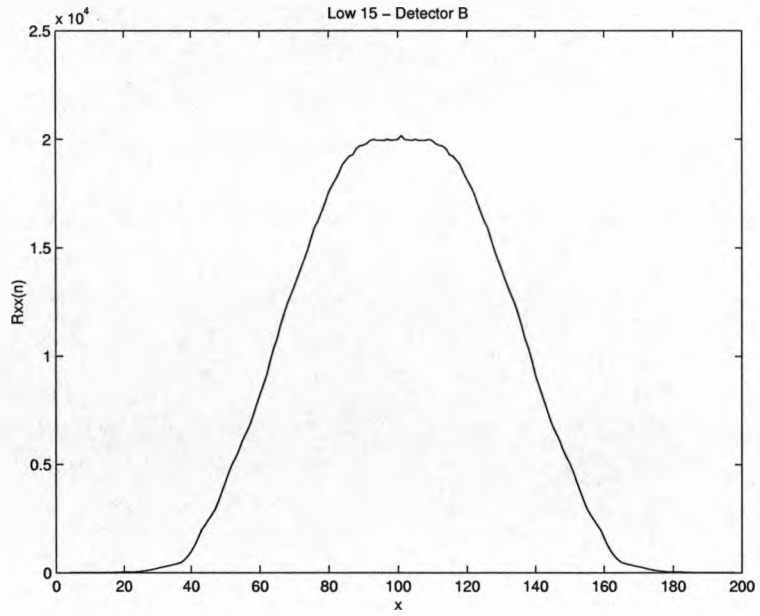


Figure 2.29: Plot of the autocorrelation (unbiased estimate) of Low 15 - detector B

vector. A sum squared error goal of 0.001 was set for the neural network. Figure 2.31 shows a typical plot of the networks training errors.

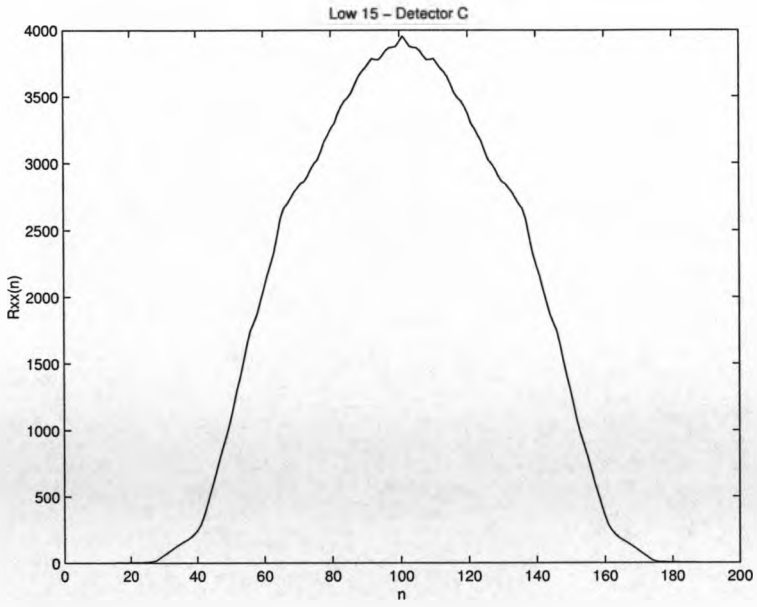


Figure 2.30: Plot of the autocorrelation (unbiased estimate) of Low 15 - detector C

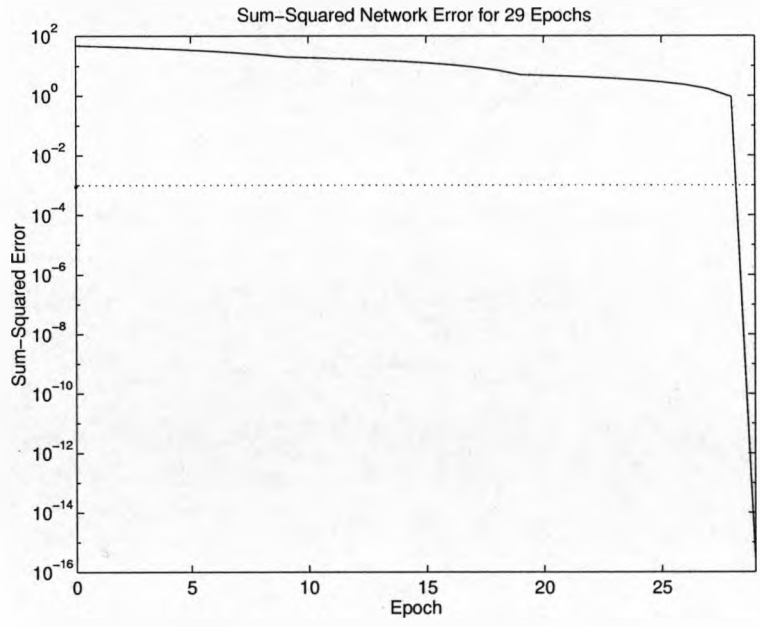


Figure 2.31: A radial basis function neural networks sum-squared network error for 29 epochs

Use of the autocorrelation function to train the neural network also created a network that failed to generalise correctly when presented with previously

unseen test vectors. The network, with 30 hidden neurons, again appeared to provide an essentially one-to-one mapping between the training input and target vectors. For a wide range of spread constants the neural network training failed to produce a network that was capable of successful generalisation. Similarly, windowing the data to reduce the dimensionality of the network required did not produce any observable change in network generalisation.

#### 2.4.2.3 Pre-Processing the Data with the Power Spectral Density Function

*Shelpsd* uses the Matlab function *psd* [sp:92] to calculate the power spectral density of the raw data. *Psd* allows the user to specify: *nfft*, the FFT length and *Fs*, the sampling frequency. The length of each raw data variable is $N = 101$ and therefore *nfft* was chosen to be 128, to allow for fastest execution. The data was sampled at 5Hz making $Fs = 5$Hz. Although it was possible to train a radial basis function network using the power spectral density functions to reach the required error goal, Figure 2.32, the network again failed to generalise. As in the cases of the cross-correlation and autocorrelation functions the network largely failed to classify test vectors correctly.

### 2.4.3 Training with the Signal Statistics

Further work involved using some of the statistics of the input signals to train a multilayer perceptron network. The statistics chosen were the mean, the variance and the eigenvalues. The function *shelstat* calculates the mean,

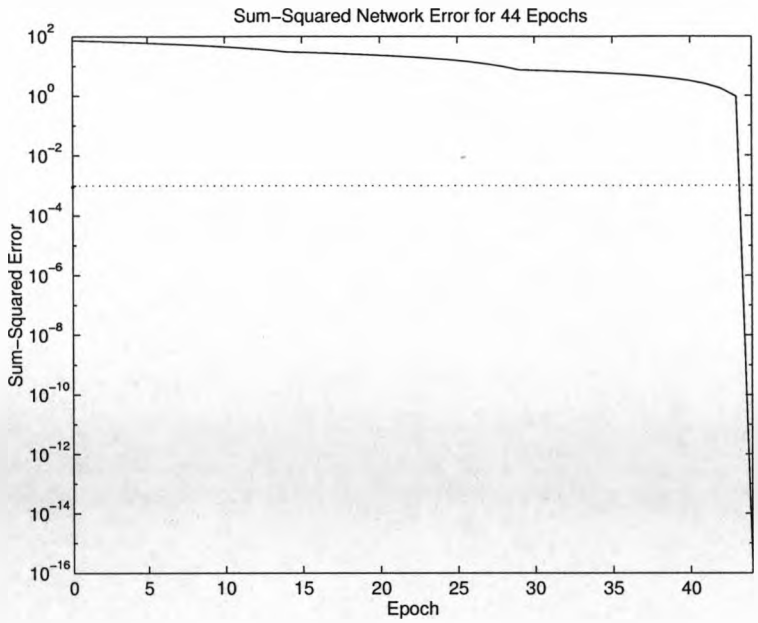


Figure 2.32: A radial basis function neural network's sum-squared network error for 44 epochs

variance and eigenvalues for each of the three signals in each set of data. The variance and the eigenvalues are calculated from the covariance matrix of the three signals using *diag(cov(x))* and *eig(cov(x)* [sp:92] respectively, where $x$ is the input signal. For each set of data the results are then concatenated to form a neural network input vector. The length of these input vectors is $N = 9$, three mean values, three variances and three eigenvalues. The function *nnstat* batches the training vectors as described previously.

The function *bpnn.m* uses the Matlab functions: *rands* or *initff* [nn:92] to initialise the weights and biases of the back propagation network, and *trainbpx* [nn:92] to train the network. Alternatively *trainbpa* [nn:92] trains the network with adaptive learning and *trainbpm* [nn:92] trains the network with momentum.

Function *bpnn* creates an network with a layer of hidden neurons (the number of hidden neurons is variable) with hyperbolic tangent sigmoid transfer functions, as shown in Figure 2.4, and one output neuron with a linear transfer function. An alternative function *bp* creates a network with two hidden layers containing nonlinear neurons and an output layer consisting of one linear neuron. These functions can be amended to create neural networks with more than one neuron in the output layer.

A feature of the multilayer perceptron is that they can often require long training periods. This is due to the repetition of the back propagation of errors algorithm required to train the network. Depending upon the initialisation of the weights and biases, some of the multilayer perceptron networks used have taken up to 500,000 epochs to train, requiring several days (a problem exacerbated by the memory limitations of the computers available). Figure 2.33 shows a typical multilayer perceptron training record.

The signal statistics extracted from the data were used to train a number of multilayer perceptron networks, with both one and two layers of hidden neurons and a wide range of neurons in the hidden and output layers. Some of these neural networks failed to train within the given maximum number of epochs. Other networks did train correctly but, when presented with previously unseen test vectors, they generally did not classify the test patterns correctly.

This work has examined pre-processing the raw data using; the statistical averages, mean and variance; the fast Fourier transform; the power spectral density function; the autocorrelation function and the cross-correlation func-

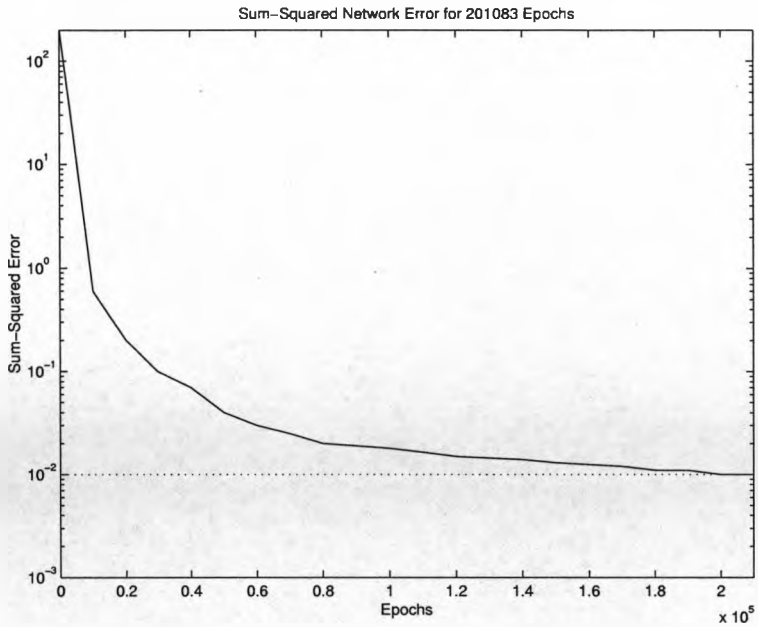


Figure 2.33: A multilayer perceptron neural network's sum-squared network error for 201083 epochs

tion, prior to training and generalisation with neural networks. Although some successes were recorded, more work is required to develop the techniques with this data to be able to implement a reliable real-time classification system.

## 2.5 Conclusion

The research work described in this section is to develop a novel methodology for stochastic time series classification. The aims were to produce an intelligent monitoring and prediction system based on neural networks. *Shell U.K.* wish to develop such a system for unmanned remote oil installations, with the aim of producing an alarm system that can anticipate problems due to oil mist, and have provided the raw data for the development of this work. The oil mist data provided, is scatter data that has been detected by three infra-red detectors all monitoring the same cell from different angles. The data falls into one of four alarm categories: High, Low, Grey and None.

Initially a radial basis function neural network was trained using the raw data. This produced a neural network that could correctly classify training vectors however, this network was unable to generalise and classify test vectors. These findings led to the conclusion that it would be necessary to preprocess the raw data. To simplify a neural network, and hence decrease the time the network takes both to train and to generalise, preprocessing of the input data is often performed. The aim of the preprocessing is to extract the key features of the data without at the same time losing any information that may be important.

A number of methods for preprocessing the data have been examined: the fast Fourier transform, the power spectral density function, the autocorrelation function and the cross-correlation function. All of these techniques have been

used to preprocess the data for training radial basis function neural networks. As with the raw data it is possible to successfully train a radial basis function neural network to reach a desired sum squared error goal. However, none of the networks have been able to generalise correctly when presented with test vectors. Therefore, preprocessing the data has not achieved a sufficient reduction in the dimensionality of the problem to allow the neural network to classify the data.

The aim of preprocessing the data is to extract the key features from the data and hence simplify the network required. While the preprocessing techniques described are all important measures of a signal, their use does not necessarily simplify the task of separating the different patterns in the transformed pattern space. Indeed both the autocorrelation function and the cross-correlation function double the length of each training vector, and therefore double the dimensionality of the pattern separating problem. The power spectral density function reduces the dimensionality of the pattern space by approximately 30%, but still leaves a problem of high dimensionality.

To overcome the problem of large training vectors, the use of some important signal statistics was examined. These statistics are the mean, the variance and the eigenvalues. The use of these statistics greatly reduces the dimensionality of the problem, without of course guaranteeing that the patterns have become separable in the pattern space. The data, having been preprocessed using the signal statistics, was then used to train multilayer perceptron neural networks.

A range of backpropagation networks were used, with varying numbers of neurons in the hidden and output layers. However, despite long training times, none of these networks reached their error goal and therefore did not train correctly. Several networks were unable to correctly classify the training vectors once training had been completed. Windowing the data, to further reduce the dimensionality of the neural networks required, did not produce any observable change in the performance of the neural networks trained.

These investigations have found that although the networks could generally be trained correctly, the difficulties lie with interpretation of new data. Despite using several different methodologies, so far it has not been possible to produce an intelligent monitoring and prediction system sufficiently reliable for a real world time series prediction problem.

# Chapter 3

# Principal Component Analysis and it's Applications

## 3.1 Introduction

*Principal Component Analysis* (PCA) is a standard technique used in feature extraction and data compression and has been used in pattern recognition. PCA belongs to a field known as *multivariate analysis.* PCA concentrates on the internal structure of the data and is defined by the *eigenvectors* of the *covariance matrix* of the input data. The goal is to find some pattern, or some natural structure, within the data. PCA takes advantage of any redundancy in the data set and enables a group of variables to be replaced by a single variable. Thus PCA linearly reduces the dimensionality of a set of measurements while retaining as much information as possible about the original data.

A data set contains a number of variables, each of which has its own variance. Each variable is an axis or dimension of variability. Usually the variables are associated with each other, that is, there is a covariance between pairs of variables. The data set as a whole will have a variance which is the sum of the individual variances.

PCA transforms the data so that the transformed data has:

- the same amount of variability
- the same total variance
- the same number of axes (or variables) as the original data.

Additionally the PCA transform is performed in such a way that:

- the first axis accounts for as much of the total variance as possible
- the second axis accounts for as much of the remaining variance as possible, while being uncorrelated with the first
- the third axis accounts for as much of the total variance remaining after being accounted for by the first two axes, whilst being uncorrelated with either, and so on [Str80].

After the PCA transform the new axes, or dimensions, are uncorrelated with each other and are weighted according to the total amount of variance they describe. Normally this results in there being a few large axes that account for most of the total variance and a larger number of small axes accounting for very small amounts of the total variance. These small axes are usually discounted from further consideration [Wol78].

Thus the data has been transformed from a set of $p$ correlated variables to a set of $m$ uncorrelated axes, or *principal components.* Where $m < p$ and the redundancy induced by the correlations is removed. The existence of $m$ uncorrelated axes is often a useful property if further analysis is planned because, $m < p$ introduces a parsimony in the representation of the data which is often desirable [JB94].

Much attention is focused on the relationship of the principal components to the original variables. How can each principal component be interpreted in terms of the original variables? What are the values of the data points after the transformation?

## 3.2 Principal Component Analysis

Assume that a random variable vector, $\mathbf{x}$, of dimension $p$ has a zero mean value and unit standard variance, that is, $E[\mathbf{x}] = 0$ and $D[\mathbf{x}] = 1$, where $E$ and $D$ are the statistical expectation operator and the statistical variance expectation operator, respectively. Let $\mathbf{u}$ denote a unit vector, also of dimension $p$, onto which the vector $\mathbf{x}$ is to be projected. This projection is defined by the inner product of vector $\mathbf{x}$ and $\mathbf{u}$, as shown by:

$$a = \mathbf{u}^T\mathbf{x} \tag{3.2.1}$$

subject to the constraint: $\|\mathbf{u}\| = (\mathbf{u}_j\mathbf{u}^T)^{1/2} = 1$.

The projection $a$ is a variable with a mean and variance related to the statistics of the measurement vector $\mathbf{x}$. The mean value of projection $a$ is zero, $E[a] = \mathbf{u}^T E[\mathbf{x}] = 0$, and the variance of $a$ is same as the mean-square value:

$$\sigma^2 = E[a^2] = \mathbf{u}^T E[\mathbf{x}\mathbf{x}^T]\mathbf{u} = \mathbf{u}^T\mathbf{R}\mathbf{u}$$

where $\mathbf{R}$ is the correlation matrix of the measurement vector as vector $\mathbf{x}$ has unit variance.

The variance, $\sigma^2$, of the projection $a$ is a function of the unit vector $\mathbf{u}$. It can be expressed as follows:

$$\psi(\mathbf{u}) = \sigma^2 = \mathbf{u}^T\mathbf{R}\mathbf{u} \tag{3.2.2}$$

on the basis of which $\psi(\mathbf{u})$ is defined as a *variance probe.*

### 3.2.1 The Eigen-Structure of the Principal Components

Principal Component Analysis is mainly concerned with finding a unit vector, $\mathbf{u}$, such that along $\mathbf{u}$ the variable probe, $\psi(\mathbf{u})$, has an extreme or stationary value (local maxima or minima), subject to the constraint of Euclidean norm of $\mathbf{u}$. The solution of this problem lies in the eigen-structure of the correlation matrix $\mathbf{R}$.

$$\mathbf{Ru} = \lambda\mathbf{u} \tag{3.2.3}$$

This eigenvalue problem (equation 3.2.3) has nontrivial solutions (i.e. $\mathbf{u} \neq \mathbf{0}$) only for special values of $\lambda$ that are known as the eigenvalues of the correlation matrix $\mathbf{R}$. The associated values of $\mathbf{u}$ are called the *eigenvectors.*

A correlation matrix is characterized by real, nonnegative eigenvalues. The associated eigenvectors are unique, assuming that the eigenvalues are distinct. Let the eigenvalues of the $p \times p$ matrix, $\mathbf{R}$, be denoted by $\lambda_0, \lambda_1, ..., \lambda_{p-1}$, and the associated eigenvectors be denoted by $\mathbf{u}_0, \mathbf{u}_1, ..., \mathbf{u}_{p-1}$, respectively. This satisfies the following equation:

$$\mathbf{Ru}_j = \lambda_j\mathbf{u}_j, \qquad j = 0, 1, ..., p-1 \tag{3.2.4}$$

Let the corresponding eigenvalues be arranged in decreasing order:

$$\lambda_0 > \lambda_1 > ... > \lambda_j > ... > \lambda_{p-1}$$

so that $\lambda_0 = \lambda_{max}$. Let the associated eigenvectors be used to construct a $m \times m$ matrix:

$$\mathbf{U} = [\mathbf{u}_0, \mathbf{u}_1, ..., \mathbf{u}_j, ..., \mathbf{u}_{p-1}]$$

then combining the set of $p$ equations represented in Eqn. 3.2.4 gives a single equation:

$$\mathbf{RU} = \mathbf{U\Lambda}$$

where $\mathbf{\Lambda}$ is a diagonal matrix defined by the eigenvalues of matrix $\mathbf{R}$:

$$\mathbf{\Lambda} = \text{diag}[\lambda_0, \lambda_1, ..., \lambda_j, ..., \lambda_{p-1}]$$

and

$$\mathbf{U}^T = \mathbf{U}^{-1} \tag{3.2.5}$$

From Eqns. 3.2.2, 3.2.4 and 3.2.5 it can be seen that the variance probes and the eigenvalues are equal, that is, $\psi(\mathbf{u}_j) = \lambda_j, \quad j = 0, 1, \ldots, p-1$.

Two important features can be seen from the eigen-structure of PCA:

1. The eigenvectors of the correlation matrix, $\mathbf{U}$, pertaining to the data vector $\mathbf{x}$ with the zero-mean and unit variance, define the vectors $\mathbf{u}_j$, representing the principal directions along which the variance probes, $\psi(\mathbf{u}_j)$, have their extreme values.
2. The associated eigenvalues define the extreme values of the variance probes $\psi(\mathbf{u}_j)$.

### 3.2.2 Derivation of the Principle Components

Principal Component Analysis requires that some of the variables in the data set are intercorrelated. If none of the variables are intercorrelated, there exists already a set of uncorrelated axes and there is no point in performing PCA.

The input vectors are random vectors $x$ with $K$ elements, representing the values of the signals at different time instants, and are mutually correlated. PCA linearly transforms vector $x$ to another vector $y$ with $N$ elements, $N < K$, so that the redundancy induced by the correlations is removed. This transformation is achieved by finding a rotated coordinate system such that the elements of $x$ in the new coordinates become uncorrelated. The data is concentrated into a set of orthogonal vectors that are able to describe the data to a good approximation, in a least squares sense. In addition to achieving uncorrelated components, the variances of the elements of $y$ will be strongly decreasing in most applications, with a considerable number of components so small that they can be discarded altogether [JB94].

If the variance-covariance matrix of the original variables is designated $\mathbf{C}$ then the variance-covariance matrix of the principal components $\mathbf{\Lambda}$ is:

$$\mathbf{\Lambda} = \mathbf{ACA}^T \tag{3.2.6}$$

In the matrix $\mathbf{A}$ the relationship between the variables and each principal component is given by the row vectors. Convert these relations to column vectors by defining a matrix $\mathbf{E} = \mathbf{A}^T$:

$$\mathbf{\Lambda} = \mathbf{E}^T\mathbf{CE} \tag{3.2.7}$$

Because the principal components are uncorrelated, their covariance terms are zero and the matrix $\mathbf{\Lambda}$ is a diagonal matrix. In the two dimensional case:

$$\mathbf{\Lambda} = \begin{Bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{Bmatrix}$$

where $\lambda_i$ represents the variance of the two principal components.

The principal components account for all the variance of the original variables, thus the sum of the $\lambda_i$ must equal the sum of the variances (the diagonal elements of $\mathbf{C}$), that is, $trace(\mathbf{\Lambda}) = trace(\mathbf{C})$. The correlation matrix $\mathbf{R}$ is related to the variance-covariance matrix $\mathbf{C}$ by:

$$R(i,j) = \frac{C(i,j)}{\sqrt{C(i,i)C(j,j)}} \tag{3.2.8}$$

Each element of the variance-covariance matrix $\mathbf{C}$ has been divided by the square root of the product of the $ith$ and $jth$ variances. Hence the variance-covariance matrix of the principal components can also be defined as:

$$\mathbf{\Lambda} = \mathbf{E}^T\mathbf{R}\mathbf{E} \tag{3.2.9}$$

where $\mathbf{\Lambda}$ will have different $\lambda_i$ from Eqn. 3.2.7, and:

$$trace(\mathbf{\Lambda}) = trace(\mathbf{R}) = p, \quad \text{the number of variables.} \tag{3.2.10}$$

since the diagonal values of $\mathbf{R}$ are unit.

The matrix $\mathbf{E}$ is computed in such a way that it becomes the matrix of normalised eigenvectors. This matrix describes the relationship of the principal components to the original variables when the variables have unit variance. The components have variances described by $\mathbf{\Lambda}$, and the matrix of component loading $\mathbf{L}$ is given by:

$$\mathbf{L} = \mathbf{E}\mathbf{\Lambda}^{\frac{1}{2}} \tag{3.2.11}$$

The component loadings are the correlation between each variable and each component, and can be seen as the projections of the unit variables onto the

principal components. That is, $L$ gives the weighted relationship of the principal components to the original variables. Squaring the elements of $\mathbf{L}$ produces the proportion of the variance of each variable "explained" by each component.

The final step in the PCA transform is to relate the original data points, in vector $\mathbf{x}$, to the principal components so that each data point may now be described by a vector $\mathbf{y}$:

$$\mathbf{y} = \mathbf{xE} \tag{3.2.12}$$

Eqn. 3.2.12 states that, the coordinates of any point measured on the component axes are the coordinates of that point measured on the variable axes multiplied by the matrix of eigenvectors. That is, the score of a data point on a particular component is simply a linear combination of that points scores on the original variables. These $y$ coordinates are known as component scores.

The component scores are the values of each data point in units of the principal components. The scores on any one component will have a mean of zero (by definition as the component axes are transformed from the variable axes by rotating about the multivariate mean) and a variance equal to the eigenvalue, $\lambda_i$, of that component (the eigenvalue is a measure of component length). The scores on any one component will also be uncorrelated with the scores on any other component (unless there is perfect correlation between two or more variables, then $p$ principal components are required to account for the $p$-dimensional variable space and hence all principal components are significant) [Dau74].

### 3.2.3 PCA Selection of the Key Variables

Having determined the principal components of the data set the question becomes; can the principal components be used to select a few key variables which adequately describe the data set and its variability? If an initial study of a large number of variables reveals some key variables in the data set then future analysis can be focused on monitoring these key variables. This has obvious practical advantages in saving time and money in monitoring the data in future.

Using principal components there are a number of techniques that can be used to determine the key variables (other techniques that do not employ PCA, for example, multiple correlation and clustering of variables, also exist). Once the key variables have been selected the other redundant variables can be discarded.

Two techniques have been used during the current work. First, PCA is performed on all the original $K$ variables. If $p$ variables are to be retained, a variable is associated with each of the last $(K-p)$ components and these $(K-p)$ variables are rejected. The variable associated with the principal component to be rejected is the variable which has the largest coefficient on the component (eigenvector).

The second technique is in a sense a backwards version of the first. Again PCA of all $K$ original variables is performed and again $p$ variables are to be retained. These $p$ components are considered successively, starting with the

largest. A variable is associated with each of the $p$ components to be retained, again by selecting the variable that has the highest loading on the component. These $p$ variables are retained and the remaining $K - p$ variables rejected.

The number of variables, $p$, to be retained is determined by the choice of $\lambda_0$. For the first method, all components with $\lambda_i$ less than the critical value, $\lambda_0$, are rejected. For the second method all components with $\lambda_i$ greater than $\lambda_0$ are retained. The value of $\lambda_0$ is problem specific but, $\lambda_0 \approx 0.7$ should give a good indication of how many variables to retain. Both of these techniques generally produce similar results [Jol72a, Jol72b].

### 3.2.4 Analysis of Component Scores

Because PCA is a transformation technique further analysis of the component scores themselves, that is, the transformed data, is often useful. As has been discussed above, a few of the $p$ components will account for most of the total variance. The question is, how many components (and their scores) should be considered?

One criterion is to eliminate all those components for which the null hypothesis, $\lambda_1 = \lambda_2 = \cdots = \lambda_p$, cannot be rejected. When $N$ is large, or when $N$ is small but large relative to $p$, only the smallest components, if any, will not be heterogeneous, leaving $m$ acceptable components where $m$ is not much less than $p$. Another criterion often used, is to eliminate all those components whose eigenvalues are less than 1.0; on the basis that these components are accounting for less of the total variance than any one variable.

Alternatively, using PCA it is possible to determine the proportion of the total variance accounted for by each component. Using this information it is then possible to decide how many components to eliminate. This can be achieved in two ways: firstly, by retaining those successive components that account for a set proportion, for example 90%, of the total variance; or secondly, by discarding those components whose individual contribution to the proportion of the total variance falls below some set level, for example 10%. Provided the heterogeneity of the eigenvalues of the principal components can be demonstrated the cut-off point is unimportant. The scores on the smaller components have smaller variances, thus their effect on the analysis is usually minimal [Str89].

## 3.3 Application of PCA to Industrial Data

### 3.3.1 Principal Component Analysis of a Power Transformer

This section is concerned with identifying the key variables of a power transformer and examining the interdependence of those variables using Principal Components Analysis. The aim is to develop a more comprehensive and systematic approach to the monitoring of changes in a power transformers operating conditions. The distribution of the transformers principal components is examined using an index number which has been introduced. The variation in the variance of the principal components of the power transformer is also reflected by this index. An analysis has been undertaken of data from transformers operating in both the normal and faulty conditions.

In modern industrial societies the maintenance of uninterrupted power supplies is of paramount importance. The provision of adequate monitoring systems for power systems is essential to avoid damage due to failures of apparatus' operation and degrading of their conditions. A power systems monitoring system should ensure that faulty equipment is identified and afterwards removed from service in the minimum of time so that; the faulty equipment is isolated and in order to limit the damage to equipment due to overheating, excessive mechanical forces, etc.

The power transformer is one of the most important pieces of apparatus

in a power system. Its in-service behaviour is vital to the systems operation. To keep their size acceptable, modern high voltage transformers are built with relatively tight insulation tolerances compared to older equipment and are consequently subject to increasingly high stresses in service. It is vital therefore to closely monitor their in-service behaviour in order to avoid catastrophic failures, costly outages and loss of production [Rog78, Duv89].

In order to minimise system outages, two strategies have evolved to monitor the transformers' conditions: (1) Various relays have been developed to respond to a severe power failure requiring immediate removal of the transformer from service, in which case outage is unavoidable, (2) On-line/off-line detectors are being developed to monitor the serviceability of transformers. This latter technique has received a great deal of attention in recent years [HY+97, MS+96]. In particular, the preventive techniques based on intelligent methodologies for early detection of faults to avoid system outages would be of great use. To this end, determination of the key variables of a transformer for monitoring and fault diagnosis purposes would be the first step in the development of intelligent monitoring systems.

A method is presented below, based on principal component analysis [ESM94, JB94, DKK96], to determine the key variables of a power transformer and the interdependence of those variables. An index number has also been defined to monitor any change in the distribution of the variance of the principal components of the transformer, which may be used for the purpose of condition monitoring.

### 3.3.2 Identification of Key Transformer Variables and Interdependence Analysis

The identification of the key transformer variables using PCA is achieved through the following steps:

1. Transform the original variables to the feature space using PCA.
2. Analyze the eigen-structure of the principal components.
3. Reconstruct the original measurement space using the principal components obtained from the selected variables.
4. Analyze the relationship between the original variables and principal components using the principal components loading.
5. Select a smaller set of original variables to maximize the information contained in the measurement space and verify the principal components properties in the feature space using the PCA transformation.
6. Select the key variables for the design of intelligent transformer monitoring and/or fault diagnosis systems.

#### 3.3.2.1 The Transformer Data

There are many variables available to measure and monitor the condition of a power transformer in service. These variables can be classified into two kinds. One is the common variables related to the transformer operation (TO), these variables are: *RH of Oil, T-B Oil Diff, PD in Neutral, Ambient RH, Ambient*

*Temp Bottom, Ambient Temp Top, Tank Temp 1, Tank Temp 2, MW, V, I WTI* and *Hydran*, giving a total of 13 variables. The other is the dissolved gas analysis (DGA) data, these variables are *hydran*, *hydrogen* ($H_2$), *methane* ($CH_4$), *ethylene* ($C_2H_4$), *acetylene* ($C_2H_2$), *water* ($H_2O$), *oxygen* ($O_2$), *nitrogen* ($N_2$), *ethane* ($C_2H_6$), *carbon monoxide* ($CO$), *carbon dioxide* ($CO_2$), *total dissolved combustible gases (TDCG)* and *total dissolved gases (TDG)*, again giving thirteen variables in total (however, four variables have been discarded since two of the variables have very small variance and two more are linear combinations of other variables).

The National Grid Company have provided two sets of data. The first set of data is from a transformer operating within it's normal range and consists of all the above variables. The second set of data is from a faulty transformer and consists of the following variables: $Hydran$, $AmbientRH$, $PD$, $TankTempBottom$, $TankTemp.Top$, $AmbTemp$. For the purpose of this analysis these same six variables will be selected from the data available for a normal transformer.

For convenience of numerical computation, the original measurements are normalized as follows:

Consider $p$ variables with $n$ observations, represented by a matrix $X : n \times p$:

$$X = \begin{bmatrix} x_1 & x_2 & \dots & x_p \end{bmatrix} \tag{3.3.1}$$

where $x_j$ is a column vector with $n$ dimension.

Let matrix $X' : n \times p$ represents the data set after normalizing the original

data set $X$. Its $j$th variable is calculated as follows:

$$x'_j = \frac{x_j - Mx_j}{Dx_j}, \qquad j = 1, \ldots, p \tag{3.3.2}$$

where $Mx_j$ and $Dx_j$ are the mean value and the standard deviation of $x_j$, respectively. By normalisation, each variable of the data set $X'$ has zero mean and unity standard variance.

As the variables already have unit variance, the correlation matrix of the input data set can be calculated as follows:

$$r_{ij} = E[x'_i x'_j] = \frac{1}{n}\sum_{k=1}^{n} x'_{ki} x'_{kj} \tag{3.3.3}$$
$$i = 1, \ldots, n; \qquad j = 1, \ldots, p$$

Applying the above equations, the normalised raw data of data sets 1 and 2 are shown in Figures 3.1 and 3.2.

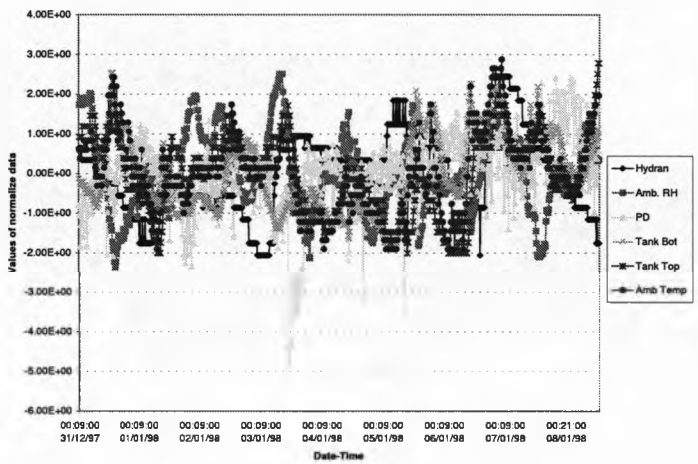


Figure 3.1: The normalized data of the normal transformer

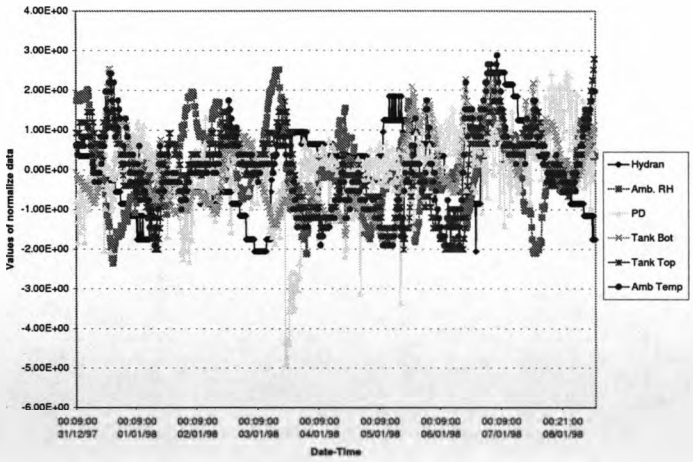


Figure 3.2: The normalized data of the faulty transformer

### 3.3.2.2 Principal Component Analysis of the Transformer Data

The PCA transform was applied to both data sets and the principal components for data sets 1 and 2 are shown in Figures 3.3 and 3.4 respectively. The variance of the principal components for data sets 1 and 2 are shown in Figures 3.5 and 3.6 respectively.

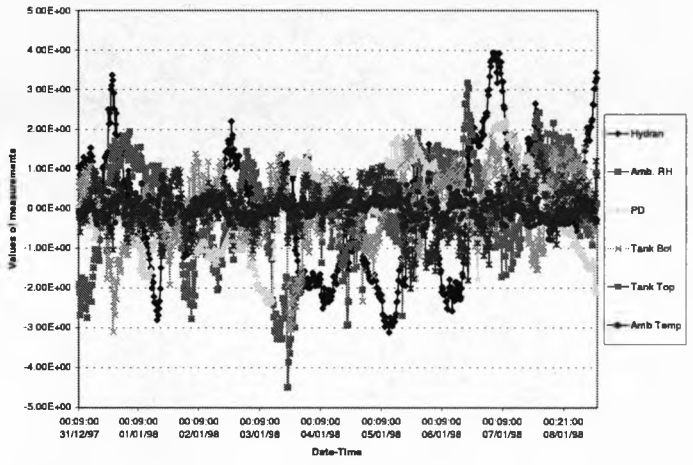


Figure 3.3: The principal components of the normal transformer

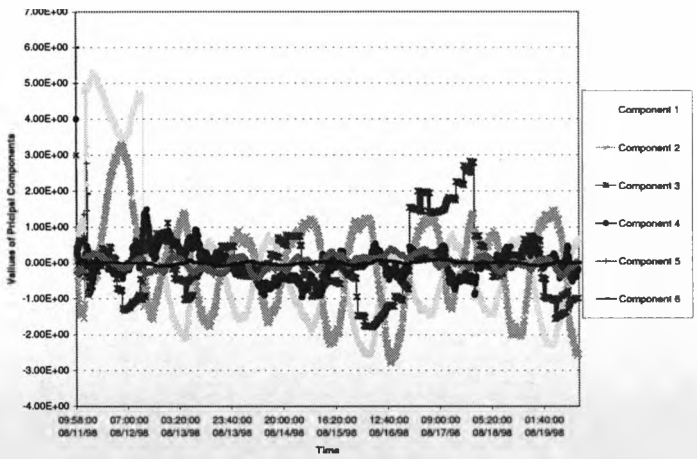


Figure 3.4: The principal components of the faulty transformer

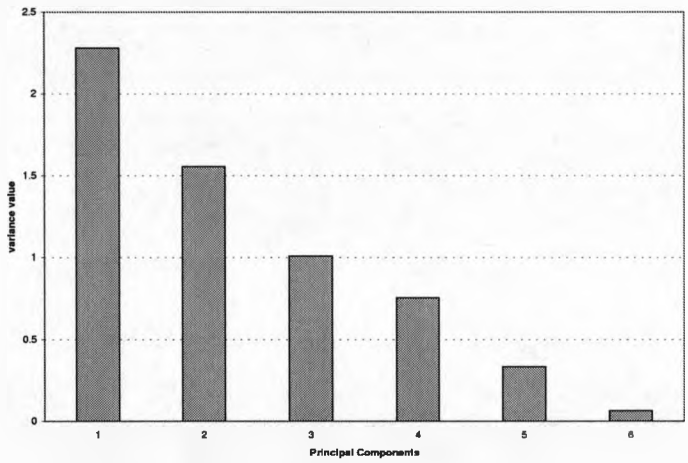


Figure 3.5: The variance of the principal components of the normal transformer

Figures 3.7 and 3.8 show the correlation coefficient matrices of the normal and faulty transformers respectively. The correlation of the variables of the two data sets changes when the operation of the power transformer changes from a normal to a faulty condition. This can be derived from the correlation matrices of the two data sets, as shown in Figures 3.7 and 3.8 respectively. The variance of the principal components of the two data sets is compared in

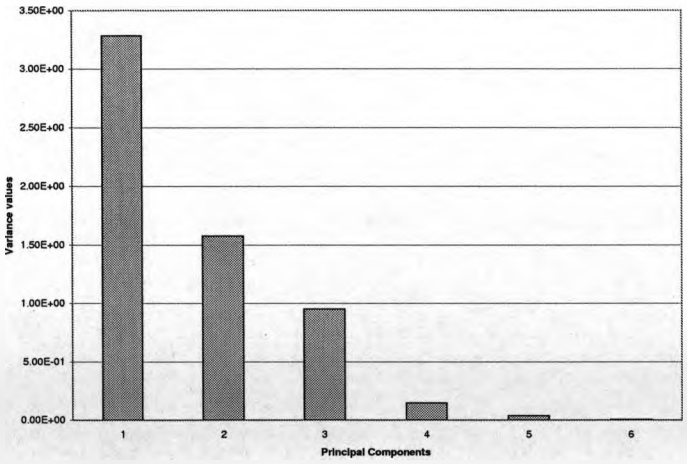


Figure 3.6: The variance of the principal components of the faulty transformer

Figure 3.9, which also demonstrates a change in the correlation of the variables in the data set of the faulty transformer.

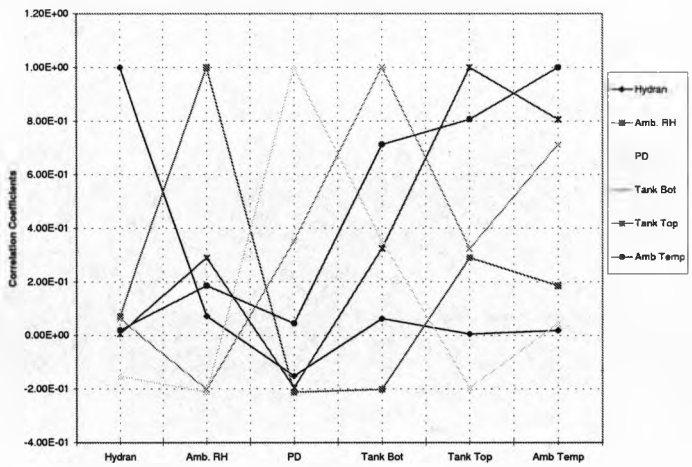


Figure 3.7: The correlation coefficients matrix of the normal transformer

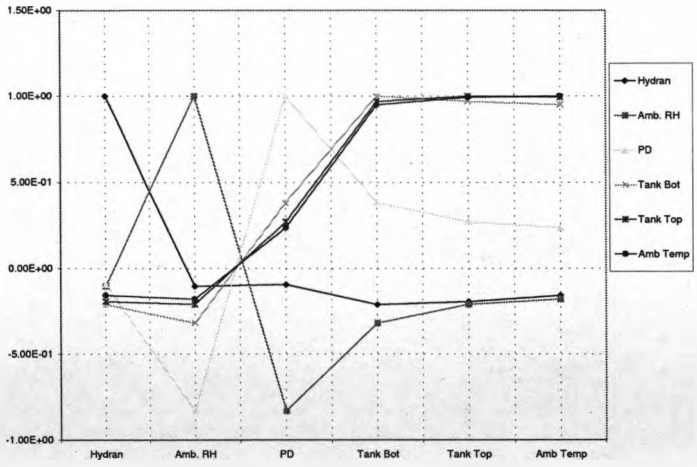


Figure 3.8: The correlation coefficients matrix of the faulty transformer

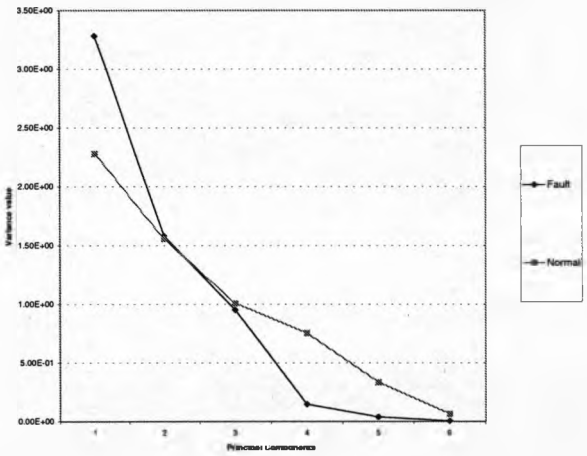


Figure 3.9: Comparison of the variance of the principal components for the normal and faulty transformers

#### 3.3.2.3 Variance Analysis of the Principal Components for Condition Monitoring

In order to investigate the varying tendency of the variance of the principal components, an index number $\delta$ is defined as follows:

$$\delta = \sum_{i=1}^{n-1} (\lambda_i - \lambda_{i+1})^2 \tag{3.3.4}$$

where $\lambda_i$ is the variance of the $i$th principal components, $i = 1, 2, \cdots, n$, where $n$ is the number of variables. As the variance value of the principle components is also the eigenvalue of the correlation coefficients matrix of the original variables, the change of correlation can be also represented by the value of such an index number.

Let $\delta_f$ and $\delta_n$ represent the index numbers of the faulty and normal transformers respectively. The index number calculated from the data sets of the faulty and normal transformers are:

$$\delta_f = 3.9662 \qquad \delta_n = 1.1384$$

and the ratio between them is

$$\delta_f / \delta_n = 3.48$$

Although $\sum_{i=1}^{n} \lambda_i = 1$, the distribution of the eigenvalues of the correlation coefficients matrix of the faulty transformer will be different to that of a normal transformer, which is indicated by the defined index number. Therefore, the index number could be used to determine the variations in a transformers operating condition, from the change in the distribution of the variance of principal components. A threshold value of the index number could also be determined if more data sets concerning the history of the transformer are available. This index number could then be used to report whether a transformer has a fault or not [LJWR00].

### 3.3.3 Principal Component Analysis of the Catalytic Cracker Data.

Shell (UK) have provided data from a catalytic cracker. The data consists of ninety five variables, sixty five input variables and thirty five output variables. The variables belong to two groups *catalyst*, sampled daily, and *process*, sampled weekly. The aim is to use v36-v95 to develop a predictive model for the responses v1-v35 (concentrating on predicting v4 and v6 initially). Because this is industrial process data there are lots of "drop-outs" and "spikes" which are not physical and need to be removed before any modelling is done. In addition there is a significant number of missing variables.

Variables indicated by $[\cdots]$ in Figure 3.10 are specific measurements of composition of fluid. The input feeds v36-v59 and v64-v66 are a mixture of hydrocarbons. The variables v2-v9, v10-v18, v19-v24 and v25-v35 are related responses of the system. v36-v95 are explanatory variables. Variables v60-v63 and v67-v72 are related process parameters. Variables v73-v 84 are other process parameters.

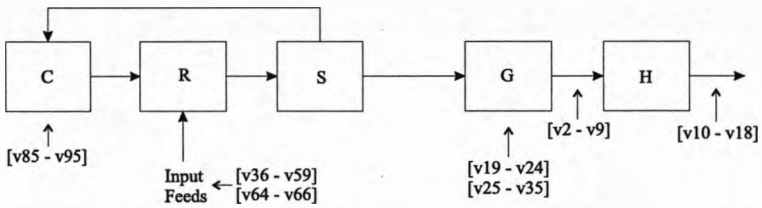


Figure 3.10: Catalytic cracker variables block diagram

Initially three groups of variables were selected for PCA; first, v36-v59, a

set of input hydrocarbons; second, v73-v84, other process parameters; third, v85-v95, the catalyst data. Because the nature of the data is unknown, for example, not all the variables are measured in the same units, and for ease of numerical computation all the original data measurements are converted to standard scores, that is, each variable of the data set has zero mean and unity standard deviation. If the data is a column vector, $\mathbf{x}$, of length $p$, then the $i$th variable of the standardised vector, $\mathbf{x}'$ is given by

$$x'_i = \frac{x_i - M(x_i)}{S(x_i)}, \quad i = 0, \ldots, p-1 \tag{3.3.5}$$

where $M(x_i)$ and $S(x_i)$ are the mean and standard deviation of $x_i$ respectively.

The raw data and the standardised data for variables v36-v59 is shown in Figures 3.11 and 3.12 respectively. The variance of the principal components of the standardised data is shown in Figure 3.13 and Figure 3.14 is a "scree" plot showing the percentage variability explained by each principal component. The scores on each principal component are shown in Figure 3.15.

If analysis of the component scores was required, for example using genetic classifiers or neural networks, then examination of Figures 3.13 and 3.14 give two, similar, alternatives of which principal components to select. Firstly, using Figure 3.13 to select only those principal components contributing more variability than any one variable would yield seven principal components for further analysis. Secondly, from Figure 3.14 it can be seen that the first nine principal components account for approximately 90% of the total variance of the standardised variables v36-v59. Thus the first nine principal components

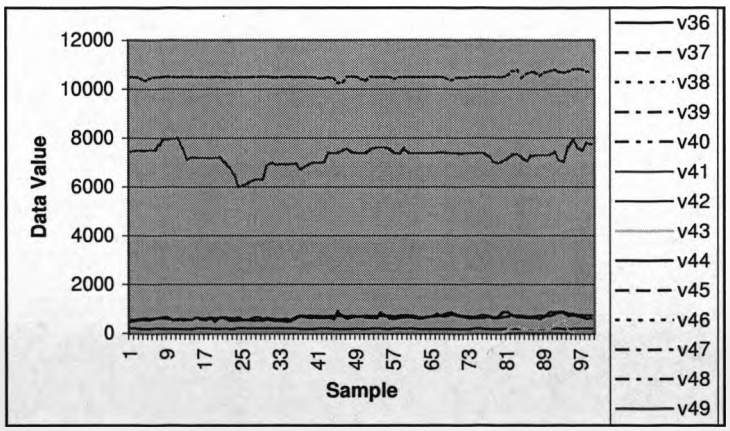


Figure 3.11: Variables: v36-v59

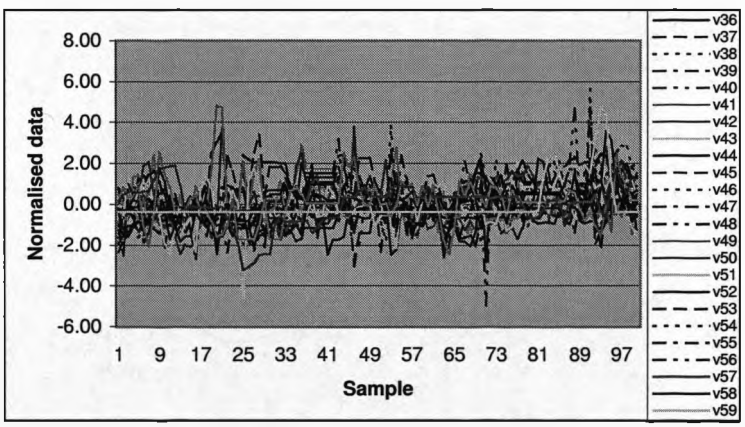


Figure 3.12: Normalised data: v36-v59

could be selected with a high degree of confidence that most of the variability of the data was being analysed. Given the lack of information about the variables in the data set, it is not possible to put a physical interpretation on the principal components for this analysis.

The raw data and the standardised data for variables v73-v84 is shown in

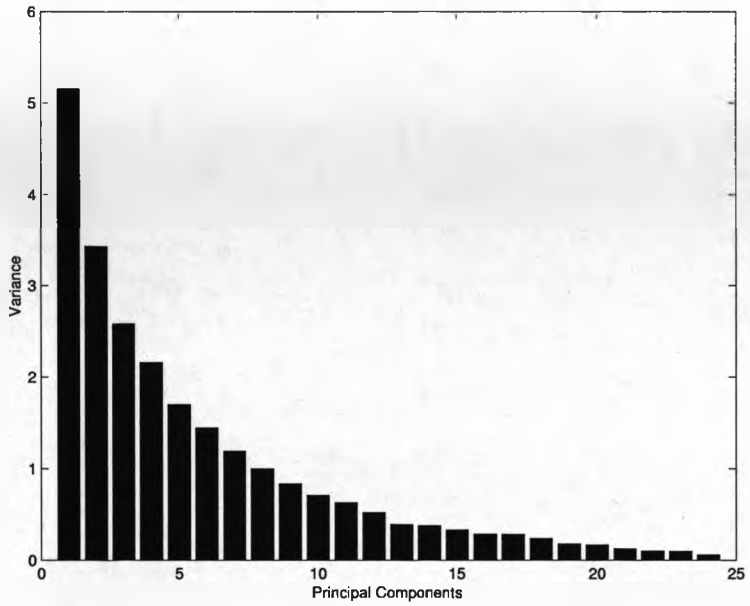


Figure 3.13: Principal components: v36-v59

Figures 3.16 and 3.17 respectively. The variance of the principal components of the standardised data is shown in Figure 3.18 and Figure 3.19 shows the percentage variability explained by each principal component. The scores on each principal component are shown in Figure 3.20. Table 3.1 shows the principal components of the data.

From Figure 3.19 it can be clearly seen that the first principal component accounts for nearly 80% of the total variance of the standardised variables v73-v84. The fact that the first principal component accounts for such a large

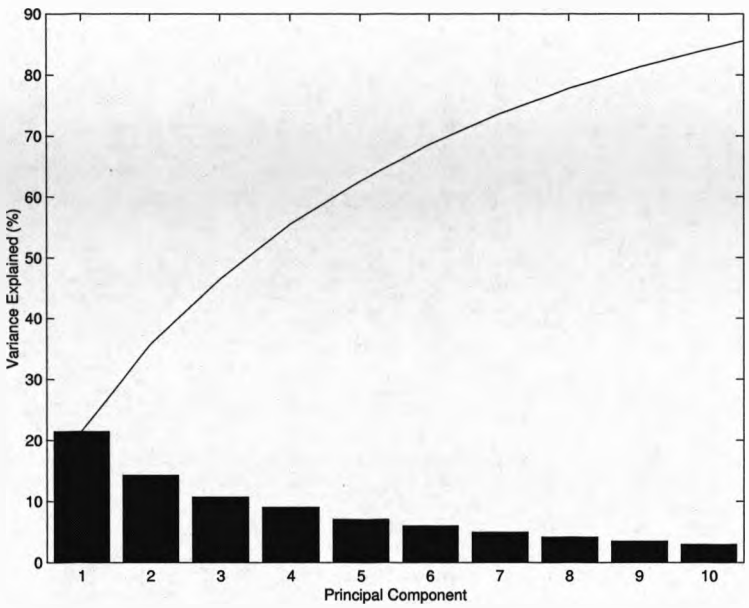


Figure 3.14: Variability explained (%) of principal components: v36-v59

proportion of the total variance of this data set implies that the variables v73-v84 are highly correlated. The first four principal components account for over 90% of the total variance and could be used directly in further analysis of the data.

If selection of the key variables from the group v73-v84 was required then analysis of Table 3.1 would reveal which variables to select. Using the first technique described above, that of rejecting the last $K - p$ variables, then analysis of Table 3.1 shows that the variable with the highest loading on the

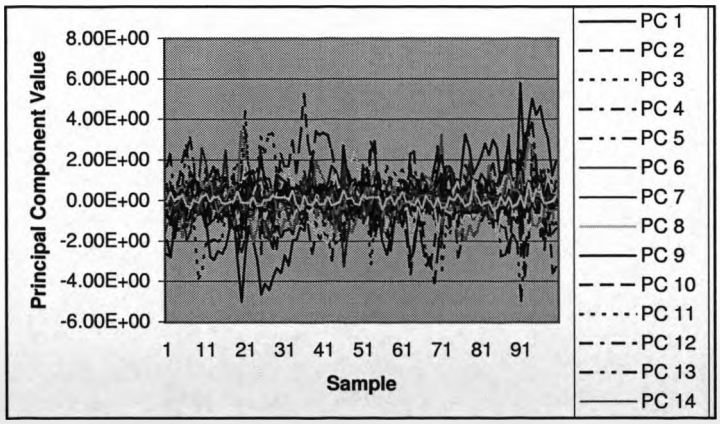


Figure 3.15: Component scores: v36-v59

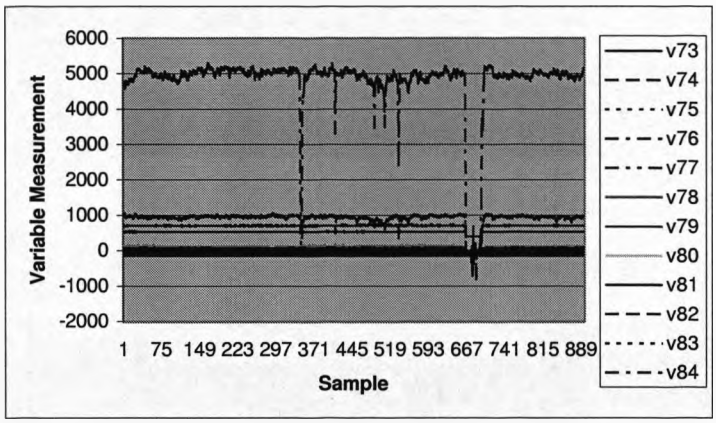


Figure 3.16: Variables: v73-v84

last principal component, PC 12, is variable two, that is, v74 (with a loading of 0.759), which would therefore be rejected. The next variable to be rejected would be that with the highest loading on PC 11, that is, v78, with a loading of 0.5421. And so on.

Alternatively, using selection of the first $p$ variables, the variable to be

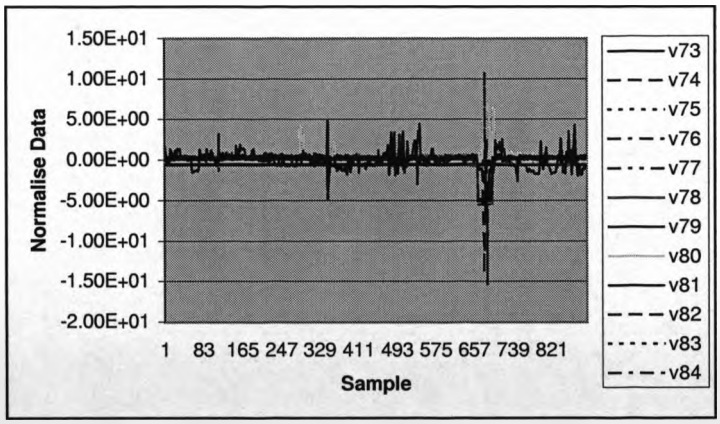


Figure 3.17: Normalised data: v73-v84

retained on the first principal component, PC 1, would be variable three, that is, v75, with a loading of 0.3241. The second variable retained would be v81, with a loading of 0.9745 on PC 2. And so on.

The raw data and the standardised data for variables v85-v95 is shown in Figures 3.21 and 3.22 respectively. The variance of the principal components of the standardised data is shown in Figure 3.23 and Figure 3.24 shows the percentage variability explained by each principal component. The scores on each principal component are shown in Figure 3.25. Table 3.2 shows the principal components of the data.

Figures 3.23 and 3.24 indicate that the variance of the principal components is much more evenly distributed than for the previous set of variables. This even distribution of the principal components implies that the variables v85-v95 are less highly correlated. From Figure 3.24 it can be seen that eight principal components are required to account for 80% of the total variance

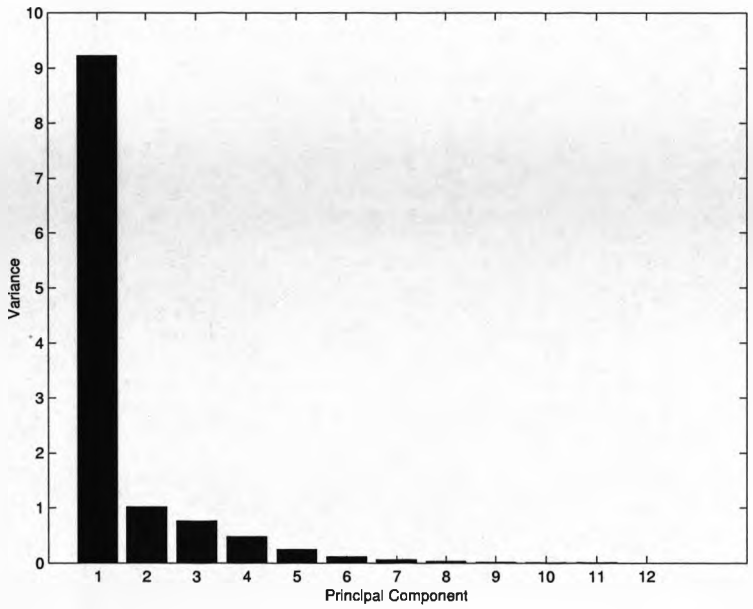


Figure 3.18: Principal Components: v73-v84

of the data set. Thus a much smaller reduction in the dimensionality of the variables v85-v95 is possible than for variables v73-v84, if it is required to keep the same percentage of the total variance.

Using Table 3.2 the key variables from this data set can be selected. Using the rejection method, the first variable to be rejected would be v92 with a loading of 0.7319 on the last principal component, PC 11. The second variable to be rejected would be v95 with a loading of 0.5315 on PC 10. Alternatively using the retention method the first variable retained would be v92 with a

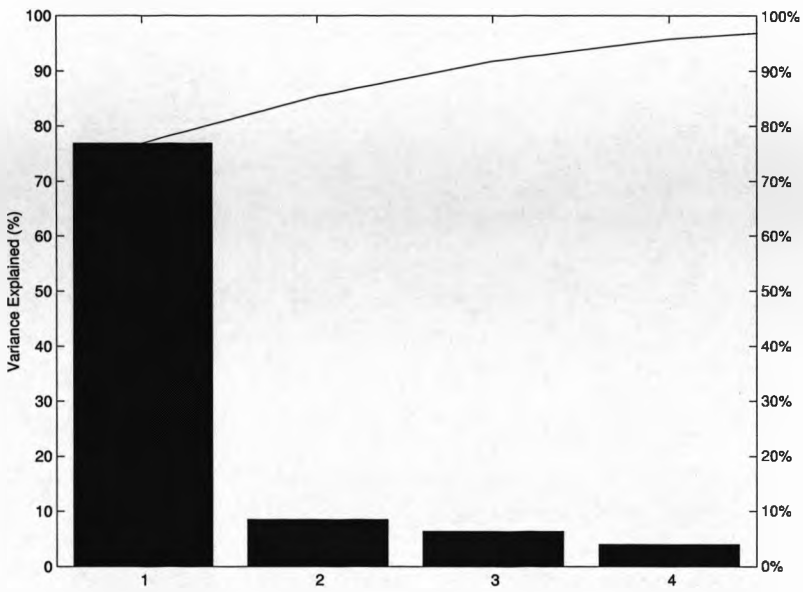


Figure 3.19: Variability explained (%) of principal components: v73-v84

loading of -0.5368 on PC 1. It can be seen from this that the two techniques do not always retain the same subsets.

Define four categories of subsets: Best, Good, Moderate and Bad. The first technique for selecting key variables, that is, that of rejecting those variables with the highest loading on the last $K-p$ principal components, retains less of the best subsets than the second method, that is, that of retaining the variables with the highest loading on the first $k$ principal components. However, if the best subsets are only a little better than the good subsets, then the first method

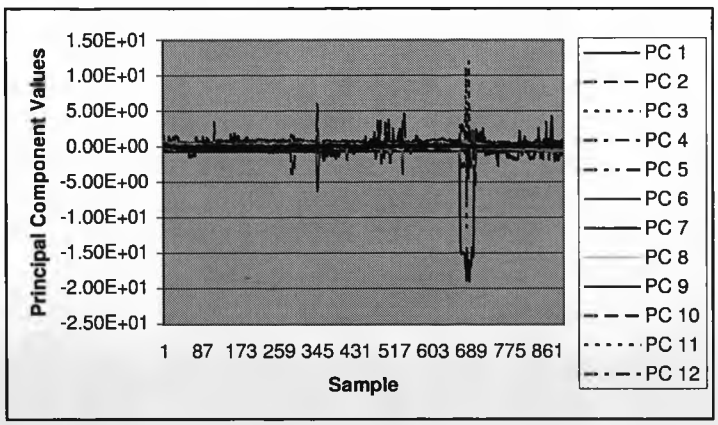


Figure 3.20: Component scores: v73-v84

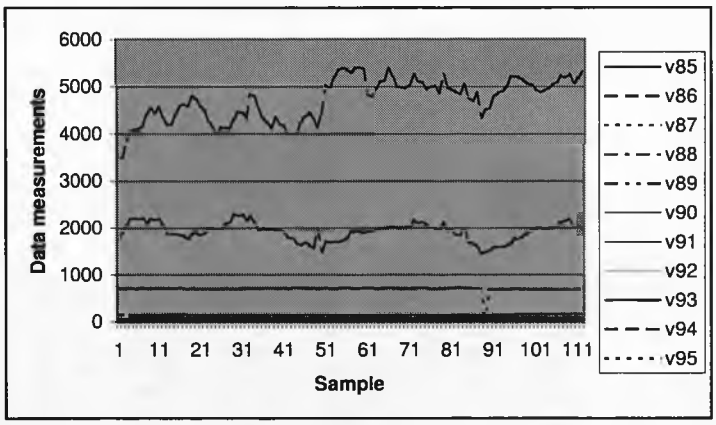


Figure 3.21: Variables: v85-v95

of selecting variables is superior to the second method [Jol72a].

The above analysis has been performed using the mathematical computing package Matlab. A principal component neural network based on the generalised Hebbian algorithm has been written in Visual C++. While this network

| PC1 | PC2 | PC3 | PC4 | PC5 | PC6 |
|---|---|---|---|---|---|
| 0.3207 | −0.0418 | 0.0561 | −0.0254 | 0.0818 | −0.4991 |
| 0.3239 | −0.0443 | 0.0652 | −0.0347 | 0.1362 | −0.3546 |
| 0.3241 | −0.0250 | 0.0756 | −0.0477 | −0.0169 | −0.2427 |
| 0.3154 | −0.0937 | 0.1134 | −0.087 | 0.398 | 0.2417 |
| 0.3225 | 0.0132 | 0.0763 | −0.0455 | −0.1106 | −0.219 |
| 0.3129 | 0.0621 | 0.0607 | −0.0151 | −0.5423 | −0.0763 |
| 0.3059 | 0.084 | 0.0832 | −0.0084 | −0.5818 | 0.4517 |
| −0.2356 | −0.096 | 0.3336 | 0.8999 | −0.0971 | −0.0579 |
| 0.0272 | 0.9745 | 0.1134 | −0.0876 | 0.1636 | −0.0031 |
| 0.1808 | 0.063 | −0.8997 | −0.3907 | 0.0115 | 0.0328 |
| 0.3151 | −0.1095 | 0.1156 | −0.1088 | 0.2948 | 0.4328 |
| 0.3224 | −0.0312 | 0.1031 | −0.0636 | 0.2191 | 0.2414 |

| PC7 | PC8 | PC9 | PC10 | PC11 | PC12 |
|---|---|---|---|---|---|
| 0.5037 | −0.0635 | 0.158 | −0.0224 | 0.0093 | −0.5937 |
| 0.3028 | −0.1053 | −0.0198 | 0.2217 | 0.1223 | 0.759 |
| −0.319 | −0.0332 | −0.6465 | −0.0171 | −0.55 | −0.0594 |
| −0.0907 | 0.5384 | −0.252 | 0.3011 | 0.4189 | −0.1634 |
| −0.4313 | 0.4508 | 0.6042 | −0.1657 | −0.1891 | 0.1013 |
| −0.3563 | −0.372 | −0.0541 | 0.1394 | 0.5421 | −0.1131 |
| 0.4692 | 0.2694 | −0.0556 | 0.033 | −0.223 | 0.0573 |
| 0.0298 | 0.0084 | −0.0085 | −0.0163 | 0.0098 | 0.0078 |
| 0.0005 | −0.0235 | 0.0003 | 0.039 | −0.0078 | −0.0129 |
| 0.0026 | 0.0073 | −0.0034 | −0.0023 | 0.0013 | −0.0004 |
| −0.0922 | −0.4965 | 0.3407 | 0.346 | −0.3029 | −0.0819 |
| 0.0468 | −0.1803 | −0.0773 | −0.8306 | 0.1887 | 0.0888 |

Table 3.1: Scores on the principal components: v73-v84

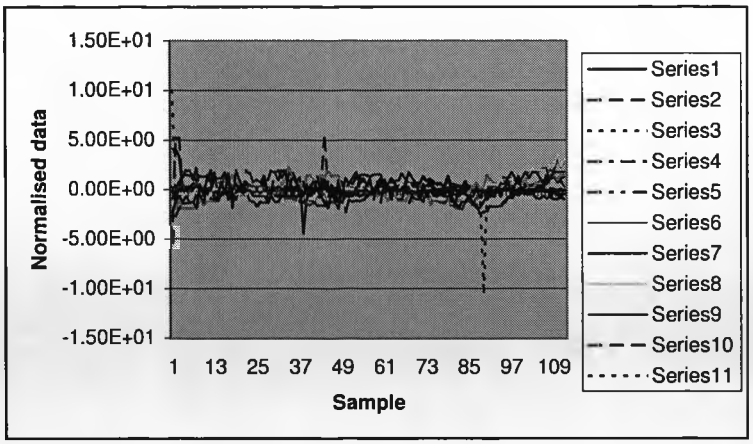


Figure 3.22: Normalised data: v85-v95

converges consistently to the same values, these values do not entirely agree with those expected using standard techniques for determining the principal components. More analysis is required to determine the degree of correlation between the two sets of results. It may be necessary to examine some of the other algorithms for neural network extraction of the principal components, for example, Robust PCA or nonlinear PCA [Xu95, KOW+97, Plu95].

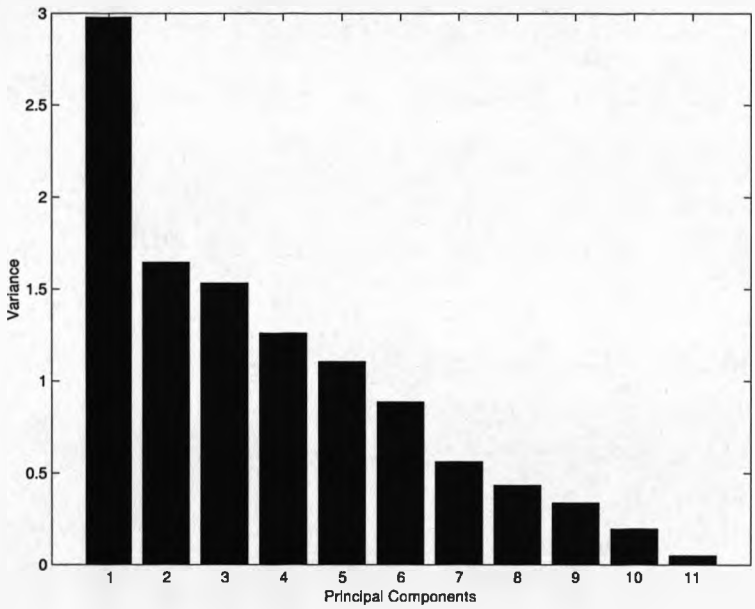


Figure 3.23: Principal components: v85-v95

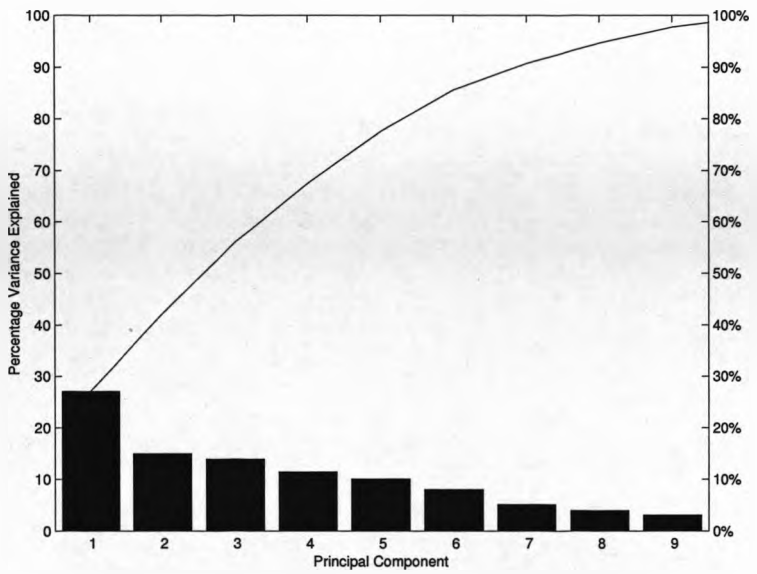


Figure 3.24: Variability explained (%) of principal components: v85-v95

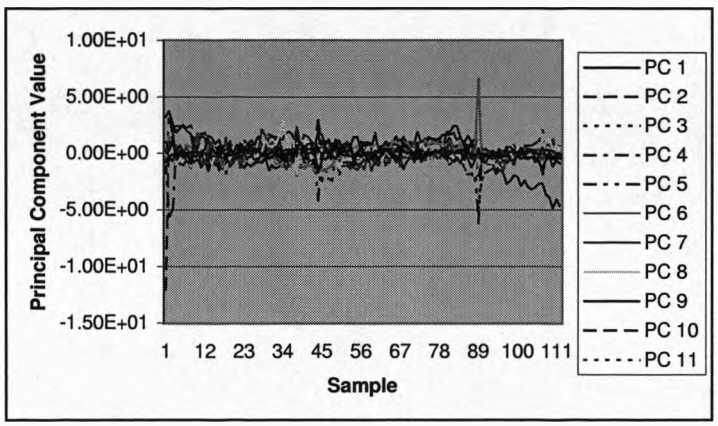


Figure 3.25: Component scores: v85-v95

| PC1 | PC2 | PC3 | PC4 | PC5 | PC6 |
|---|---|---|---|---|---|
| 0.2345 | 0.3251 | 0.0299 | −0.5617 | 0.2883 | 0.0671 |
| 0.0802 | 0.5009 | 0.1446 | −0.5308 | 0.2837 | 0.1137 |
| 0.035 | −0.0651 | 0.2919 | 0.1775 | 0.5788 | −0.6866 |
| 0.0366 | −0.0347 | 0.683 | 0.2034 | −0.0162 | 0.1246 |
| −0.3693 | 0.0713 | −0.0631 | 0.4023 | 0.2747 | 0.2675 |
| −0.2477 | 0.2277 | −0.3534 | −0.1387 | −0.2283 | −0.5936 |
| −0.4485 | −0.1261 | 0.2861 | −0.3216 | −0.1898 | −0.0026 |
| −0.5368 | −0.0247 | 0.1228 | −0.1855 | −0.0395 | 0.0431 |
| −0.4763 | −0.1573 | 0.0557 | −0.0779 | 0.1908 | −0.0158 |
| 0.1298 | 0.2191 | 0.444 | −0.0072 | −0.5106 | −0.231 |
| 0.0725 | −0.7012 | −0.0222 | −0.0698 | −0.197 | −0.1275 |

| PC7 | PC8 | PC9 | PC10 | PC11 |
|---|---|---|---|---|
| −0.4 | 0.3107 | 0.0027 | −0.4185 | 0.0599 |
| 0.0531 | −0.0212 | −0.2722 | 0.5149 | −0.0638 |
| 0.035 | −0.2381 | −0.0904 | −0.0714 | −0.0144 |
| 0.2776 | 0.6238 | 0.0135 | 0.0409 | 0.0761 |
| −0.4895 | 0.1724 | −0.4803 | −0.0126 | −0.2134 |
| 0.0205 | 0.57 | −0.1071 | −0.0094 | −0.0348 |
| 0.212 | −0.1911 | −0.0872 | −0.2683 | −0.6329 |
| 0.0971 | −0.1702 | −0.2421 | −0.1602 | 0.7319 |
| −0.2759 | 0.0736 | 0.6702 | 0.4146 | −0.0063 |
| −0.6231 | −0.1776 | −0.0072 | 0.027 | 0.0572 |
| −0.0111 | 0.003 | −0.4 | 0.5315 | −0.0014 |

Table 3.2: Scores on the principal components: v85-v95

### 3.3.4 Eigenvalue Analysis of the Oil Mist Data

Eigenvalues indicate the natural frequency of a system. Therefore the Shell oil mist data was reduced to a 3 by 3 correlation matrix $R$ (equation 3.2.3) and the eigenvalues used to create input vectors for neural network training.

Figure 3.26 shows a scatter plot of the first 5 samples of the eigenvalues of all four alarm conditions and indicates that a neural network will have difficulty separating the patterns in the pattern space. In order to reduce the difficulty of separating the patterns, it was decided to select eigenvalues of only two alarm conditions (High and Low). Figure 3.27 shows a plot of the first 7 eigenvalues for the High and Low alarm conditions.

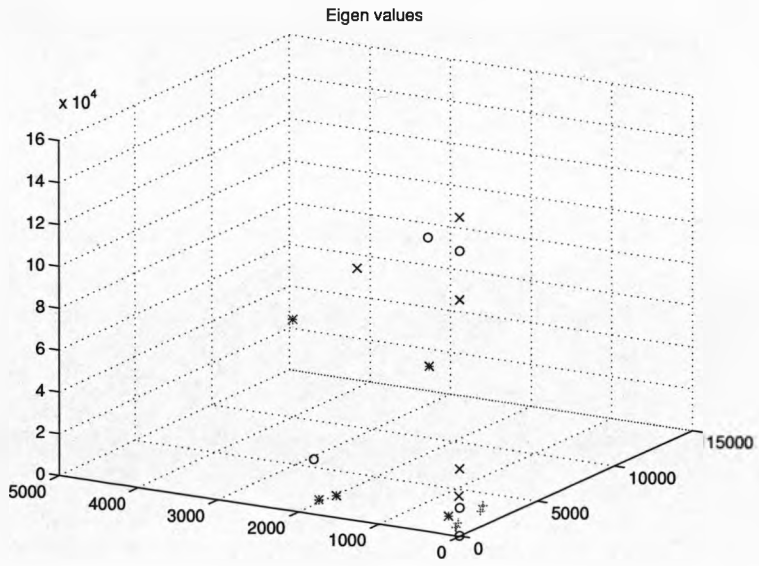


Figure 3.26: The first 5 eigenvalues of the High, Low, Grey and None oil mist data

The eigenvalues where then used to create a neural network input vector and the target vector shown below in Table 3.3 was created. These vectors

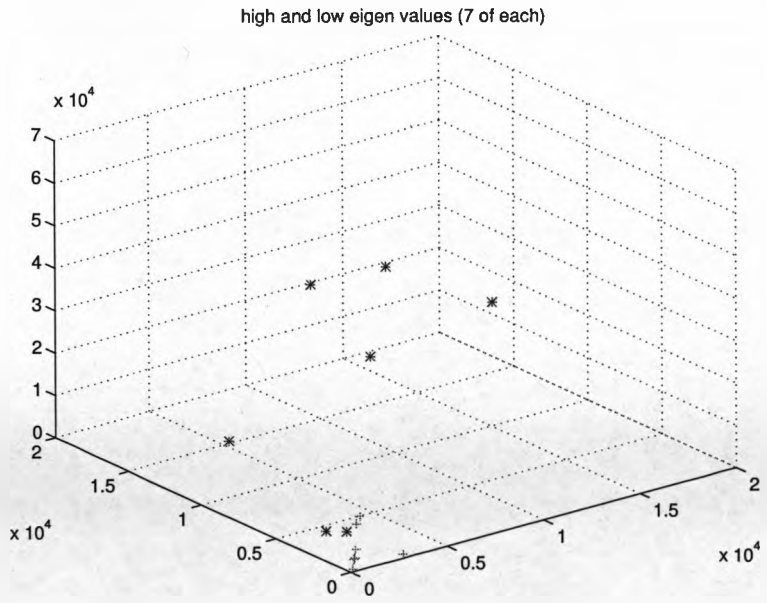


Figure 3.27: The first 7 eigenvalues of the High and Low oil mist data

where then used to train a whole range of multilayer perceptron neural networks. These networks had varying numbers of neurons in both the hidden and output layers and, despite some long training times (with up to 500,000 epochs elapsing) none of the neural networks trained correctly. Figure 3.28 shows a typical networks sum squared errors during training.

| | |
|---|---|
| High Alarm | 1 |
| Low Alarm | 0 |

Table 3.3: Neural network target vector for High and Low alarm conditions

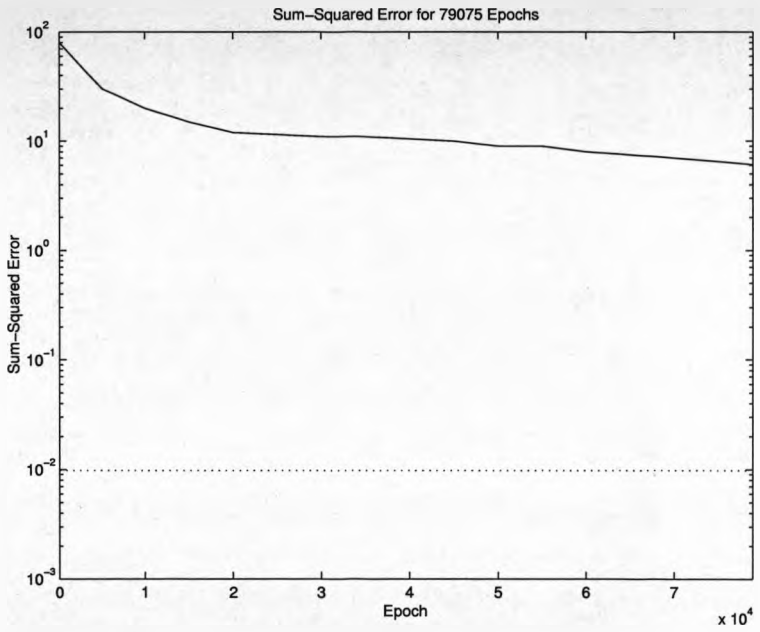


Figure 3.28: Neural network training errors for High and Low eigenvalues

## 3.4 Conclusion

Principal Components Analysis can be used to identify any variation in the correlation of the transformers operating variables caused by changes in the transformers condition. Considering the results shown above, comparison of the normal and faulty transformers, suggests that the key variables selected by PCA under a normal operating condition, will become more correlated and the variance of the principal components will have a more uneven distribution if the transformers operating condition changes. This is due to the key variables selected by PCA forming a maximum information space. Any change in the transformer condition, represented by the same variables, will result in a contraction of the information space. An index number has been introduced to measure the distribution of the variance of the principal components. It also reflects the tendency of the correlations of the key transformer parameters to vary. It has been found that the ratio between the index numbers of the faulty and normal transformers is sufficiently large that it could be used to report the variation of the transformers condition. The index number could therefore be used as an alarm for the purposes of fault monitoring.

As the data from the normal and faulty transformers were not measured from identical transformers and they were sampled within a limited operation range, the sensitivity of PCA as applied to condition monitoring could not be investigated. The quantitative analysis of this method requires more data, obtained from a variety of different transformer conditions, in order to pro-

vide an accurate assessment of the condition variation. However, the trend is clearly positive and suggests that the methodology could be used for condition monitoring. In contrast to the conventional correlation analysis methods which have been applied, for example monitoring of transformer partial discharges, the PCA provides a more systematic and comprehensive approach to analysis of correlation variations occurring as the transformer condition changes. Although the PCA cannot be used directly for fault diagnosis, the information extracted by the PCA could form a maximum-information space for further classification leading to an identification of the details of transformer condition variations.

The use of principal component analysis on the catalytic cracker data has allowed the key variables in the various data sets to be identified. Although use of the rejection method and the retention method for selecting which variables to keep provides some discrepancies, there is a large degree of agreement as to which variables account for the main principal components. Once selected, these variables could be used in conjunction with either neural networks or learning expert systems to provide further classification of the catalytic cracker data.

Use of the principal components (eigenvalues) of the infra-red scatter data, has hugely reduced the dimensionality of the pattern space. However, this has not produced a set of patterns that have been separable by a multilayer perceptron neural network. Backpropagation networks with a wide range of neurons in the hidden and output layers have so far been unable to reach

the desired error goal despite some networks training for up to 500,000 epochs. These networks have in fact largely failed to correctly classify training patterns during generalisation and have therefore have not been capable of classifying correctly any previously unseen test vectors.

These results show that PCA can be used for identifying the key variables of the catalytic cracker data and reducing the dimensionality of the infra-red scatter data. However, PCA was not able to produce data that could be used for classification by a neural network. As a result of these findings work moved on to examine the use of the wavelet transform as a method of information extraction.

# Chapter 4

# The Wavelet Transform for Information Extraction

## 4.1 Introduction

The fundamental idea behind wavelets is to analyse according to scale. As with Fourier analysis, wavelets use superposition of functions, however, in wavelet analysis the scale that is used to look at the data plays a special role. Wavelet algorithms process data at different *scales* or *resolutions*. If a signal is examined using a large "window" then its gross features are observed. If the signal is examined using a small "window" then its detailed features are observed. The result is that using wavelet analysis it is possible to see both the "wood" and the "trees" [Gra95].

The procedure used in wavelet analysis is to adopt a wavelet prototype

function called an *analysing wavelet* or *mother wavelet.* Temporal analysis is performed with a contracted, high frequency version of the analysing wavelet. Frequency analysis is performed with a dilated, low frequency version of the same wavelet. Unlike the Fourier transform which only has a single set of basis functions, that is, the sine and cosine functions, the wavelet transform has an infinite set of possible analysing functions. Therefore, wavelet analysis can provide information that may be obscured by the use of Fourier analysis [BH96, Gra95].

There are two possible ways that the wavelet transform could be used in the development of an intelligent monitoring and prediction system. Like the Fourier transform the wavelet transform is an orthonormal basis function. The wavelet transform could therefore be used to preprocess the data prior to input to the neural network. Wavelet transforms are capable of a sparse representation of the data, while still allowing the data to be reconstructed from the wavelet coefficients. Secondly, is the use of *wavelet neural networks.* Due to the similarity between wavelet decomposition and one hidden layer neural networks, the idea of combining both wavelets and neural networks has emerged [Joh, MCKDV97]. One possible route for this combination is as a special case of radial basis function networks. In this case the radial basis functions of the hidden layer are replaced by wavelet functions. The wavelet neural network provides an efficient representation of a given function. However, they are not generally RBF networks since multi-dimensional scaling functions can be non radialsymmetric [ZB92, ZWML95, Zha97].

## 4.2 Limitations of the Fourier Transform

Fourier analysis works with linear problems. Nonlinear problems tend to be much harder, the behaviour of nonlinear systems is much less predictable: a small change in input can cause a big change in output. Another limitation of the Fourier transform is that it hides information about time [BH96]. The Fourier transform allows one to switch between the time and frequency domains of a signal. However, only one of them is available at a time. That is, no frequency information is available in the time domain signal and no time information is available in the Fourier transformed signal [Gra95].

The Fourier transform tells us how much of each frequency component is in the signal, that is its spectral content, but the information on *when in time* these spectral components exist is buried deep within the phases of the Fourier transform. In theory the time information can be extracted by calculating the phases from the Fourier coefficients. In practice, computing them with enough precision is impossible [BH96]. For a stationary signal, that is, one whose frequency content does not change in time, this presents no problems.

However, a problem arises for non-stationary signals. Fourier analysis is poorly suited to very brief signals or signals that change suddenly and unpredictably [Str93, Ued95]. Consider two signals. Firstly, Signal 1 that has four frequencies, say 5, 10, 20 and 40 Hz sinusoids, present at all times as shown in figure 4.1. Secondly, Signal 2 that has these frequencies present for a set time, say the first interval 0 to 250ms has the 5Hz sinusoid present, the second

interval 250 to 500ms has the 10Hz sinusoid present and so on, as shown in Figure 4.2.

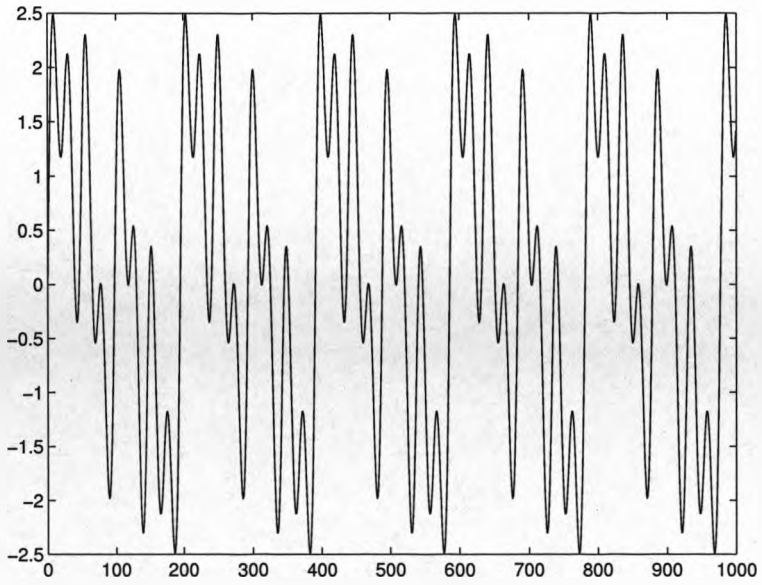


Figure 4.1: Signal 1

Both of these signals will have essentially the *same* Fourier transform. This is because, as can be seen from equation 2.3.12, the signal is multiplied by an exponential term at some frequency, $\omega$, and integrated over *all time*. The Fourier transform gives the spectral content of a signal. The information about one instant of a signal (in the time domain) is dispersed throughout all the frequencies of the entire transform. Hence, a *local* characteristic of the signal becomes a *global* characteristic of the Fourier transform [BH96].

Figure 4.3 shows the FFT of Signal 1 and Figure 4.4 shows the FFT of Signal 2. It can be seen from these two figures that both signals have similar Fourier transforms. Both transforms have major frequency components at 5,

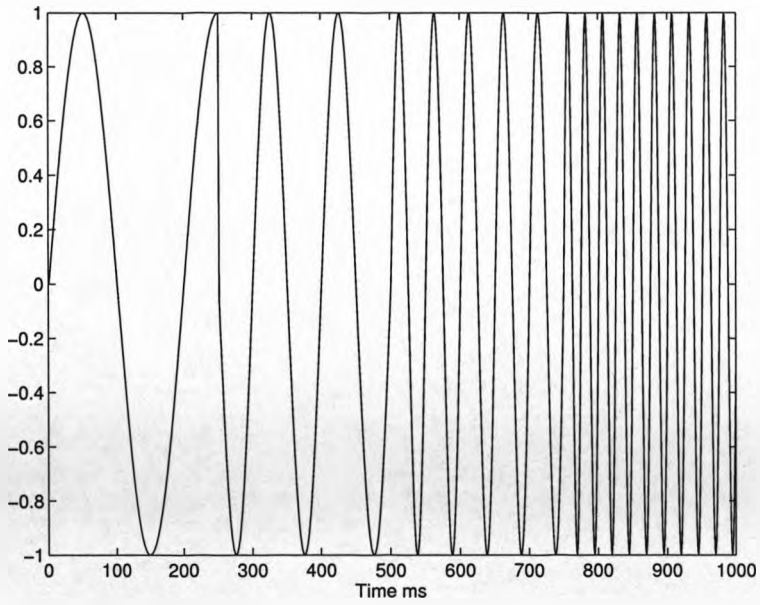


Figure 4.2: Signal 2

10, 20 and 40Hz. The noise like components in Figure 4.4 are due to the abrupt changes in frequency in the second signal. Although these other frequencies exist in the signal, they have a small amplitude in the Fourier transform because they are not major spectral components of the signal.

When the time localisation of the spectral components of a non-stationary signal are required then the Fourier transform is inadequate. The *wavelet transform* does however provide information about the time-frequency representation of a signal [Pola], as do others, for example, the short time Fourier transform. Figures 4.5 and 4.6 show the wavelet transforms of signals one and two respectively.

Note that the axes are translation and scale not time and frequency. However, translation is strictly related to time, since it indicates where the mother

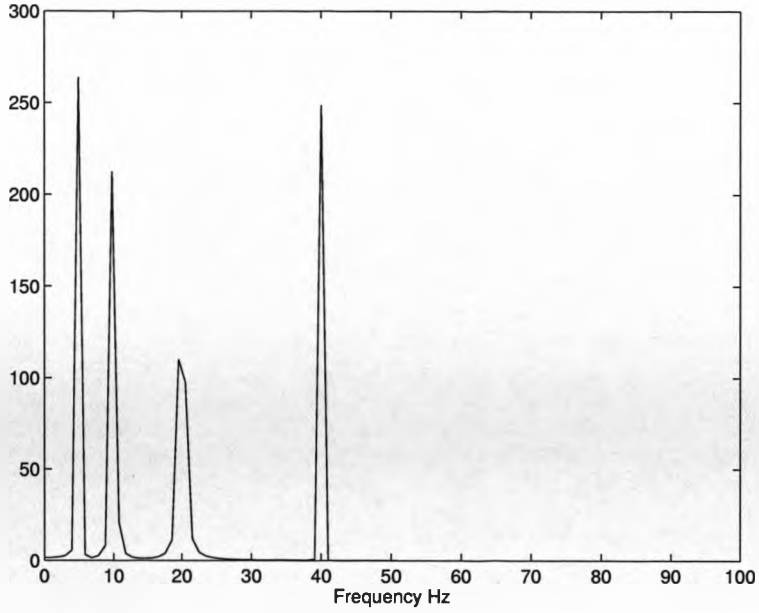


Figure 4.3: The fast Fourier transform of Signal 1

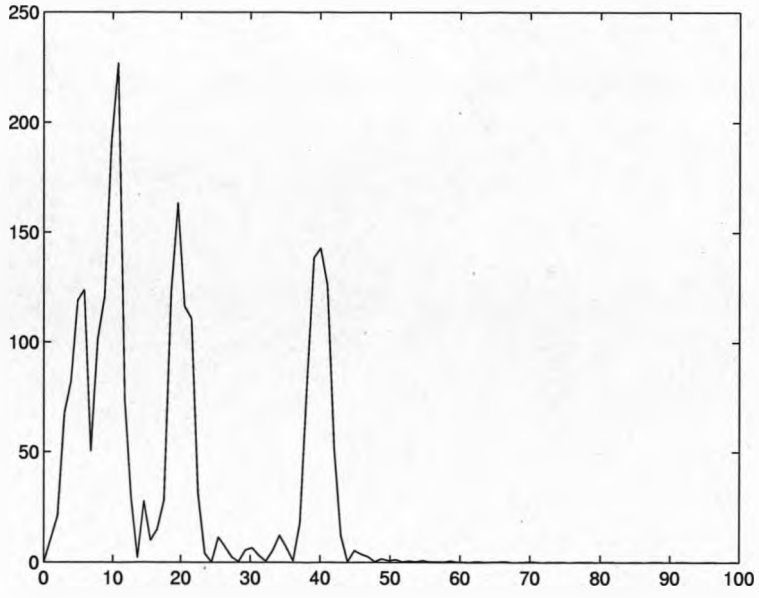


Figure 4.4: The fast Fourier transform of Signal 2

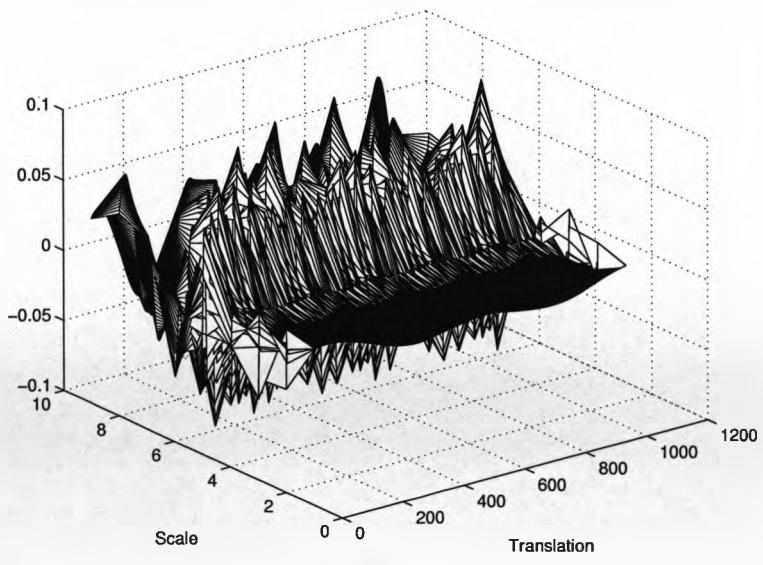


Figure 4.5: The wavelet transform of signal 1

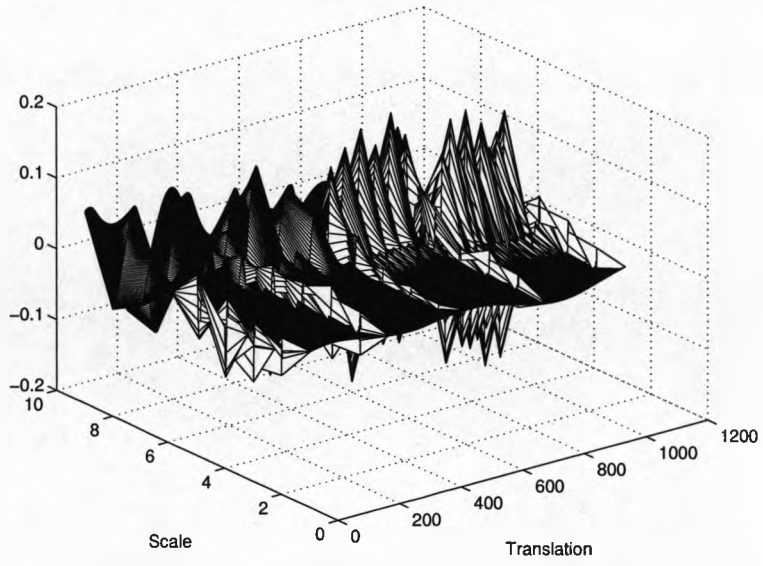


Figure 4.6: The wavelet transform of signal 2

wavelet is located. The translation of the mother wavelet can be thought of as the time elapsed since $t = 0$. The scale however is actually the inverse of frequency. Smaller scales correspond to higher frequencies and frequency decreases as scale increases [VM91]. Therefore, the portion of the graph with scales around zero correspond to the highest frequencies in the analysis, in this case 40Hz. It can be seen from Figure 4.5 that all frequencies are present for all times. Figure 4.6 is a little difficult to interpret. However, it can be seen that each frequency component exists at separate times. It can also be seen from Figure 4.6 that the wavelet transform has good time and poor frequency resolution at high frequencies and good frequency and poor time resolution at low frequencies [BH96]. In addition wavelet analysis has advantages over Fourier analysis if the signal has transient behaviour or discontinuities [BDG96].

## 4.3 The Continuous Wavelet Transform

The Continuous Wavelet Transform (CWT) was developed to overcome the time-frequency resolution problem described above and provides a method for determining information regarding the time-frequency representation of a signal. Wavelets analyse according to scale by the use of the superposition of functions at different scales. Wavelet algorithms processes data at different *scales* or *resolutions* to examine a signals different features [Wei94, CK96].

It should be noted, however, that this time-frequency resolution problem is a result of the Hiesenberg uncertainty principle and exists regardless of the transform used. In quantum mechanics the Hiesenberg uncertainty principle tells us that an elementary particle *does not simultaneously have* a precise position and a precise momentum. In time-frequency decompositions the effect of the Hiesenberg uncertainty principle is that a signal can not be concentrated simultaneously in time and frequency, that is, it does not have a precise location in time and a precise frequency. The product $\Delta t \Delta f$, approximately $1/4\pi$, sets a lower limit to the size of the window that can be used [Str94].

Figure 4.7 shows an idealised time-frequency plane decomposed with windowed Fourier analysis. Note, that the width of the boxes remains fixed for each decomposition: the same for high frequencies and low frequencies. Figure 4.8 shows an idealised time-frequency plane decomposed with wavelets. The wide windows, used to look at low frequencies, are precise about frequency and vague about time. The narrow windows, used to look at high frequencies, are precise

about time and vague about frequency. Note, that the boxes in Figure 4.8 all have the same (non-zero) area and represent an equal area of the time-frequency plane. The fact that these boxes have a non-zero area implies that the value of a particular point in the time-frequency plane cannot be known. All the points in the time-frequency plane that fall into a box are represented by one value of the wavelet transform [Polb].

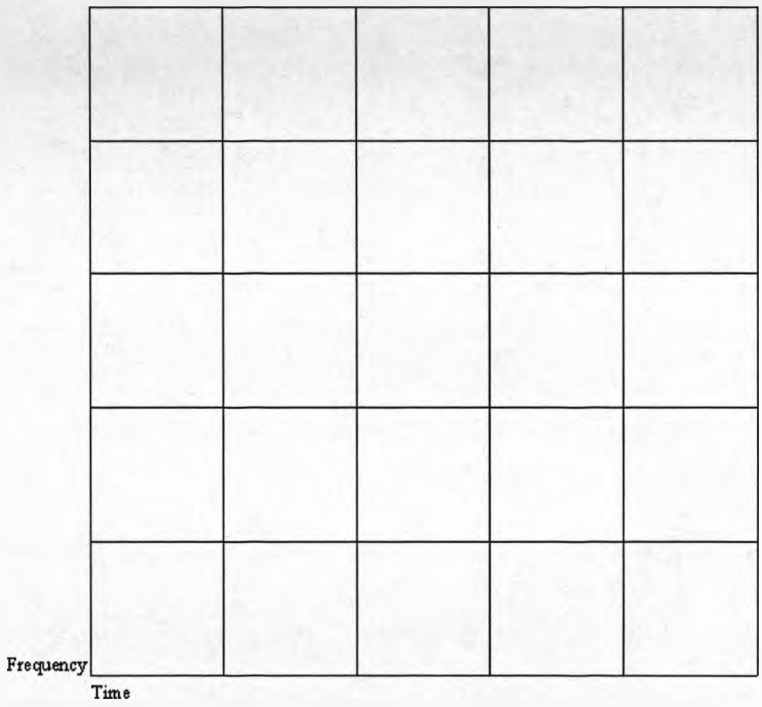


Figure 4.7: An idealised time-frequency plane decomposed with windowed Fourier analysis

Wavelets do not overcome the limit imposed by the Hiesenberg uncertainty principle, but adapt automatically to a signals components. At low frequencies a wider wavelet is used and at high frequencies a narrow wavelet is used. In effect the uncertainty principle imposes a compromise: knowledge gained

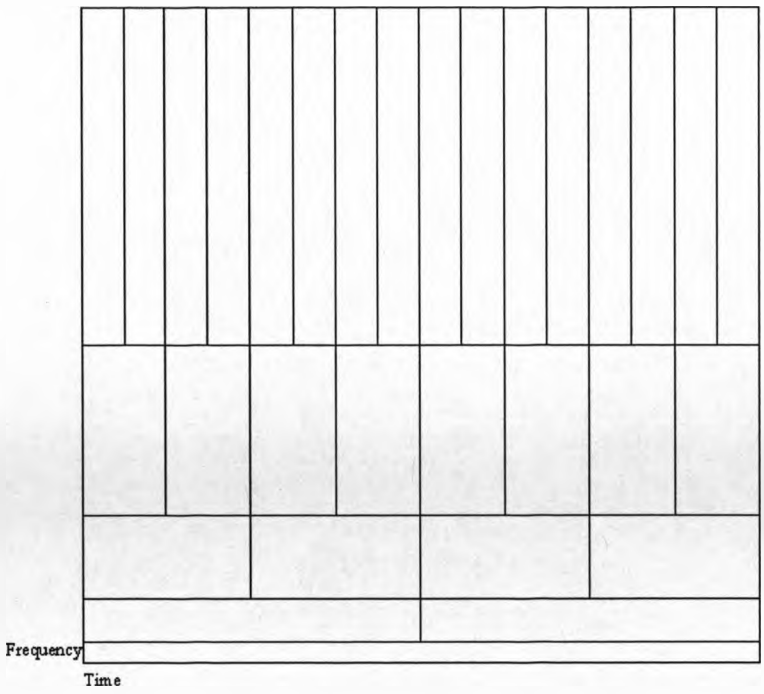


Figure 4.8: An idealised time-frequency plane decomposed with wavelets

about time is paid for in frequency and vice versa [BH96, Chu92]. The wavelet transform uses multiresolution analysis to give good time resolution and poor frequency resolution at high frequencies, and good frequency resolution and poor time resolution at low frequencies [Polc].

Wavelet analysis is performed by multiplying the signal with a function, that is the wavelet, and the transform is then computed separately for different segments of the time domain signal. The width of this window is changed as the transform is computed for every single spectral component [Dau92]. The continuous wavelet transform, $\Psi_x^{\psi}$, is defined as:

$$\Psi_x^{\psi}(\tau, s) = \frac{1}{\sqrt{|s|}} \int x(t)\psi^* \left( \frac{t - \tau}{s} \right) dt. \tag{4.3.1}$$

The transformed signal is a function of the translation, $\tau$, and scale, $s$, param-

eters. $\psi(t)$ is the transforming function, that is, the mother wavelet.

The continuous wavelet transform is a reversible transform and the inverse wavelet transform is defined as:

$$x(t) = \frac{1}{c_\psi^2} \int_s \int_\tau \Psi_x^\psi(\tau, s) \frac{1}{s^2} \psi \left( \frac{t-\tau}{s} \right) d\tau ds. \tag{4.3.2}$$

However, reconstruction of the original signal using equation 4.3.2 is only possible if the admissibility condition is met. Equation 4.3.3 defines the admissibility constant:

$$c_\psi = \left( 2\pi \int_{-\infty}^{\infty} \frac{|\hat{\psi}(\xi)|^2}{|\xi|} d\xi \right)^{\frac{1}{2}} < \infty, \tag{4.3.3}$$

where $\hat{\psi}(\xi)$ is the Fourier transform of $\psi(t)$. Equation 4.3.3 implies that $\hat{\psi}(0) = 0$, which is:

$$\int \psi(t) dt = 0. \tag{4.3.4}$$

Equation 4.3.4 requires that the wavelet function integrates to zero, that is, the wavelet must be oscillatory.

The term translation is related to the location of the windowing function or mother wavelet, as the window is shifted through the signal. This term corresponds to time information in the transform domain. However, there is no frequency parameter. Instead there is a scale parameter, $s$, which is defined as $s = 1/f$. Scaling, as a mathematical operation, either dilates (large scales) or compresses (small scales) a signal [Dau88].

High scales correspond to a non-detailed global view of the signal and low scales correspond to a detailed view. In terms of frequency, low frequencies (high scales) correspond to global information of the signal (that usually spans

the entire signal), whereas high frequencies (low scales) correspond to the detailed information of a hidden pattern in a signal (that usually lasts a relatively short time) [Polc].

### 4.3.1 Computing the Wavelet Transform

Let $x(t)$ be the signal to be analysed using the wavelet transform, equation 4.3.1. A mother wavelet is chosen to serve as a prototype for all windows in the process. There is a wide choice of mother wavelets available, figure 4.9 shows the Daubechies D4 wavelet and Figure 4.10 shows the Haar wavelet [BCD+95].

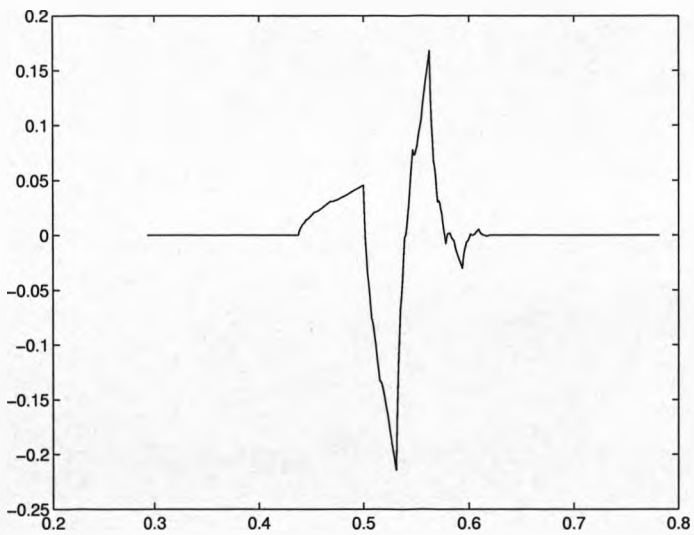


Figure 4.9: The Daubechies D4 wavelet

Once the mother wavelet is chosen the computation starts, with $s = 1$ for example, and continues for increasing values of $s$, that is, the analysis starts from high frequencies and proceeds towards low frequencies. This first value of

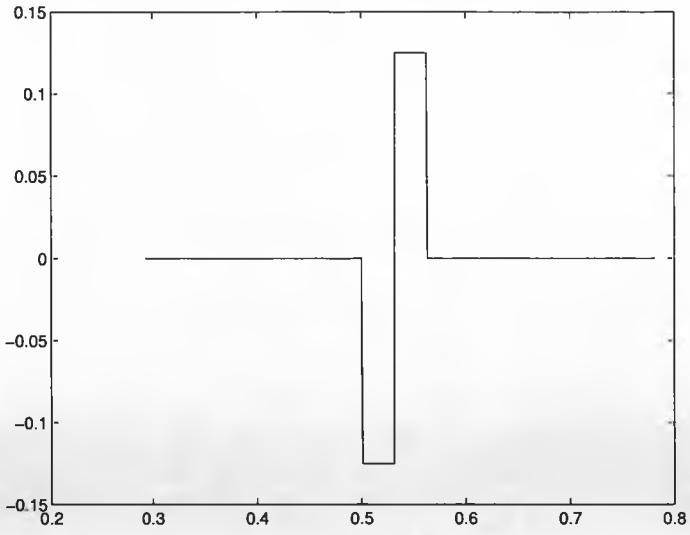


Figure 4.10: The Haar wavelet

$s$ corresponds to the most compressed wavelet. As $s$ is increased, the wavelet dilates.

The wavelet is placed at the beginning of the signal, that is at time $t = 0$. The signal is then multiplied by the wavelet function at scale $s = 1$ and then integrated over all times. The resulting integration is then multiplied by the constant factor $\frac{1}{\sqrt{|s|}}$ to normalise the transformed signal energy. This ensures that the transformed signal will have the same energy at every scale. The value of the CWT has now been calculated for $t = 0$ and $s = 1$, that is, the point $\tau = 0$ and $s = 1$ in the time-scale plane.

The wavelet is then shifted along the signal by $\tau$ to the point $t = \tau$, with $s = 1$ still. The CWT is then computed for the point $t = \tau, s = 1$ in the time-scale plane. This procedure is then repeated until the wavelet reaches the end of the signal. For the scale $s = 1$, one row of points on the time-scale plane

has now been computed.

This procedure is repeated for every value of $s$, with $s$ being incremented by a small amount each time. Every computation for a given value of s fills the corresponding (single) row in the time-scale plane. When this process is completed the wavelet transform of the signal has been calculated [BDG96, Polc].

## 4.4 The Discrete Wavelet Transform

Although the discretised continuous wavelet transform enables the computation of the continuous wavelet transform by computers, it is not a true discrete transform. In fact, the wavelet series is simply a sampled version of the CWT, and the information it provides is highly redundant as far as the reconstruction of the signal is concerned. This redundancy, on the other hand, requires a significant amount of computation time and resources. The discrete wavelet transform (DWT) provides sufficient information both for analysis and synthesis of the original signal, with a significant reduction in the computation time. In addition, the DWT is considerably easier to implement when compared to the CWT. If the scales and positions of the mother wavelet are chosen based on the power of two, that is, *dyadic* scales and positions, then the analysis will be much more efficient without losing any accuracy [SLL$^+$].

The main idea is the same as it is in the CWT. A time-scale representation of a digital signal is obtained using digital filtering techniques. Recall that the CWT is a correlation between a wavelet at different scales and the signal with the scale (or the frequency) being used as a measure of similarity. The continuous wavelet transform was computed by changing the scale of the analysis window, shifting the window in time, multiplying by the signal, and integrating over all times. In the discrete case, filters of different cutoff frequencies are used to analyze the signal at different scales. The signal is passed through a series of highpass filters to analyze the high frequencies, and it is passed through a

series of lowpass filters to analyze the low frequencies [BH96].

The resolution of the signal, which is a measure of the amount of detail information in the signal, is changed by the filtering operations, and the scale is changed by upsampling and downsampling (subsampling) operations. Subsampling a signal corresponds to reducing the sampling rate, or removing some of the samples of the signal. For example, subsampling by two refers to dropping every other sample of the signal. Subsampling by a factor n reduces the number of samples in the signal n times [Str89].

Upsampling a signal corresponds to increasing the sampling rate of a signal by adding new samples to the signal. For example, upsampling by two refers to adding a new sample, usually a zero or an interpolated value, between every two samples of the signal. Upsampling a signal by a factor of n increases the number of samples in the signal by a factor of n.

For many signals, the low-frequency content is the most important part. It is what gives the signal its identity. The high-frequency content, on the other hand, imparts flavour or nuance. Consider, for example, the human voice. If the high-frequency components are removed, the voice sounds different, but it is still possible to tell what is being said. However, remove enough of the low-frequency components, and the voice becomes incoherent. It is for this reason that wavelet analysis often uses the terms *approximations* and *details.*

The approximations are the high-scale, low-frequency components of the signal. The details are the low-scale, high-frequency components. Figure 4.11 shows the filtering process at its most basic level.

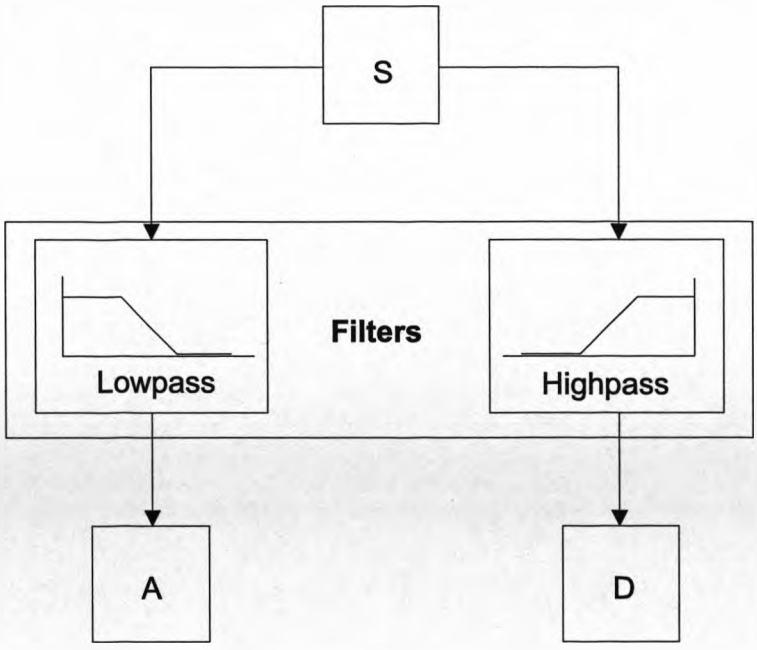


Figure 4.11: Signal S passes through two complimentary filters and emerges as two signals

If a signal is denoted $x[n]$, where $n$ is an integer, then the procedure begins by passing this signal through a half band digital lowpass filter with impulse response h[n]. Filtering a signal corresponds to the mathematical operation of convolution of the signal with the impulse response of the filter. The convolution operation in discrete time is defined as:

$$x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k] \tag{4.4.1}$$

A half band lowpass filter removes all frequencies that are above half of the highest frequency in the signal. For example, if a signal has a maximum of 1000Hz component, then half band lowpass filtering removes all the frequencies above 500Hz.

In discrete signals, frequency is expressed in terms of radians. Accordingly, the sampling frequency of the signal is equal to $2\pi$ radians in terms of radial frequency. Therefore, the highest frequency component that exists in a signal will be $\pi$ radians, if the signal is sampled at Nyquists rate (which is twice the maximum frequency that exists in the signal); that is, the Nyquists rate corresponds to $\pi$ rad/s in the discrete frequency domain [BH96].

After passing the signal through a half band lowpass filter, half of the samples can be eliminated according to the Nyquists rule, since the signal now has a highest frequency of $\pi/2$ radians instead of $\pi$ radians. Simply discarding every other sample will subsample the signal by two and the signal will then have half the number of points. The scale of the signal is now doubled. Note that the lowpass filtering removes the high frequency information, but leaves the scale unchanged. Only the subsampling process changes the scale. Resolution, on the other hand, is related to the amount of information in the signal, and therefore, it is affected by the filtering operations. Half band lowpass filtering removes half of the frequencies, which can be interpreted as losing half of the information. Therefore, the resolution is halved after the filtering operation. Note, however, that the subsampling operation after filtering does not affect the resolution, since removing half of the spectral components from the signal makes half the number of samples redundant anyway. Half the samples can be discarded without any loss of information. In summary, the lowpass filtering halves the resolution, but leaves the scale unchanged. The signal is then subsampled by two since half of the samples are redundant. This doubles the

scale. This procedure can mathematically be expressed as:

$$y[n] = \sum_{k=-\infty}^{\infty} h[k] \cdot x[2n - k] \tag{4.4.2}$$

The DWT analyzes the signal at different frequency bands with different resolutions by decomposing the signal into a coarse approximation and detail information. DWT employs two sets of functions, called scaling functions and wavelet functions, which are associated with lowpass and highpass filters, respectively [Pold]. The decomposition of the signal into different frequency bands is simply obtained by successive highpass and lowpass filtering of the time domain signal. The original signal x[n] is first passed through a halfband highpass filter g[n] and a lowpass filter h[n]. After the filtering, half of the samples can be eliminated according to the Nyquists rule, since the signal now has a highest frequency of $\pi/2$ radians instead of $\pi$. The signal can therefore be subsampled by two, simply by discarding every other sample. This constitutes one level of decomposition and can mathematically be expressed as follows:

$$y_{high}[k] = \sum_{n} x[n] \cdot g[2k - n] \tag{4.4.3}$$

$$y_{low}[k] = \sum_{n} x[n] \cdot h[2k - n] \tag{4.4.4}$$

where $y_{high}[k]$ and $y_{low}[k]$ are the outputs of the highpass and lowpass filters, respectively, after subsampling by two.

This decomposition halves the time resolution since only half the number

of samples now characterizes the entire signal. However, this operation doubles the frequency resolution, since the frequency band of the signal now spans only half the previous frequency band, effectively reducing the uncertainty in the frequency band by half. The above procedure, which is also known as subband coding, can be repeated for further decomposition. At every level, the filtering and subsampling will result in half the number of samples (and hence half the time resolution) and half the frequency band spanned (and hence double the frequency resolution). Figure 4.12 illustrates this process, where $x[n]$ is the original signal to be decomposed and $h[n]$ and $g[n]$ are lowpass and highpass filters respectively. The bandwidth of the signal at each level is marked on the figure as $f$.

As an example, suppose that the original signal $x[n]$ has 512 sample points, spanning a frequency band of zero to $\pi$ rad/s. At the first decomposition level, the signal is passed through the highpass and lowpass filters, followed by subsampling by two. The output of the highpass filter has 256 points (hence half the time resolution), but it only spans the frequencies $\pi/2$ to $\pi$ rad/s (hence double the frequency resolution). These 256 samples constitute the first level of DWT coefficients. The output of the lowpass filter also has 256 samples, but it spans the other half of the frequency band, frequencies from 0 to $\pi/2$ rad/s. This signal is then passed through further lowpass and highpass filters for further decomposition. The output of the second lowpass filter followed by subsampling has 128 samples spanning a frequency band of 0 to $\pi/4$ rad/s, and the output of the second highpass filter followed by subsampling has 128

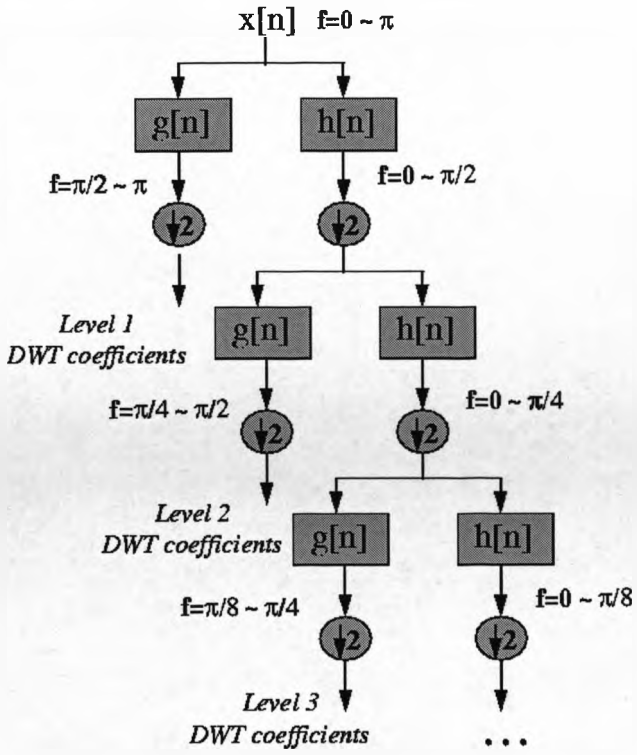


Figure 4.12: The Subband Coding algorithm

samples spanning a frequency band of $\pi/4$ to $\pi/2$ rad/s. The second highpass filtered signal constitutes the second level of DWT coefficients. This signal has half the time resolution, but twice the frequency resolution of the first level signal. In other words, time resolution has decreased by a factor of four, and frequency resolution has increased by a factor of four compared to the original signal. The lowpass filter output is then filtered once again for further decomposition. This process continues until two samples are left. For this specific example there would be eight levels of decomposition, each having half the number of samples of the previous level. The DWT of the original signal

is then obtained by concatenating all coefficients starting from the last level of decomposition (remaining two samples, in this case). The DWT will then have the same number of coefficients as the original signal.

The frequencies that are most prominent in the original signal will appear as high amplitudes in that region of the DWT signal that includes those particular frequencies. The difference of this transform from the Fourier transform is that the time localization of these frequencies will not be lost. However, the time localization will have a resolution that depends on at which level they appear. If the main information of the signal lies in the high frequencies, as happens most often, the time localization of these frequencies will be more precise, since they are characterized by a greater number of samples. If the main information lies only at very low frequencies, the time localization will not be very precise, since few samples are used to express signal at these frequencies. This procedure in effect offers a good time resolution at high frequencies, and good frequency resolution at low frequencies. Most practical signals encountered are of this type. The frequency bands that are not very prominent in the original signal will have very low amplitudes, and that part of the DWT signal can be discarded without any major loss of information, allowing data reduction [BH96].

One important property of the discrete wavelet transform is the relationship between the impulse responses of the highpass and lowpass filters. The highpass and lowpass filters are not independent of each other and are related by:

$$g[L-1-n] = (-1)^n \cdot h[n] \tag{4.4.5}$$

where $g[n]$ is the highpass and $h[n]$ the lowpass filter and $L$ is the filter length. Note that the two filters are odd index alternated reversed versions of each other. Lowpass to highpass conversion is provided by the $(-1)^n$ term. Filters satisfying this condition are commonly used in signal processing and are known as *Quadrature Mirror Filters* (QMF). The two filtering and subsampling operations can be expressed by:

$$y_{high}[k] = \sum_n x[n] \cdot g[-n+2k] \tag{4.4.6}$$

$$y_{low}[k] = \sum_n x[n] \cdot h[-n+2k] \tag{4.4.7}$$

## 4.5 Analysis of the Oil Mist data

### 4.5.1 Raw Data

Data has been provided by Shell UK which consists of limited real world infra-red scatter data. This data comes from three separate sensors arranged spatially around a point to detect oil mist. The data falls into four alarm categories; High, Low, Grey and None. The physical characteristics of this data is unknown. Figures 4.13, 4.14, 4.15 and 4.16 show typical plots of high, low, grey and none alarm data samples respectively.

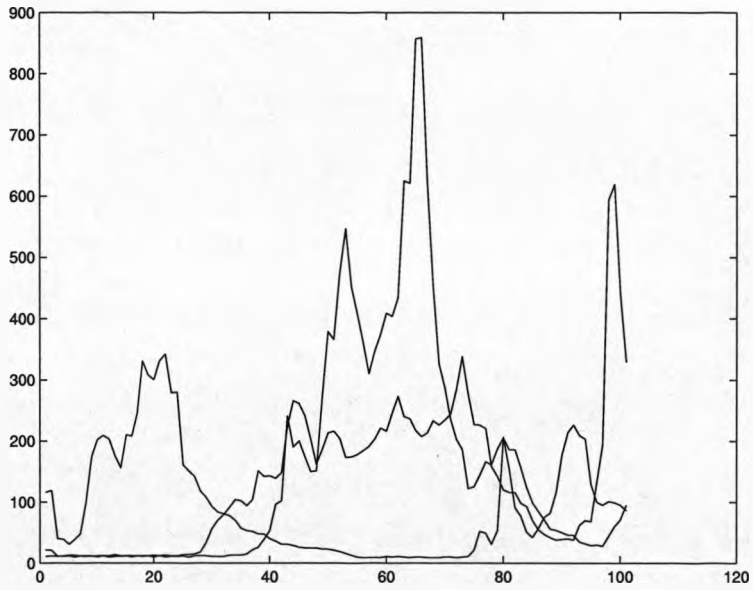


Figure 4.13: High alarm data

Figure 4.17 gives a scatter plot of High1 beams A and B and clearly shows that the data does not fall into the sort of curvilinear relationship described above in Section 2.3.6 and thus suggests that correlation coefficient will not

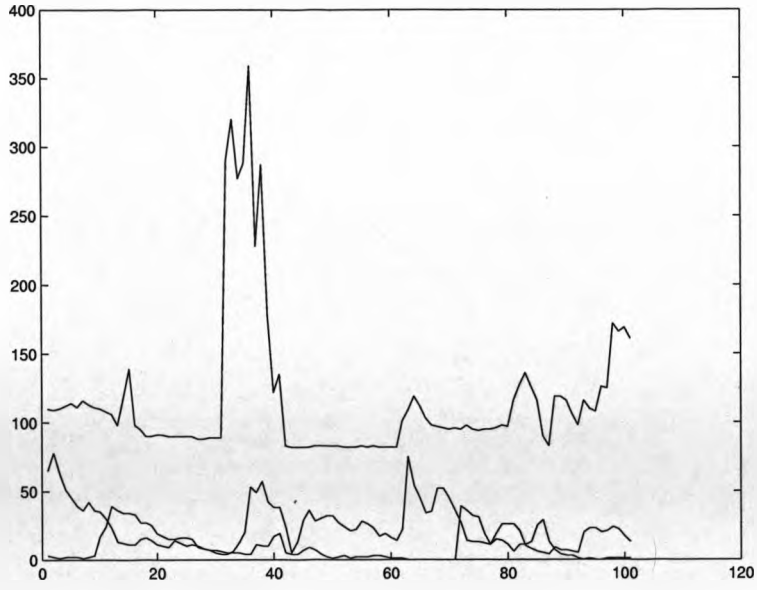


Figure 4.14: Low alarm data

reveal any useful information about this particular set of data.

Figures 4.18, 4.19, 4.20, 4.21 and 4.22 show further scatter plots of the raw data, in both the High and Low alarm categories. These figures show raw data that also clearly does not fit the curvilinear relationship that is generally required for the successful use of the correlation coefficients, in revealing any useful information about the raw data.

A wavelet analysis was then performed on the data using a range of mother wavelets [FTK96, BBPB96, KP]. The Daubechies wavelet of order 5 was eventually selected as the wavelet that appeared to reveal most consistently, that is, across the widest range of samples of the data, the most interesting features of the data. Specifically it was the level 3 detail coefficients of the wavelet transform, using the Daubechies 5 mother wavelet, that was chosen as the

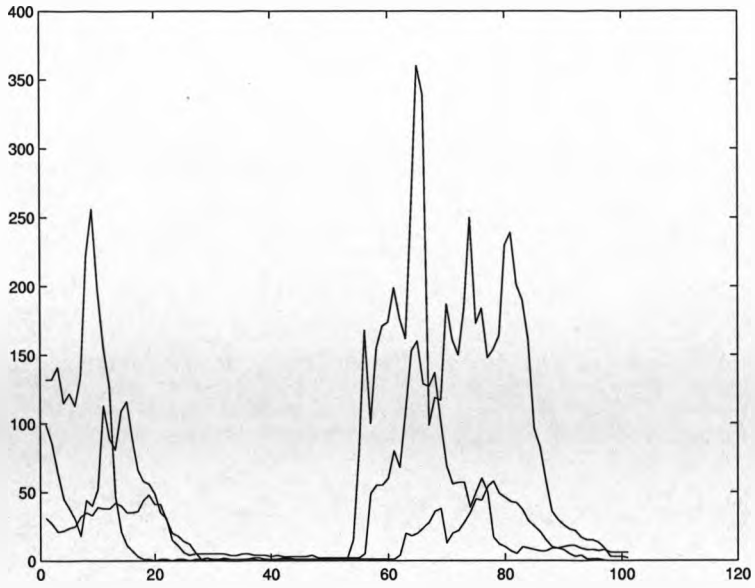


Figure 4.15: Grey alarm data

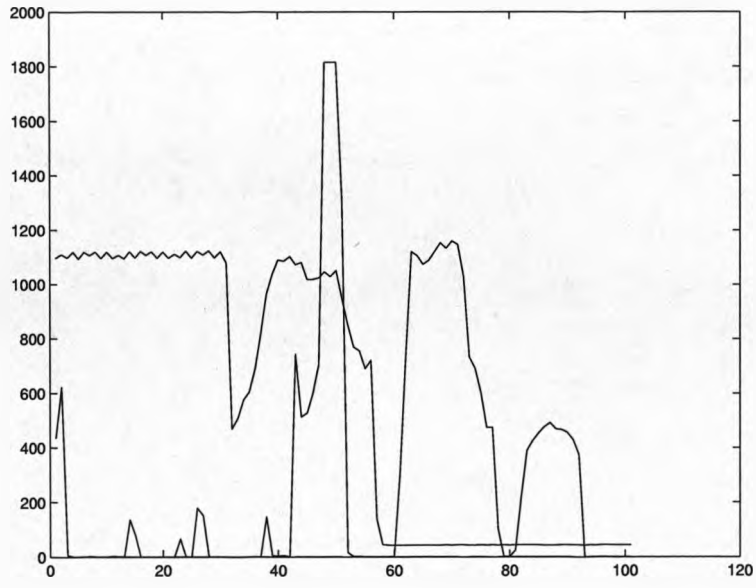


Figure 4.16: None alarm data

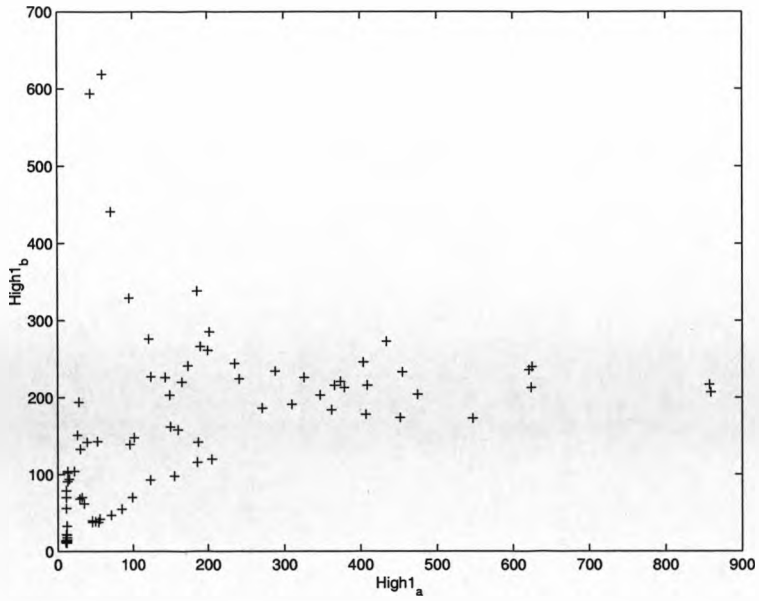


Figure 4.17: Scatter plot of High1 beams A and B

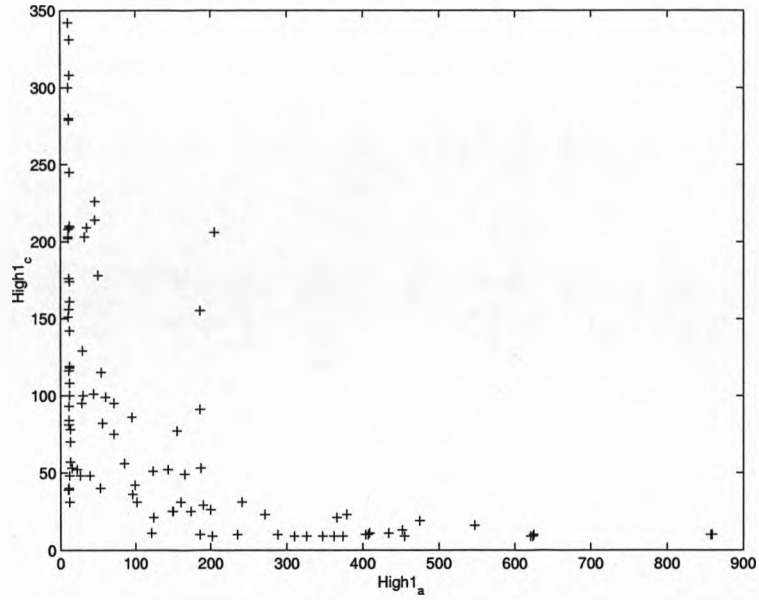


Figure 4.18: Scatter plot of High1 beams A and C

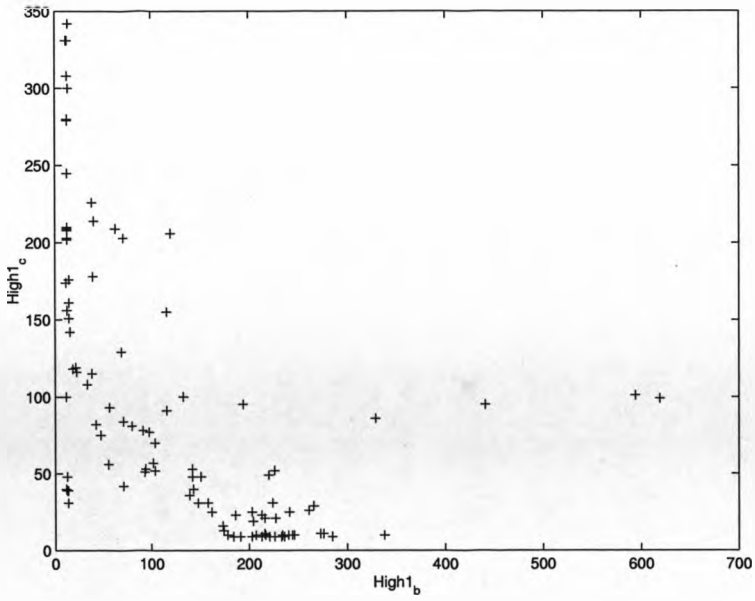


Figure 4.19: Scatter plot of High1 beams B and C

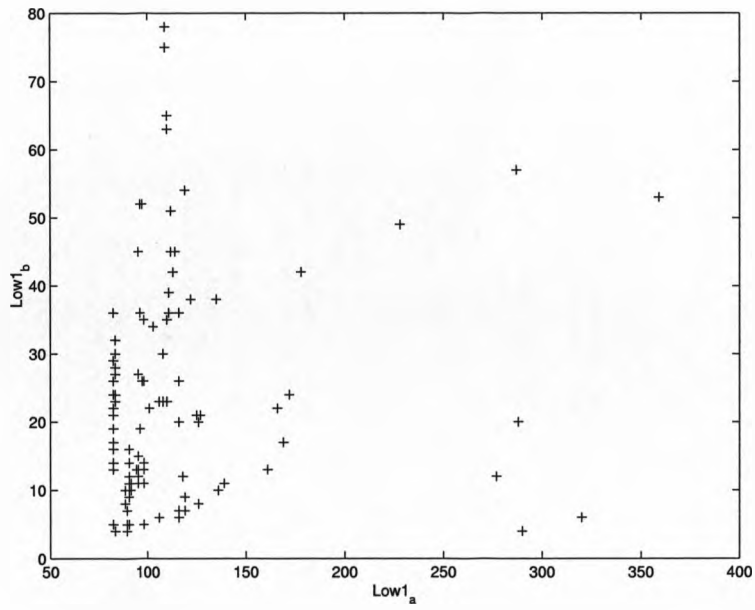


Figure 4.20: Scatter plot of Low1 beams A and B

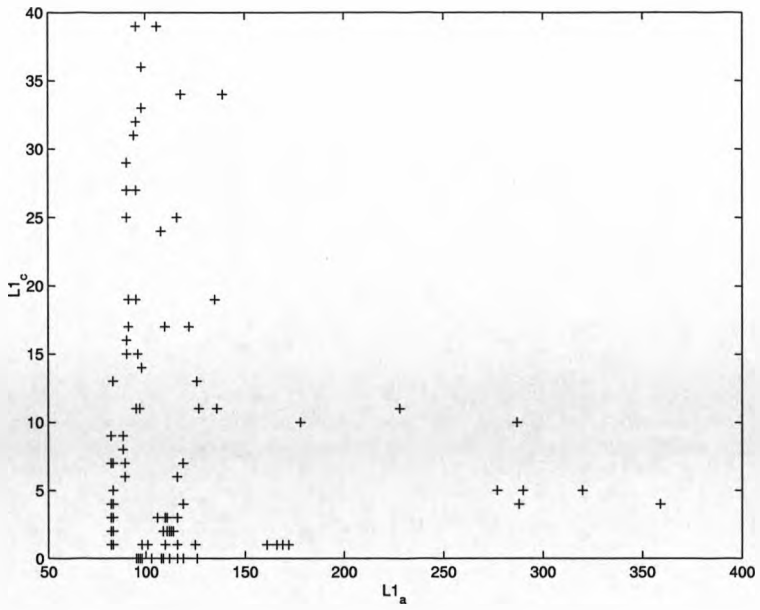


Figure 4.21: Scatter plot of Low1 beams A and C

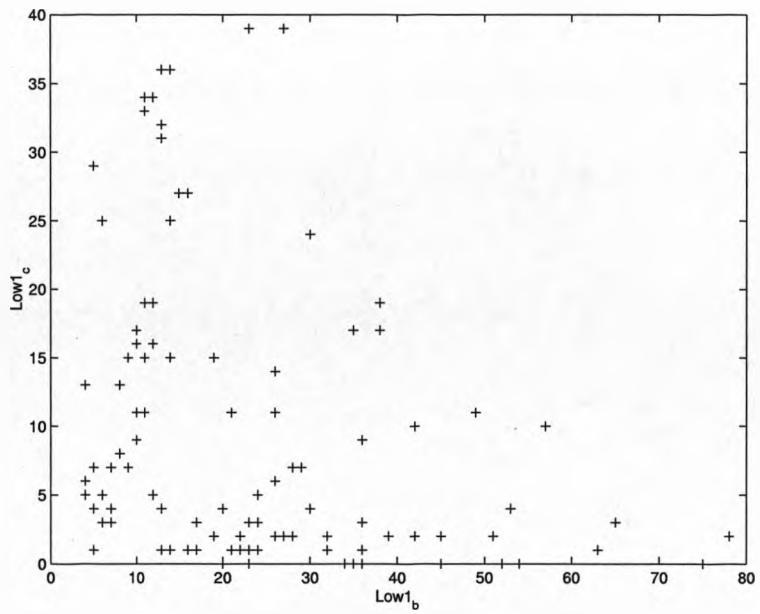


Figure 4.22: Scatter plot of Low1 beams B and C

most suitable for training a neural network. Figures 4.23 and 4.24 show the wavelet coefficients of two samples of the data extracted using the Daubechies 5 wavelet.

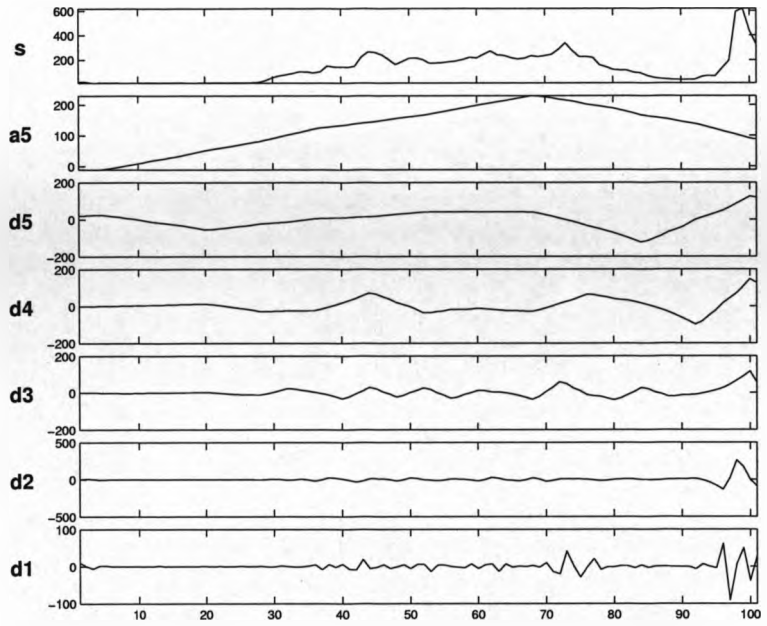


Figure 4.23: Wavelet coefficients of a typical sample of High alarm data using Daubechies 5 mother wavelet

Figures 4.25, 4.26, 4.27, 4.28, 4.29 and 4.30 show scatter plots of the selected wavelet coefficients of the data in Figures 4.18, 4.19, 4.20, 4.21 and 4.22. The scatter plots in Figures 4.25, 4.26, 4.27, 4.28, 4.29 and 4.30 again indicate that use of the correlation coefficients of the wavelet transform is unlikely to yield any useful information.

Nevertheless, the correlation coefficients were extracted from the wavelet coefficients and used to train backpropagation neural networks. In an attempt to ease the pattern separation problem only two of the alarm conditions, high

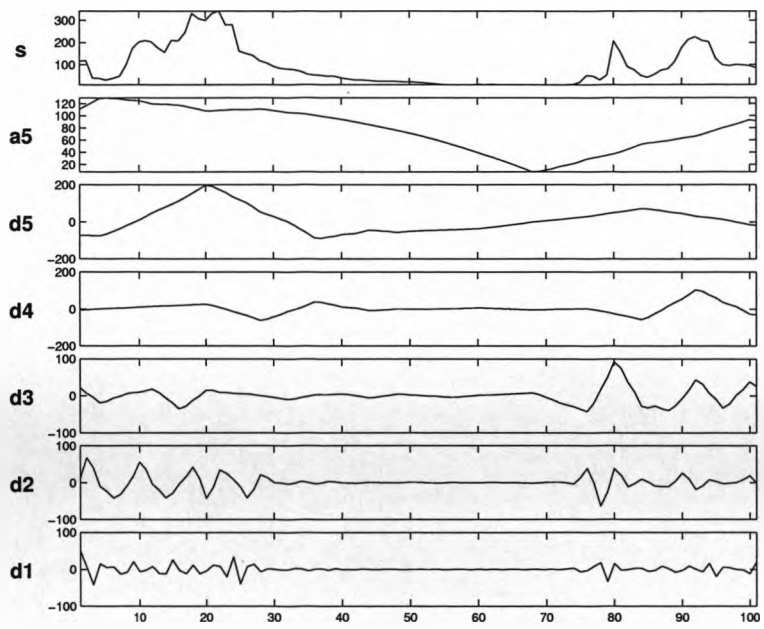


Figure 4.24: Wavelet coefficients of a typical sample of Low alarm data using Daubechies 5 mother wavelet

and low, were selected for training the neural network. The processed data was split into two groups for neural network training and testing purposes. The training data was then used to train a whole range of feedforward neural networks, with differing numbers of neurons in the hidden and output layers.

Figure 4.31 shows a plot of the training errors for a backpropagation network with three hidden neurons. The training of this network was successful, with all training patterns being classified correctly by the trained network. However, the generalisation was poor when the network was tested on previously unseen test data, the prediction results showed only 45% of Low alarm test data and 49% of High alarm test data being classified correctly. Figure 4.32 shows a plot of the training errors for a backpropagation network with six hid-

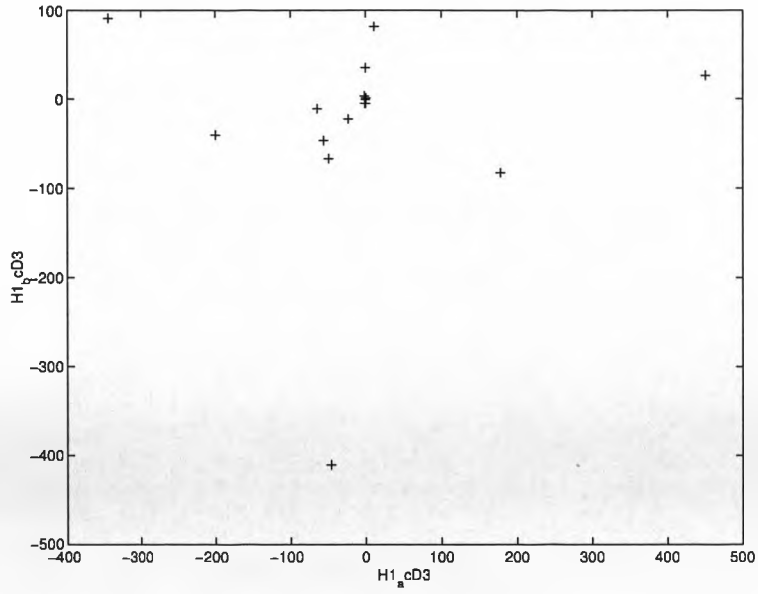


Figure 4.25: Scatter plot of Wavelet coefficients of High1 beams A and B

den neurons. Again training was successful but, generalisation was poor with 78% of Low test cases and 45% of High test cases being classified correctly. Figure 4.33 shows a plot of the training errors for a backpropagation network with nine hidden neurons. The network trained successfully once again but, again the generalisation of the network was poor, with the classification of the test data showing no improvement with 66% and 49% of Low and High alarm data respectively being classified successfully. Addition of further neurons resulted in no further increase of the prediction accuracy of the neural networks.

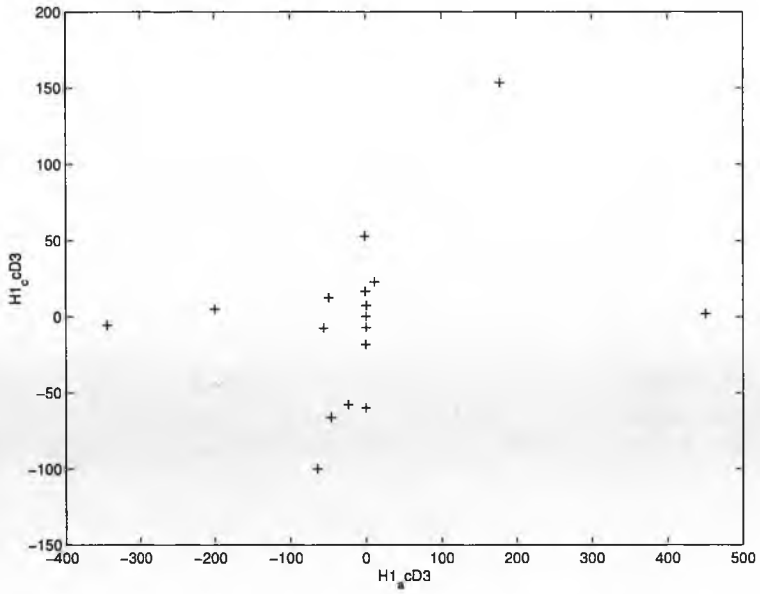


Figure 4.26: Scatter plot of Wavelet coefficients of High1 beams A and C

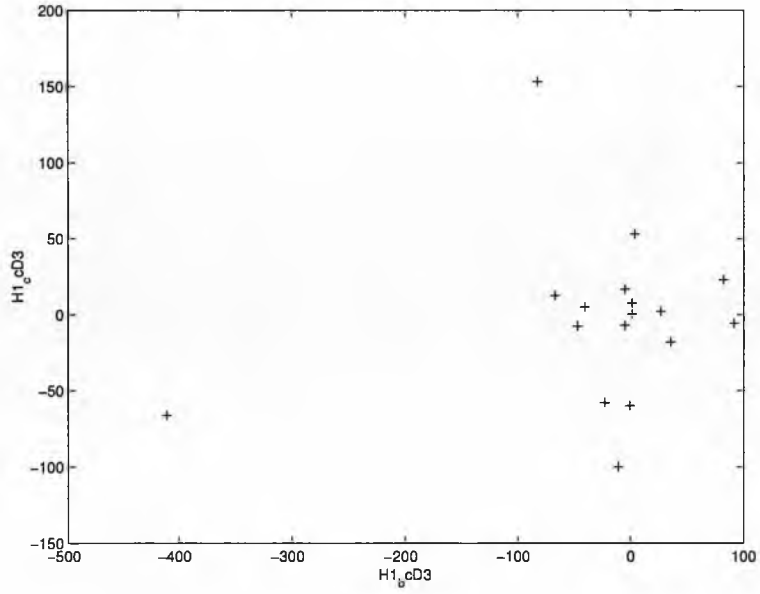


Figure 4.27: Scatter plot of Wavelet coefficients of High1 beams B and C

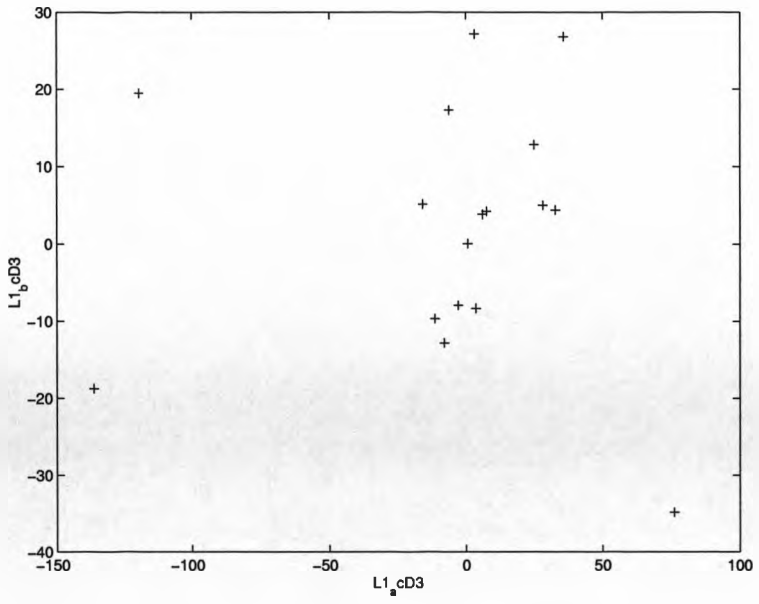


Figure 4.28: Scatter plot of Wavelet coefficients of Low1 beams A and B

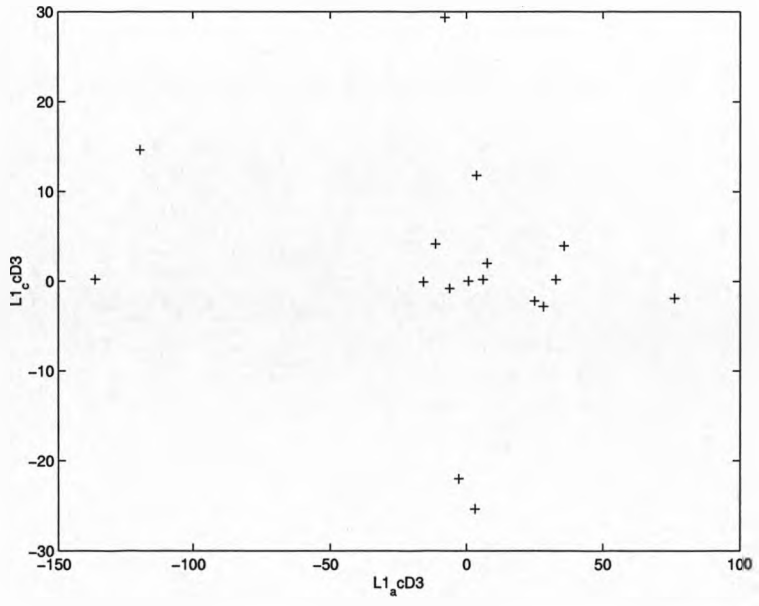


Figure 4.29: Scatter plot of Wavelet coefficients of Low1 beams A and C

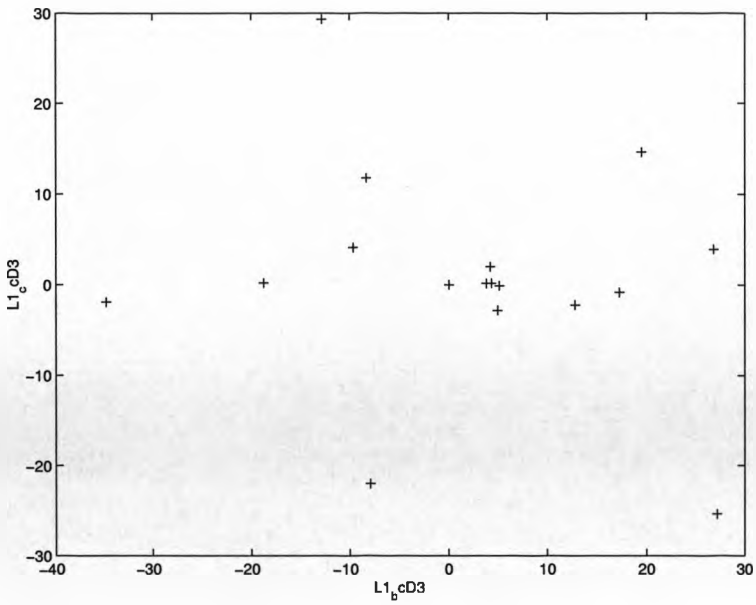


Figure 4.30: Scatter plot of Wavelet coefficients of Low1 beams B and C

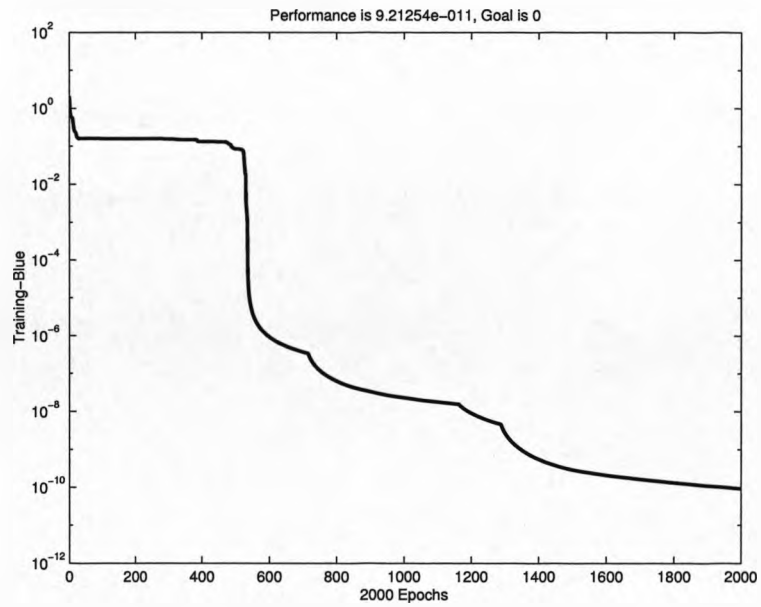


Figure 4.31: Training errors for backpropagation neural network with 3 hidden neurons

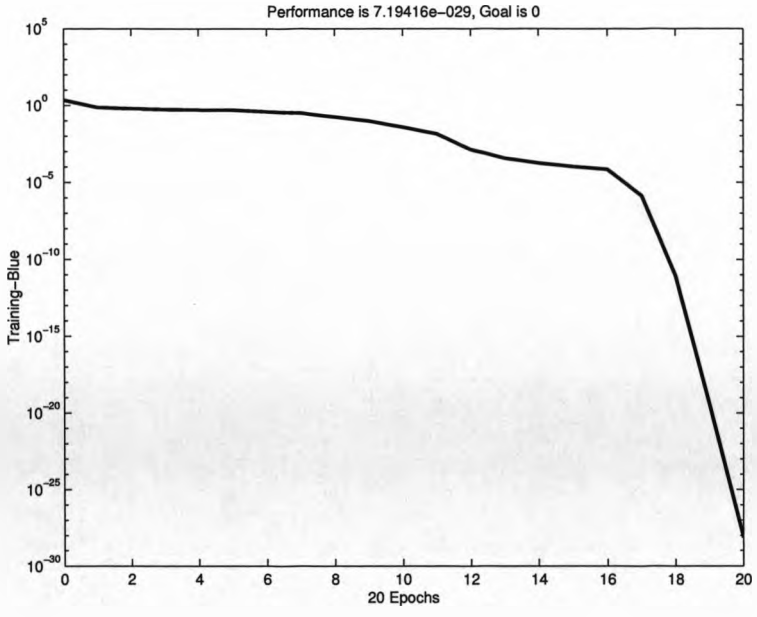


Figure 4.32: Training errors for backpropagation neural network with 6 hidden neurons

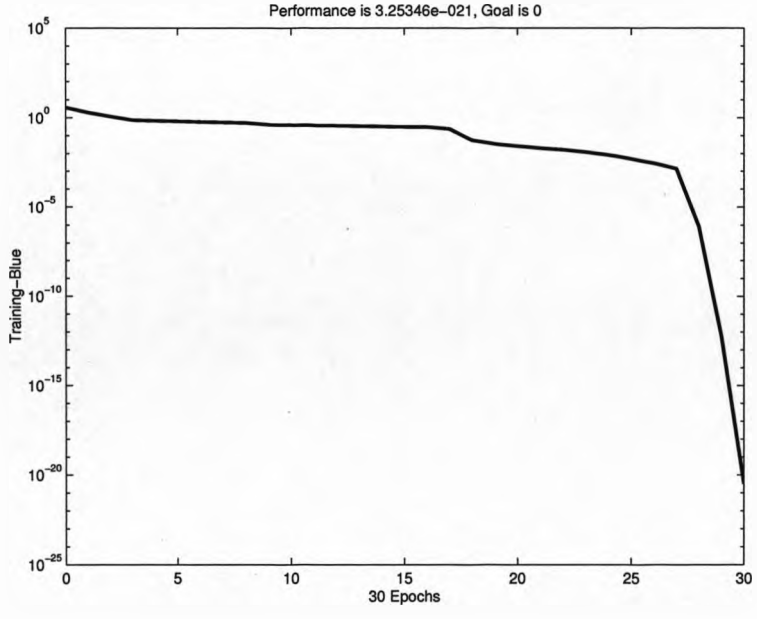


Figure 4.33: Training errors for backpropagation neural network with 9 hidden neurons

### 4.5.2 Normalised Data

The next stage involved normalising the data before using other techniques for extracting the key features of the data. Normalisation of the data involves processing the data so that it has a mean of zero and a standard deviation of 1. The formula for normalising the data is given by:

$$x_{norm} = \frac{x - \overline{x}}{\sigma_x} \tag{4.5.1}$$

where $\overline{x}$ is the mean of $x$ and $\sigma_x$ is the standard deviation of $x$.

Figures 4.34, 4.35, 4.36 and 4.37 show plots of the normalised alarm data for the High, Low, Grey and None alarm categories respectively.

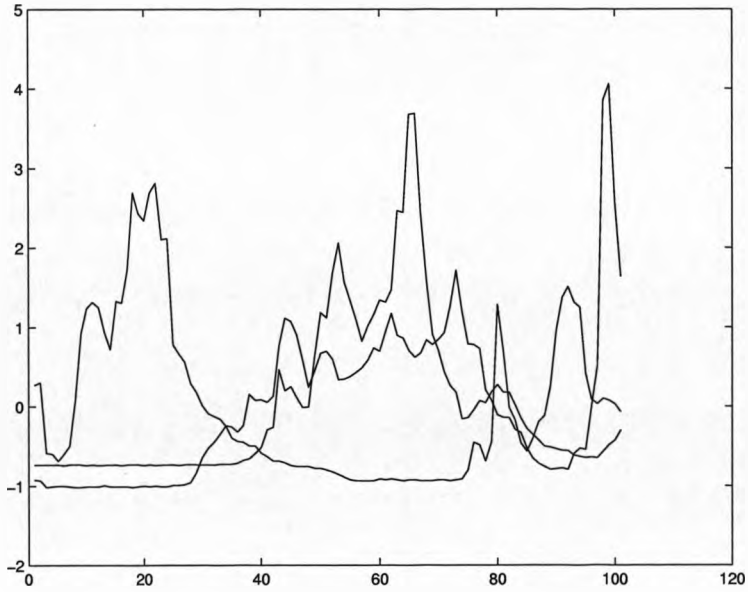


Figure 4.34: Normalised High alarm data

Once the data had been normalised, the wavelet transform was applied to the normalised data. The correlation coefficients of the wavelet coefficients were

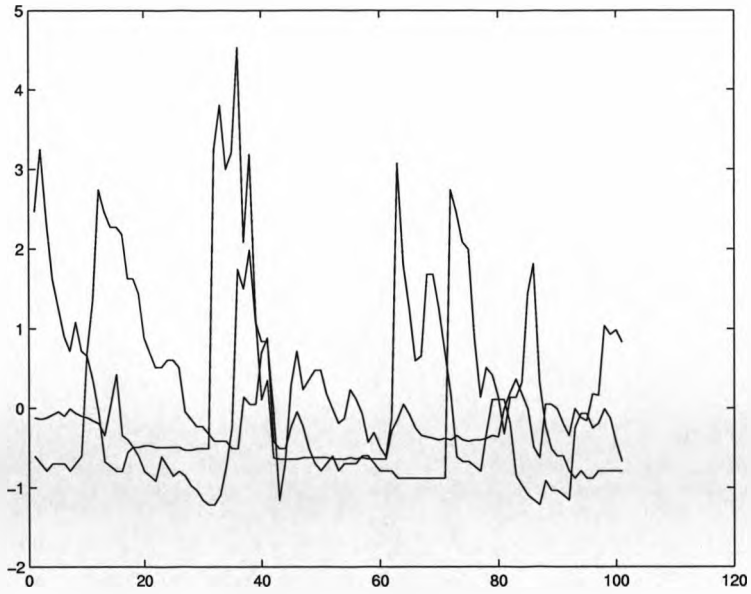


Figure 4.35: Normalised Low alarm data

then calculated as was the variance of the wavelet coefficients. The correlation coefficients were then concatenated with the variances to create a set of input vectors, of dimensionality six, for the feedforward neural network. Only two of the alarm conditions were chosen for training and testing purposes, to increase the probability of the neural networks being able to successfully classify the data. The two alarm conditions chosen were High and None, as this allowed for a greater number of training vectors to be used in the training of the neural networks.

Figures 4.38, 4.39, 4.40, 4.41, 4.42 and 4.43 show plots of the training errors for backpropagation neural networks containing, 3, 6, 9, 12, 18 and 24 hidden neurons respectively.

All the neural networks, with the exception of the network with 3 hidden

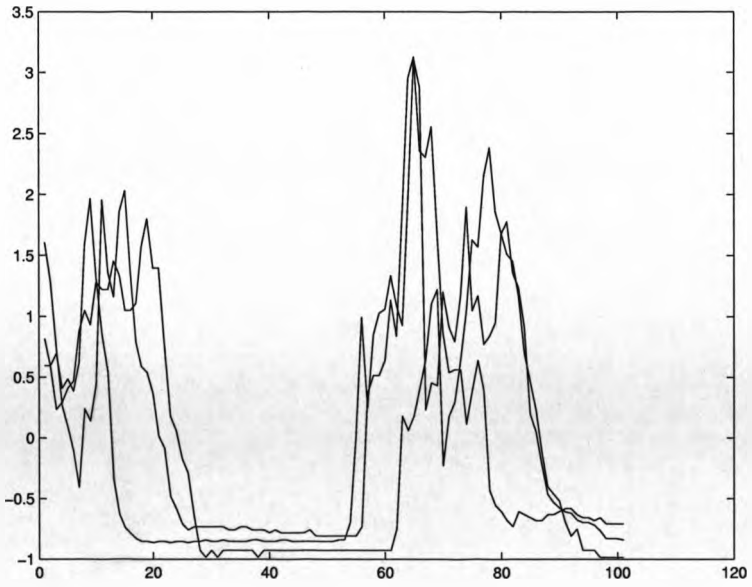


Figure 4.36: Normalised Grey alarm data

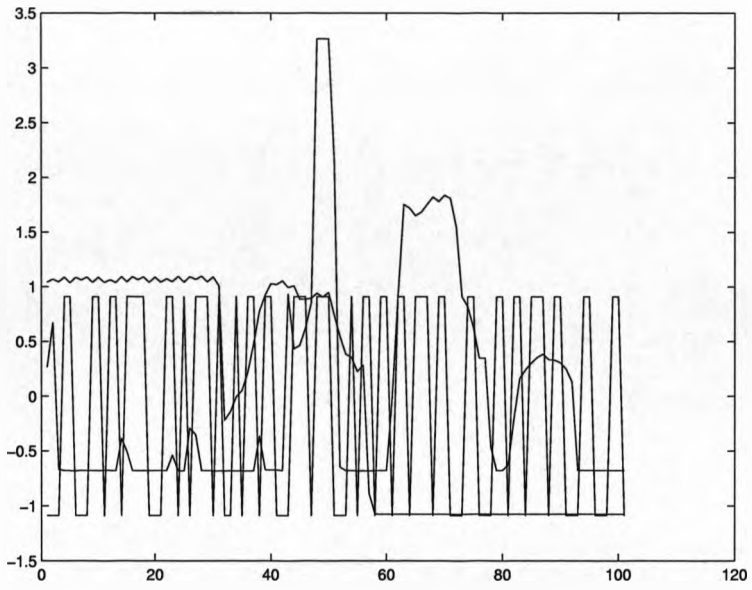


Figure 4.37: Normalised None alarm data

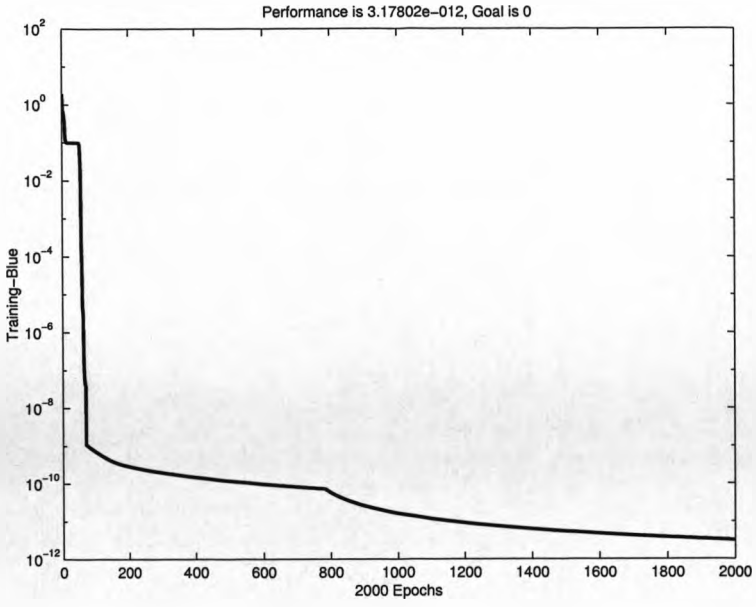


Figure 4.38: Training errors for backpropagation neural network with 3 hidden neurons

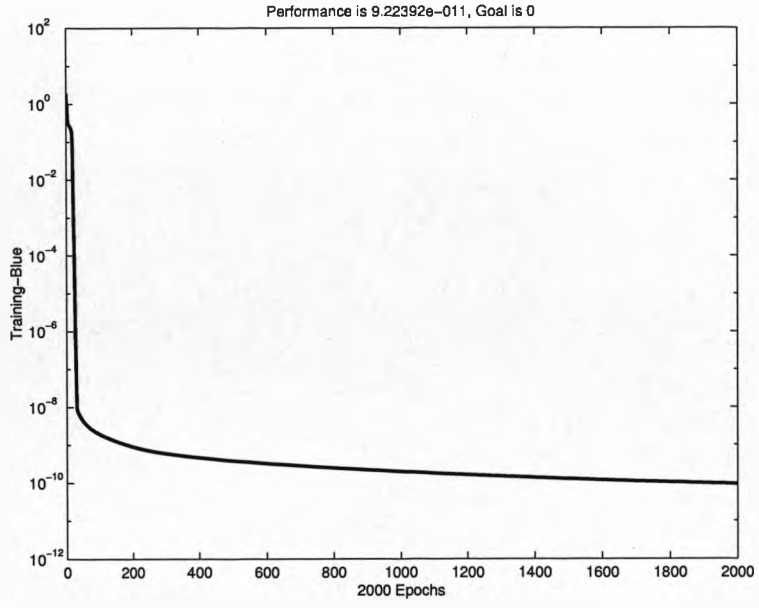


Figure 4.39: Training errors for backpropagation neural network with 6 hidden neurons

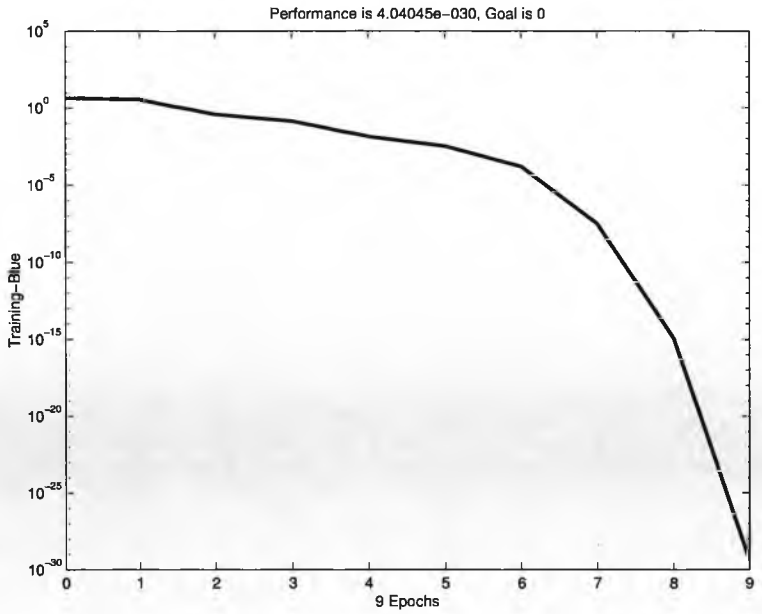


Figure 4.40: Training errors for backpropagation neural network with 9 hidden neurons

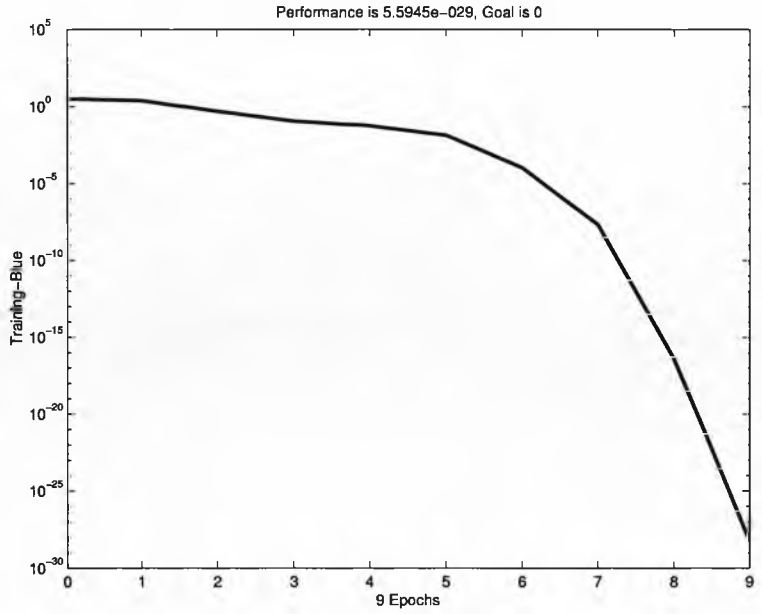


Figure 4.41: Training errors for backpropagation neural network with 12 hidden neurons

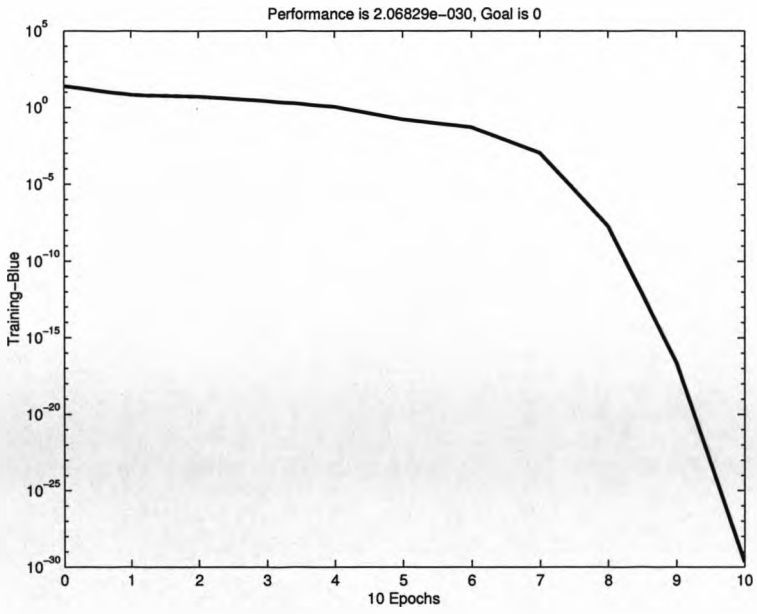


Figure 4.42: Training errors for backpropagation neural network with 18 hidden neurons

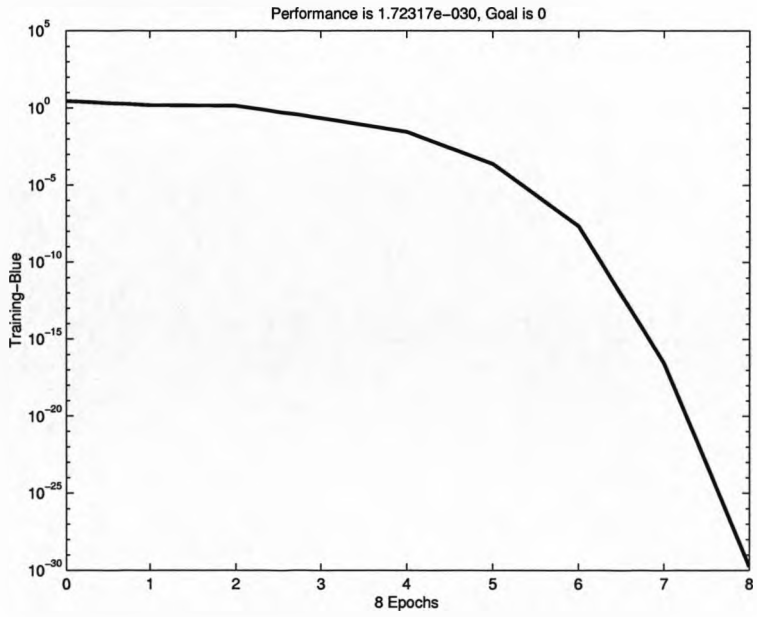


Figure 4.43: Training errors for backpropagation neural network with 24 hidden neurons

neurons, trained correctly. That is, once trained, the networks were then presented with the training data again and successfully classified all the training samples of the High and None alarm data. The neural network with 3 hidden neurons achieved success rates of 90% in the testing of the None alarm training data and 80% in the testing of the High alarm training data.

Table 4.1 shows the percentage accuracy of the different neural networks predictions in classifying data, when tested on previously unseen test data. As can be seen from table 4.1 the prediction results are disappointing. The neural networks trained have all completely failed to generalise when presented with previously unseen test data and have proved incapable of accurate prediction results.

| Number of hidden neurons | High | None |
|---|---|---|
| 3 | 44% | 33% |
| 6 | 51% | 25% |
| 9 | 54% | 25% |
| 12 | 62% | 41% |
| 18 | 58% | 42% |
| 24 | 74% | 58% |

Table 4.1: Neural network prediction accuracy

## 4.6 Conclusion

The research work undertaken in this section was to develop a novel methodology for time series prediction. The aim was to produce an intelligent monitoring and prediction system based on neural networks. As previously mentioned in Section 1.3 *Shell U.K.* wish to develop such a system for unmanned remote installations, with the aim of developing an alarm system that can anticipate problems due to oil mist, and have provided the raw data for the development of this work. The scatter data is provided by three infra-red detectors all monitoring the same cell from different angles, and falls into one of four alarm categories: High, Low, Grey and None.

Initially neural networks were trained using the raw infra-red scatter data. While it has been possible to train many multilayer perceptron neural networks, with differing numbers of neurons in the hidden and output layers, that can correctly classify training patterns when they are represented to the trained network, so far it has not been possible to train networks that generalise satisfactorily. That is, none of the networks have been able to generalise correctly when presented with test vectors that have not been used in the training of the network. The neural networks prediction accuracy, achieved when classifying previously unseen test data, has generally been of the order of $40 - 60\%$. This is not an acceptable degree of accuracy for any real world applications.

Further work involved extracting the wavelet coefficients of the data. The mother wavelet selected was the Daubechies wavelet of level 5, as initial obser-

vations suggested this revealed the most interesting features of the data. The correlation coefficients of the wavelet transform were then examined to discover whether or not the patterns showed any likelihood of being separable in the pattern space. Again, a range of backpropagation networks trained correctly but performed poorly when generalising using test patterns.

The next stage involved normalising the data, so that it had a mean of 0 and a standard deviation of 1. The wavelet coefficients, Daubechies level 5, were extracted from the standardised data and concatenated with the variances to form neural network training vectors. To reduce the dimensionality of the problem in the pattern separation, only the High and None alarm conditions were selected for neural network training. Again a number of backpropagation networks were trained. Although most of these networks trained correctly as they were able to successfully classify the training data, the networks did not perform consistently when presented with previously unseen test vectors.

Overall, these studies have found that using the wavelet transform on both the raw infra-red scatter data and the normalised data produced neural networks that could be trained successfully. However, even with the reduced dimensionality of the data as a result of the wavelet transform, the neural networks could not successfully classify the test vectors of either the raw data or the normalised data on a consistent basis.

# Chapter 5

# Expert Systems And Evolutionary Computation

## 5.1 Introduction

Rule-based expert systems, or production systems, represent the underlying technology for many of today's artificial intelligence systems. Rules have been used extensively in artificial intelligence because rules are simple to work with and because each rule can be considered independent of the others. This latter fact allows for the incremental construction of expert systems.

The main components of an expert system are a knowledge base and a demonstrator. The knowledge base contains rules, of the form if-then, and facts in a declaratory and modular form, and exists independently of the demonstrator. The demonstrator must be able to interpret the information in the

knowledge base and use it to solve a question within the relevant problem domain.

Expert systems generally include symbolic representation, symbolic inference and heuristic search. Expert systems perform difficult tasks at expert levels of performance by employing domain-specific problem-solving strategies. An expert system is capable of employing self-knowledge to reason about their own inference processes and can provide justifications for conclusions reached.

Evolutionary computation can be an effective method for finding the solution of a problem. Evolutionary computation searches through the space of all the potential solutions to that problem.

Genetic algorithms frequently operate on fixed length character strings, often binary, as the structure undergoing adaptation. Fitness is determined by executing task specific routines and algorithms using an interpretation of the character string as the set of parameters. The two main genetic operators are reproduction and crossover (sexual recombination). Both are performed with a probability based on fitness. The mutation operator is of secondary importance.

Genetic classifier systems are a form of genetics-based machine learning system. Genetic classifiers combine a simple, parallel production system based on string rules, an apportionment of credit algorithm modelled after an information based service economy, and genetic algorithms.

Genetic programming is an offshoot of genetic algorithms in which the computer structures that undergo adaptation are themselves computer programs.

Specialised genetic operators are used which generalise crossover and mutation for the tree structured computer programs undergoing adaptation.

Genetic programming is usually nonlinear and operates on tree structured genetic material. The genetic material that genetic programming operates on can vary in size. Genetic programming creates executable genetic material, that is, the genetic material is executed in order to form the desired function from which the fitness is derived. Syntax preserving crossover is performed by genetic programming, in order to preserve the syntactic correctness of the program which is the genetic material.

Learning expert systems, genetic classifiers, are a cognitive architecture that allow the adaptive modification of a set of if-then rules. The architecture of the classifier system blends important features from the contemporary paradigms of artificial intelligence, connectionism and machine learning, including:

- The power, understandability and convenience of if-then rules from expert systems.
- A connectionist-style allocation of credit that rewards specific rules when the system as a whole takes an external action that produces a reward.
- The creative power and efficient search capability of the conventional genetic algorithm operating on fixed length character strings.

In a learning expert system there is a set of if-then rules. Both the condition part and the action part of each if-then rule consists of a fixed length string. When the condition part of the rule is satisfied by some external environmental

feature the rule is "fired". Rules that are fired, and cause the learning expert system to contribute to some environmental action, then have their fitness increased. Genetic operators are then used on the set of if-then rules to create new rules. The objective of the classifier system is to breed a co-adapted set of if-then rules that successfully work together to solve a problem.

An intelligent monitoring and prediction system could be developed using: signal processing to extract the key features from the raw data, neural networks for pattern classification and a learning expert system for the adaptive adaptation of the systems rule base. Figure 5.1 shows the block diagram of an ideal intelligent monitoring and prediction system.

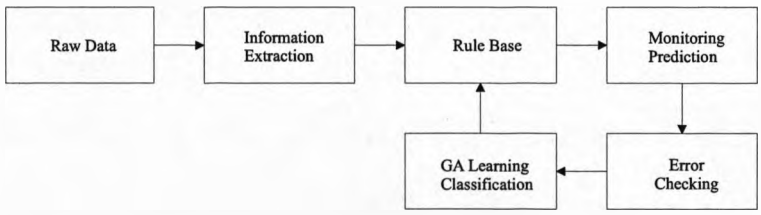


Figure 5.1: Block diagram of an ideal intelligent monitoring and prediction system

## 5.2 Expert Systems

The aim of an *expert system* is to reproduce the behaviour of a human expert, thus performing an intellectual task in a specified field. Expert systems position themselves at the junction of the two approaches to *artificial intelligence* (AI); the representation of information on the one hand and its automatic 'demonstration' on the other.

These form two totally independent systems:

1. A knowledge base.
2. A *demonstrator* of theorems (an *inference engine* or *rule interpreter*).

The knowledge base translates expert knowledge in a given field into a declaratory and modular form. The demonstrator has the task of calling up and using this information in a useful way in order to answer a question or solve a problem. The knowledge base must be readable on its own and must exist independently from the inference engine, but must be capable of being interpreted by it, and this knowledge base must be under the control of the human expert.

An expert system differs from a conventional computer program in two essential ways since at any time an expert system can [Gon86]:

1. Explain its behaviour to the human expert.
2. Receive new pieces of information from the human expert without any new programming being required.

### 5.2.1 Components of an Expert System

Figure 5.2 shows an idealised representation of an expert system. No expert system contains all these components, but one or more components occur in every expert system.

The ideal expert system contains a language processor for problem-oriented communications between the user and the expert system; a "blackboard" for recording intermediate results; a knowledge base comprising facts as well as heuristic planning and problem solving rules; an interpreter that applies these rules; a scheduler to control the order of rule processing; a consistency enforcer that adjusts previous conclusions when new data (or knowledge) alter their bases of support; and a justifier that rationalises and explains the systems behaviour [Kos93].

The user interacts with the expert system in a problem-oriented language, usually some restricted variant of English. The language processor mediates information exchanges between the expert system and the user. Typically the language processor parses and interprets user questions, commands and volunteered information. Conversely, the language processor formats information generated by the system, including answers to questions, explanations and justifications for its behaviour, and requests for information.

The blackboard records intermediate hypotheses and decisions that the expert system manipulates. Every expert system uses some type of intermediate decision representation, but only a few explicitly employ a blackboard. The

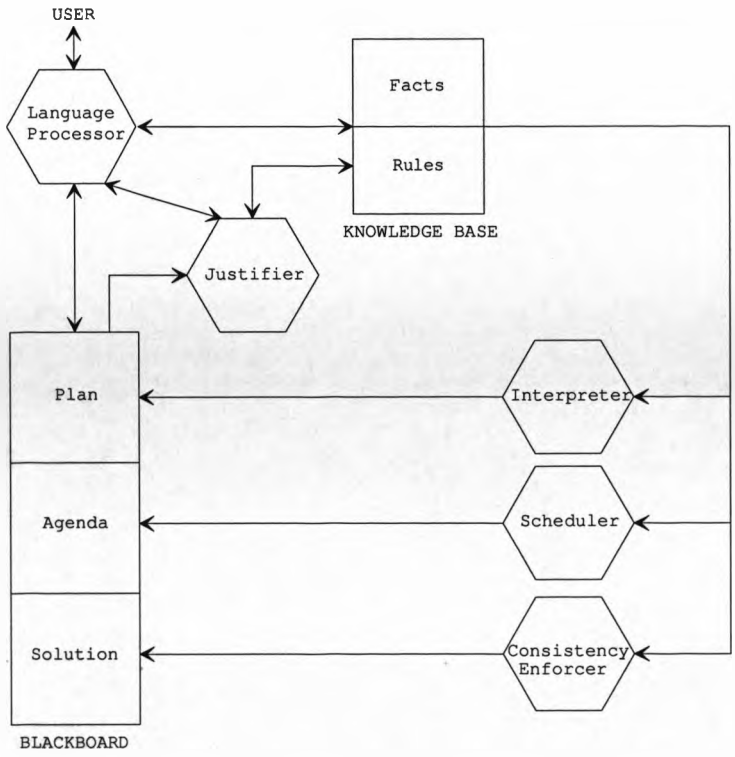


Figure 5.2: Anatomy of an ideal expert system

figure identifies three types of decisions recorded on the blackboard: plan, agenda and solution elements. Plan elements describe the overall attack the system will pursue against the current problem, including current plans goals, problem states and contexts. The agenda elements record the potential actions awaiting execution, which generally correspond to knowledge based rules that seem relevant to some decision placed on the blackboard previously. The solution elements represent the candidate hypotheses and decisions the system has generated thus far, along with the dependencies that relate decisions to one another. Often these dependencies are called *links.*

The scheduler maintains control of the agenda and determines which pending action should be executed next. Schedulers generally prioritise agenda items according to their relationship to the plan and other extant solution elements. Schedulers therefore need to estimate the affects of applying the potential rule.

The interpreter executes the chosen agenda item by applying the corresponding knowledge base rule. Generally, the interpreter validates the relevance conditions of the rule, binds variables in these conditions to particular solution blackboard elements, and then makes those changes to the blackboard that the rule prescribes.

The consistency enforcer attempts to maintain a consistent representation of the emerging solution. Most expert systems use some numerical adjustment scheme to determine the degree of belief in each potential decision. This scheme attempts to ensure that plausible conclusions are reached and inconsistent ones are avoided.

The justifier explains the actions of the expert system to the user. In general, it answers questions about why some conclusion was reached or why some alternative was rejected. To do this the justifier traces backward along blackboard solution elements from the questioned conclusion to the intermediate hypotheses or data that support it. The justifier collects these intermediate inferences and translates them into English for presentation to the user.

Finally, the knowledge base records rules, facts, and information about the current problem that may be useful in formulating a solution. Whereas the

rules of the knowledge base have procedural interpretations, the facts only play passive roles [Sel85].

### 5.2.2 Expert System Rules

Rules are used extensively in AI because each rule is simple to work with and because each rule can be considered independently of the others. This latter fact allows for the incremental construction of AI programs. Production rules, or if-then rules, are statements of the form:

$$\text{LHS} \rightarrow \text{RHS}$$

where LHS (left hand side) determines the conditions or situations that must be satisfied for the rule to be applicable and RHS (right hand side) is the action(s) that must be taken once the rule is applied. The terms *premise* or *condition* are frequently used for LHS and *conclusion* or *action* for RHS. The system progresses by inferring the conclusion whenever the conditions match the problem at hand. This is the simplest and most fundamental rule of inference and is known as *modus ponens.*

Often the rule of *modus ponens* is expressed using the if-then construct:

IF A1 THEN B1.

Each side of the rule may be in the form of a conjunction:

IF A1, A2, ..., Am THEN B1, B2, ..., Bn.

The above rule means that whenever A1, A2, ..., Am hold, actions B1, B2, ..., Bn must take place.

Central to expert systems are the concepts of *symbols* and search. "Physical symbol systems" manipulate collections of *symbolic structures* and perform problem-solving tasks using *heuristic search.* A symbol is defined as a physical pattern that can occur as a component of a symbol structure composed of a number of symbols related in some physical way, as being next to each other. Symbols can be thought of as strings of characters, while symbol structures are a type of data structure called list structures containing symbols. Examples of symbols are:

Transistor
Running
3.142

Examples of symbol structures are:

(On Block1 Block2)
(Research Location 1.01)

A number of factors affect the expert system in problem solving. For example, the inference engine must know where to start the process, in which direction to pursue the search for a solution, and how to make a choice when several rules are eligible for selection at once [HRWL83].

Expert systems allow for both (forward) data-driven and (backward) goal-driven computation. Starting from the initial known facts the system can go forward until the expected conclusion is reached or, alternatively, backwards from the goal until the path is completed.

*Forward chaining* can be regarded as a "recognise-act" cycle. In each cycle, by matching the contents of the working memory (blackboard) to the condition side of the rules, the expert system works out which rules have become eligible for execution. The interpreter chooses one of these rules, and adds the conclusion part of that rule to the working memory. A new cycle can now begin.

*Backward chaining* may be regarded as a "recognise-reduce" cycle. Backward chaining assumes that a rule is true and, by examining the rules, attempts to find a chain of inferences that can establish a link between the known facts and the goal. Thus proving the goal is indeed true.

Along with a decision on the direction of search, a methodology must be selected for making choices when more than one rule is a candidate for firing. There are numerous strategies available for this.

Two systematic methods that can be used with both forward and backward chaining are depth-first and breadth-first search. The depth-first search digs deeper and deeper in a spot, until the goal is found or the limit of the depth is reached. The search then examines and digs another spot. By contrast, a breadth-first search sweeps across all possible areas before going to a deeper level.

Forward chaining has a number of conflict resolution strategies for deciding which rule to fire next. Often a combination of these strategies is used to eliminate conflicts [Ade90]. Some of these strategies are:

1. *Elimination of executed rules:* Rules just executed are discarded from the conflict set, thus a rule is not allowed to fire more than once on the same set of facts.

2. *Textual position:* Choose the rule that appears earliest in the list of rules in the knowledge base. This requires a prioritising of rules when drawing up the list.

3. *Ordering of rules:* The rule with the highest priority will be chosen. This requires assignment of priority values to the rules.

4. *Specificity:* When the conditions of one eligible rule form a superset of the conditions of another triggered rule, then by specificity the rule with the larger number of conditions met is picked.

5. *Recency*: Priority is given to the rule that has been added to the conflict set most recently. This concentrates on finding more details about the current subproblem.

## 5.3 Evolutionary Computation

### 5.3.1 Genetic Algorithms

*Genetic algorithms* are a highly parallel technique that apply evolutionary processes to the solution of a variety of problems. The *genetic algorithm* transforms a *population* of individual objects, each with an associated *fitness* value, into a new *generation* of the population using the Darwinian principle of reproduction and survival of the fittest and naturally occurring genetic operators such as *crossover* (*recombination*) and *mutation*. Each *individual* in the population represents a possible solution to the given problem. The genetic algorithm attempts to find the best solution to the problem by genetically breeding the population of individuals. While randomised, genetic algorithms are not a simple random walk. They efficiently exploit historical information to speculate on new search points with expected improved performance.

In preparing to use the genetic algorithm operating on a fixed length character string to solve a problem, there are four major preparatory steps required [Gol89]:

1. Determine the representation scheme.
2. Determine the fitness measure.
3. Determine the parameters and variables for controlling the algorithm.
4. Determine a way of designating the result and a criterion for terminating the run.

For the conventional genetic algorithm, the individuals in the population are usually fixed-length character strings patterned after chromosome strings. The *representation scheme* is determined by choosing a string length $L$ and an alphabet size $K$. The most important part of the representation scheme is the mapping that expresses each possible point in the search space of the problem as a fixed-length character string (that is as a chromosome) and each chromosome as a point in the search space. This often requires a deep understanding of the problem.

The evolutionary process is driven by the *fitness measure.* The fitness measure assigns a fitness value to each possible string in the population.

The main parameters for controlling the genetic algorithm are the population size $M$ and the maximum number of generations to be run, $G$.

Each run of the genetic algorithm requires specification of a *termination criterion* for deciding when to terminate a run and a method of *result designation.* One frequently used method of result designation for a run of the genetic algorithm is to designate the best individual obtained in any generation of the population during the run (that is the *best-so-far individual*) as the result of the run.

Once these steps have been completed, the genetic algorithm can be run. A simple genetic algorithm that yields good results in many practical problems is composed of three operators:

1. Reproduction.

2. Crossover.

3. Mutation.

The genetic algorithm is as follows:

1. Randomly create an initial population of individual fixed-length character strings.

2. Iteratively perform the following substeps on the population of strings until the termination criterion has been satisfied:

   (a) Assign a fitness value to each individual in the population using the fitness measure.

   (b) Create a new population of strings by applying the following three genetic operations. The genetic operations are applied to individual string(s) in the population chosen with a probability based on fitness (with reselection allowed).

      i. Reproduce an existing individual string by copying it into the new population.

      ii. Create two new strings from two existing strings by genetically recombining substrings using the crossover operation at a randomly chosen crossover point.

      iii. Create a new string from an existing string by randomly mutating the character at one randomly chosen position in the string.

3. The string that is identified by the method of result designation (for example, the best-so-far individual) is designated as the result of the genetic

algorithm for the run. This result may represent a solution, or an approximate solution, to the problem.

Reproduction is the artificial version of the Darwinian principle of survival of the fittest. Reproduction selects individual strings to be copied, unchanged, to the next generation based on their fitness. Strings with a higher fitness have a higher probability of contributing one or more offspring to the next generation [Gol89].

Crossover allows new individuals, that is, new points in the search space to be created and tested. Crossover is a two step process. First two parents are independently selected, again selection is probabilistic based on the strings fitness. Secondly, a crossover point is randomly selected. Swapping all genetic material between the crossover site and the end of the string then creates two new strings. Crossover results in two offspring that each contains genetic material from both their parents. Crossover is illustrated below on two parent strings of length $L = 5$ over an alphabet size of $K = 2$. If a crossover site of $k = 3$, say, is randomly chosen we have the two parents, $P_1$ and $P_2$:

$$P_1 = 010|10$$

$$P_2 = 111|01$$

where | indicates the crossover site and the resulting offspring are $C_1$ and $C_2$:

$$C_1 = 01001$$

$$C_2 = 11110$$

The two offspring are usually different from their parents and different from each other.

Mutation allows new individuals to be created. A string is again randomly selected with a probability based on its fitness. Then a point on the string is randomly selected and the character at that point randomly mutated.

Despite the simplicity of reproduction and crossover, they give genetic algorithms much of their power. Mutation is needed because despite the power of reproduction and crossover to search the problem space they can occasionally lose some potentially useful genetic material. Mutation can help protect against such loss. The mutation operator plays a secondary role in the genetic operator, with mutation rates typically of the order of one per thousand [Gol89].

Despite the fact that the genetic algorithm knows nothing about the problem domain, operating only on the strings in the population, it is highly effective in searching even complex, highly nonlinear, multidimensional search spaces. The genetic algorithm searches the space of possible strings looking for highly fit strings. This search is carried out by the simple operations of reproduction, crossover and occasional mutation, guided by the fitness of the existing strings in the population.

### 5.3.2 Genetic Classifiers

Despite the strengths of the genetic algorithm, there is a problem. The problem however, lies not with the genetic algorithm itself, but with the struc-

tures it is required to adapt. The solution to this difficulty is to change the adapted structure. Genetic classifiers belong to a branch of engineering known as Genetics-Based Machine Learning (GBML) and use genetic search as their primary search heuristic. The adaptive systems theory of GBML pays special attention to the role of program replication as a method of emphasising past programs.

A classifier system is a machine learning system that learns syntactically simple string rules, called *classifiers*, to guide its performance in an arbitrary environment. A classifier system consists of three main components:

1. Rule and message system.
2. Apportionment of credit system.
3. Genetic algorithm.

A classifiers rule and message system is a special kind of *production system.* A production system is a computational scheme that uses rules as its only algorithmic device. Although they can take on many forms, they are generally of the form:

**if**<condition> **then**<action>.

That is, if the condition is satisfied then the action is taken (the rule is "fired").

Production systems are computationally both complete and convenient. A single rule or small set of rules can represent a complex set of thoughts compactly. Rule-based expert systems are production systems. Traditional

expert systems are not generally used in situations where learning is required, mainly because of their complex rule syntax. Classifiers differ in using fixed-length string representation for rules and actions. Using fixed-length strings means that genetic operators can operate on strings. This allows the use of the genetic algorithm to search the space of permissible rules [Koz92].

A further advantage of classifier systems over expert systems is that of speed. This is because expert systems use serial rule activation whereas classifiers use parallel rule activation, this permits multiple simultaneous thoughts and actions. When a classifier system must decide between mutually exclusive alternatives, it uses competitive arbitration strategies rather than the arbitrary procedures of expert systems.

An expert system has the value of its rules fixed by the programmer. In a classifier system the relative values of different rules is one of the most important pieces of information that needs to be learned. Classifier systems achieve this by holding a competition amongst the classifiers where the right to answer relevant messages goes to the highest bidder, with the subsequent payment of bids serving as a source of income to previously successful message senders. In this way, the good, profitable, rules survive and bad, unprofitable, rules die off.

The payment made to and from a rule increases and decreases its net worth, or strength. Strength determines a rules bid, it also serves as the rules fitness in a genetic algorithm search for new rules. Thus not only can the system learn by ranking extant rules, it can also discover new, possibly better rules as innovative combinations of its old rules. The genetic algorithm adopted in

classifier systems is very close to those used in search applications; however only a portion of the population is reproduced at a time, and more attention is focused on who replaces whom [Gol89].

### 5.3.3 Genetic Programming

*Genetic programming* is an offshoot of *genetic algorithms* in which the computer structures that undergo adaptation are themselves computer programs. Specialised genetic operators are used which generalise crossover and mutation for the tree structured programs undergoing adaptation. Genetic programming is capable of evolving computer programs that solve, or approximately solve, a variety of problems. The aim of genetic programming is to train computers to solve problems without being explicitly programmed.

The search space in genetic programming is the space of all possible computer programs composed of functions and terminals appropriate to the problem domain. The functions may be standard arithmetic operations, standard programming operations, standard mathematical functions, logical functions, or domain specific functions.

In applying genetic programming to a problem, there are five major preparatory steps. These are determining:

1. The set of terminals.
2. The set of primitive functions.
3. The fitness measure.

4. The parameters for controlling the run.

5. The method for designating a result and the criterion for terminating a run.

The first step is to identify the set of terminals. The terminals can be viewed as the inputs to the as-yet undiscovered computer program. The set of terminals (along with the set of functions) are the ingredients from which genetic programming attempts to construct a computer program to solve the problem.

The second step is to identify the set of functions that are to be used to generate the mathematical expression that attempts to fit the given finite sample of data.

Each computer program is a composition of functions from the *function set,* $F$ and terminals from the *terminal set,* $T$.

Each of the functions in the function set should be able to accept, as its arguments, any value and data type that may possibly be returned by any function in the function set. In addition, each of the functions in the function set should be able to accept any value and data type that may possibly be assumed by any terminal in the terminal set. That is, the function set and terminal set selected should have the *closure* property.

These first two major steps correspond to the step of specifying the representation scheme for the conventional genetic algorithm. The remaining three major steps for genetic programming correspond to the last three major

preparatory steps for the genetic algorithm [Koz94].

In genetic programming, populations of thousands of computer programs are genetically bred. This breeding is done using the Darwinian principle of survival and reproduction of the fittest, along with a genetic crossover operation appropriate for mating computer programs. From this process of genetic programming a computer program may emerge that solves, or approximately solves, a given problem.

Genetic programming starts with an initial population of randomly generated computer programs composed of functions and terminals appropriate to the problem domain. The creation of this initial random population is, in effect, a blind random search of the search space of the problem represented as computer programs.

Each individual computer program in the population is measured in terms of how well it performs in the particular problem environment. This measure is called the *fitness measure.* The nature of the fitness measure varies with the problem.

For many problems, fitness is naturally measured by the error produced by the computer program. The closer this error is to zero the better the computer program. For pattern recognition, for instance, the fitness of a particular program may be measured by some combination of the number of instances handled correctly (that is, true positives and true negatives) and the number of instances handled incorrectly (that is, false positives and false negatives). Typically, each computer program in the population is run over a number of

different *fitness cases* so that its fitness is measured as a sum or an average over a variety of representative different situations.

The computer programs in the initial generation (that is, generation 0) of the process will generally have exceedingly poor fitness. Nonetheless, some individuals in the population will turn out to be somewhat fitter than others are. These differences in performance are then exploited. Reproduction and crossover are used to create a new offspring population of individual computer programs from the current population of programs.

The reproduction operation involves selecting a computer program from the current population of programs based on fitness (that is, fitter programs are more likely to be selected) and allowing it to survive by copying it to the next generation.

The crossover operation is used to create new offspring computer programs from two parent programs selected based on fitness. The parental programs are typically of different sizes and shapes. The offspring programs are composed of subexpressions (subtrees, subprograms, subroutines and building blocks) from their parents. These offspring programs are generally of different sizes and shapes to their parents.

Intuitively, if two computer programs are somewhat effective in solving a problem, then some of their parts probably have some merit. Recombining randomly chosen parts of somewhat effective programs, may produce new computer programs that are even fitter at solving the given problem than either parent is.

The mutation operator may also be used in genetic programming.

After the genetic operations are performed on the current population, the population of offspring replaces the previous generation. Each individual in the new population of computer programs is then measured for fitness and the process is repeated over many generations.

At each stage of this highly parallel process, the state of the process will consist only of the current population of individuals. The force driving this process consists only of the observed fitness of the individuals in the current population in grappling with the problem environment [Kin94].

The genetic algorithm produces populations of computer programs that over many generations tend to exhibit increasing average fitness in dealing with their environment. In addition, these populations of computer programs can rapidly and effectively adapt to changes in the environment. The best individual appearing in any generation of a run (that is, the best so far individual) is, typically, designated as the result produced by the run of genetic programming.

## 5.4 Learning Expert Systems

As described above traditional expert systems have the values of their rules fixed by the expert during the programming of the expert system. Expert systems use a serial rule activation, that is, during each matching cycle only a single rule is activated.

In a learning expert system on the other hand, the relative value of different rules is one of the key pieces of information that must be learned. To enable this learning the classifier is forced to exist in an information based service economy. The classifiers take part in an auction where the right to answer relevant messages goes to the highest bidder. The payment of the bid to previously successful message senders provides their source of income. The competitive nature of the economy ensures that good, that is, profitable, rules survive and bad, that is, unprofitable, rules die off. Learning expert systems overcome the bottleneck of serial rule activation in traditional expert systems by using parallel rule activation during a matching cycle. Thus, learning expert systems permit multiple activities to be coordinated simultaneously. If choices must be made between mutually exclusive actions, or when the size of the matched rule set must be pruned to accommodate the size of the fixed message list, these choices are left to the last possible moment and the arbitration is then performed competitively. A diagram of a learning expert system is shown in Figure 5.3.

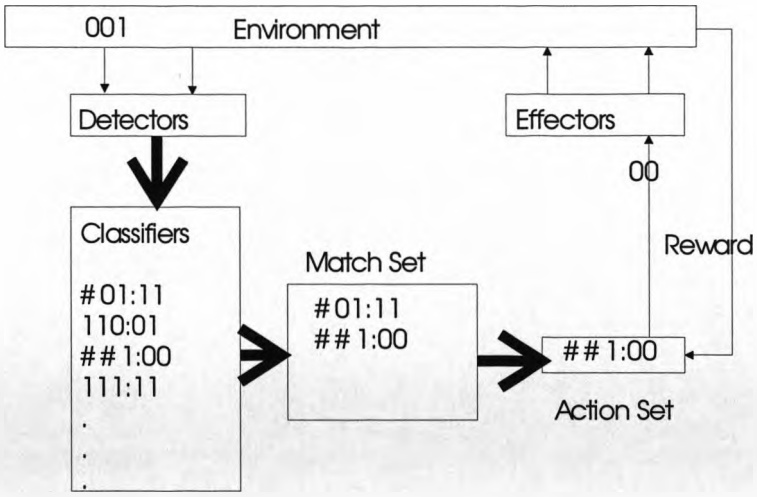


Figure 5.3: A learning expert system

### 5.4.1 Message Coding

A learning expert system, or classifier, is a special kind of production system that uses rules as its only algorithmic device. The rules are usually of the following form:

**if**<condition> **then**<action>.

That is, if the condition is satisfied then the action is taken (the rule is "fired").

The **if** part of the rule is often referred to as the *condition.* The **if** operator is a binary string encoding of the problem state space. The **then** part of the rule is known as the message and is a binary encoding of possible classes that the input data can be placed in.

Despite appearing restricting, such systems are computationally complete. They are also computationally convenient. A single rule or small set of rules can compactly represent a complex set of thoughts.

Learning expert systems differ from the traditional systems by restricting a rule to a fixed-length representation. This means that all strings under the permissible alphabet are syntactically meaningful. In addition, fixed string representation permits the use of genetic string operators. Thus it becomes possible to search the space of possible rules using genetic algorithms [Koz92].

A message within a learning system is simply a finite-length string over some finite alphabet. For example, using the binary alphabet:

$$< \text{message} >::= \{0,\ 1\}^l \tag{5.4.1}$$

That is, the message is a concatenation of $l$ 0's or 1's.

Messages form the basic currency in the information exchange of a classifier system. The messages on the message list may match one or more classifiers or string rules. A classifier is a production rule with the syntax:

$$< \text{classifier} >::=< \text{condition} >:< \text{message} > \tag{5.4.2}$$

The condition is a simple pattern recognition device where a # is added as a don't care symbol, that is, it can match either a 0 or a 1.

The use of a don't care symbol allows a form of generalisation in the condition part of the rule:

$$< \text{condition} >::= \{0,\ 1,\ \#\}^l \tag{5.4.3}$$

If at every position in a condition a message has, a 0 matching a 0, or a 1 matching a 1, or a # matching either a 0 or a 1, then the condition is matched

by the message. For example the condition #1#0 matches the message 1100 but doesn't match the message 0011.

Once a classifiers condition is matched, that classifier becomes a candidate to post its message to the message list at the next time step. Whether the classifier posts its message depends upon the fitness of the classifier as it takes part in an activation auction [Gol89].

### 5.4.2 Training the Intelligent System

A random population of fixed-length strings, each representing an individual if-then rule, is created. Each string is then added to a rule base of strings. Next the first element of the input set is posted from the environment. Any rule that matches the message posted by the environment posts its message to a *match set.* It should be noted that because the "wild card", #, allows for generalisation of the condition, a number of rules that are not identical may be posted. Messages that are posted may then activate other rules or may cause the system to take some external action through its effectors.

The next stage is to determine which rules in the match set shall be placed in the *action set.* To achieve this all rules in the match set take part in an auction. Each rule makes a *bid* in the auction that is proportional to its *strength.* In this way the fitter rules, that is, those that have been activated more often, are given preference over less fit rules.

If a rule is successful in the auction its strength is debited by the size of its bid. The successful rules bid is then distributed amongst those rules that

activated it. The successful rule is then placed in the action set.

When the successful rule has posted its message this may in turn invoke other rules by matching their condition. These rules will then take part in further auctions. The next environmental message is then posted and this process continues iteratively until all messages from the environment have been posted.

The next stage is to apply the genetic algorithm in an attempt to create new and hopefully better rules. The genetic algorithm creates new rules using the processes of reproduction, crossover and mutation.

Rules that have been activated more often in the first stage of the process will have a higher fitness function. These rules will therefore be more likely to be selected for the operations of the genetic algorithm. Hence, the new rules created by the genetic algorithm will be based on previously successful rules.

These new rules are then posted to the system. The new rules are processed, by matching messages to conditions and auctions as before, to determine their fitness. However, in these systems care generally has to be taken about replacing the whole population (usually the whole population would not be replaced) and to which rule replaces which [Gol89].

## 5.5 Conclusion

An expert system is a knowledge-based system that emulates expert thought to solve significant problems in a particular domain of expertise. Expert systems are rule-based systems that employ, symbolic representation, symbolic inference and heuristic search. Two independent systems make up an expert system: a knowledge base and an inference engine. The knowledge base holds whatever information, rules and facts for example, that are appropriate to solving problems in the given domain. The inference engine provides the motive power to the system. Its function is to employ the contents of the knowledge base to draw inferences, thus enabling it to attempt to solve a particular problem.

Evolutionary computation implicitly utilises a directed search, allowing it to search the space of possible computer structures and find solutions to tasks in much shorter times than would be possible using random searches. The genetic algorithm uses the Darwinian processes of natural selection to breed new populations of individuals containing the fittest members of the old population. Each individual represents a possible solution to the problem. The main genetic operators are reproduction and crossover (sexual reproduction). Both reproduction and crossover operate with a probability based on the fitness of the individuals in the population. A secondary operator is mutation. One of the difficulties with the genetic algorithm is that it is not suitable for the structures we choose to adapt. Therefore, genetic classifiers are used as

machine learning systems that use genetic search as their primary discovery heuristic. Genetic programming is an extension of the conventional genetic algorithm in which the structures undergoing adaptation are hierarchical computer programs of dynamically varying size and shape.

Learning expert systems have their rule sets adaptively modified by genetic operators. The learning expert system architecture combines important features from the models of artificial intelligence, connectionism and machine learning. Artificial intelligence provides the power, understandability and convenience of an expert systems if-then rules. When the system as a whole takes some action that is rewarded by the environment, specific rules, that contributed to the action, are rewarded via a connectionist style credit allocation. Machine learning provides the system with the creative power and efficient search capability of the genetic algorithm operating on fixed length strings.

Time constraints prevented the application of this methodology to the problems described in this thesis. However, future work could continue the development of an intelligent monitoring and prediction system consisting of: signal processing to extract the key features from the raw data, neural networks for pattern classification and a learning expert system for the adaptive adaptation of the systems rule base as illustrated in Figure 5.4.

For example this methodology could be implemented in the following way. Using the infrared scatter data select a sample of the Low and High data for training purposes. To create training vectors extract the wavelet coefficients from chosen data samples. Create the rule base and then test the rules. In

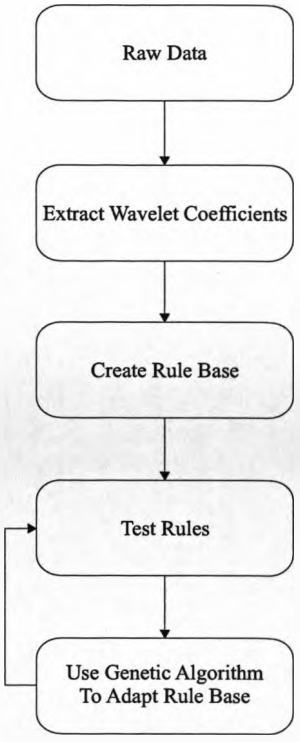


Figure 5.4: An intelligent monitoring and prediction system

an iterative process use the genetic algorithm to adapt the rule base and then retest the rules. It is expected that this architecture could form the basis of an intelligent monitoring and prediction system that could be adapted to other real world time series prediction problems.

Overall, this work has illustrated the value of intelligent monitoring and prediction systems to the problems described in this thesis. Clearly, this would be an important system for further investigation.

# Chapter 6

# Conclusion

## 6.1 Introduction

The main aim of the work described in this thesis was to develop novel methodologies for real world chaotic time series prediction and classification problems. The objective was to produce an intelligent monitoring and prediction system based on various information techniques, neural networks and genetic classifiers.

As discussed elsewhere in this thesis, three sets of real world industrial data were examined during the course of this research. *Shell UK* provided two sets of data for this work. The first set consists of infra-red scatter data and the second set is catalytic cracker data. Further data was provided by NGC from both a normal and a faulty power transformer.

The infra-red scatter data comes from three infra-red detectors that are arranged spatially around a common reference point and are used to measure

oil mist. The data falls into four alarm categories: High, Low, Grey and None. The physical significance of the data is not known. The aim was to develop an alarm system for remote unmanned oil installations that is capable of detecting whether or not there is an oil leak and alerting the operators in the case of a positive alarm.

The second set of data is from a catalytic cracker. The aim was to develop a predictive model for this industrial process data. As with many cases of real world industrial data, the data contains a large number of "drop-outs" and "spikes". Where variables are missing, the missing values have been replaced by taking the mean value of the two data points either side of the missing measurement. The data consists of 95 process variables measured over nearly three years. The idea was to use v36-v95 to predict v1-v35, concentrating on v4 and v6. Again the physical significance of this data is not provided.

The third set of industrial process data is provided by NGC. This consists of two sets of power transformer data: the first set is from a transformer working normally and the second set is from a faulty transformer. The aim was to develop a more comprehensive and systematic approach to the monitoring of changes in power transformers operating conditions.

## 6.2 Methodologies

Back propagation neural networks and radial basis function neural networks were both examined as a method of classifying any patterns in the industrial data provided. In order to simplify the neural networks required for pattern classification the raw data was preprocessed to extract its key features.

A number of techniques were examined for extracting the key features from the raw data. These techniques included: the autocorrelation function, the cross-correlation function and the power spectral density function. However, these techniques all resulted in an increasing of the dimensionality of the pattern space. Therefore further methods of extracting the data's key features were examined. These were principal component analysis and the wavelet transform.

Genetic classifiers, learning expert systems, were also investigated as components of an intelligent monitoring and prediction system.

## 6.3 Results

### 6.3.1 Infra-Red Scatter Data

The first data set examined was the infra-red scatter data. Initially a radial basis function neural network was trained using the raw data. This produced a neural network that could correctly classify training vectors but was unable to generalise, that is, to correctly classify test vectors once trained.

The raw data was then preprocessed using a number of techniques including: cross-correlation, autocorrelation, power spectral density and the signal statistics, mean, variance and eigenvalues. As with the raw data a radial basis function neural network has been successfully trained to reach a desired sum squared error goal. However, none of the networks have been able to generalise correctly when presented with test vectors. Therefore, an alternative neural network paradigm, the multilayer perceptron, was investigated.

Multilayer perceptron neural networks with one and two hidden layers of nonlinear neurons and one output layer of linear neurons were also successfully trained. The neural networks were trained with data preprocessed by each of the above techniques used for preprocessing the data for a radial basis function network. Several neural networks were generated during the development of this model; each with different combinations of neurons in various layers.

Despite long training times, with up to 500,000 epochs elapsing, none of the neural networks successfully reached the required error goal during training and were unable to correctly classify the training data.

In an attempt to improve the response of the neural networks to the test data, principal component analysis was used to reduce the dimensionality of the data. Using the principal components of the scatter data to train backpropagation networks produced networks that were unable to complete the training process. The networks failed to reach their error goal and misclassified training data when it was re-presented to the network.

The infra-red scatter data was then processed using the wavelet transform and a scatter plot of the correlation coefficients of the wavelet coefficients was examined to determine the possibility of a neural network separating the different training patterns. These coefficients where then used to train back propagation neural networks.

This work has developed and trained many multilayer perceptron neural networks, these different have different numbers of neurons both in the hidden and output layers. Furthermore, these networks have been able to correctly classify training patterns when they are re-presented to the trained network. However, further work is needed to fully develop neural networks that will consistently generalise correctly when presented with test vectors.

The infra-red scatter data was then normalised and the standardised data's wavelet coefficients were extracted. Again a scatter plot of the wavelet coefficients was examined and the data was used to train backpropagation networks with a range of neurons in the hidden and output layers. This protocol generated several neural networks that trained correctly. In addition, when re-presented with training data after training was completed the network classified

it correctly. However, the neural networks performed poorly when presented with test vectors. Therefore, further studies are needed to fine tune this system. The neural networks were able to accurately and correctly predict from previously unseen test data in $40-60\%$ of occasions. However, for commercial development, further work is needed to increase this accuracy to that required of a real time alarm system.

### 6.3.2 Catalytic Cracker Data

The second set of data is a set of ninety five variables from a catalytic cracker used in oil refineries. Because of the high dimensionality of the data, it is necessary to preprocess the data in order to extract its key features and hence reduce the dimensionality of the problem. A principal component analysis was performed on the data to identify the key variables for monitoring.

Use of either the retention or rejection method based on the component scores, produced largely the same results as to which variables accounted for most of the variance of the catalytic cracker data, although there were some anomalies.

Principal component analysis neural networks were also trained. However, the results from these networks were inconclusive and require further examination with other principal component algorithms.

Work focused on the related processes v60-v72 to try to predict v4 and v6. A multilayer perceptron neural network was successfully trained using this data. Again, despite using both one and two hidden layers and various numbers

of neurons in the hidden layers, it has not been possible to successfully train a neural network.

### 6.3.3 Power Transformer Data

Principal Components Analysis can be used to identify any variation in the correlation of the transformers operating variables caused by changes in the transformers condition. From the above results, comparing the normal and faulty transformers, it is noted that the key variables selected by PCA under a normal operating condition, will become more correlated and the variance of the principal components will have a more uneven distribution, if the transformers operating condition changes. This is due to the key variables selected by PCA forming a maximum information space. Any change in the transformer condition, represented by the same variables, will result in a contraction of the information space. An index number has been introduced to measure the distribution of the variance of the principal components. It also reflects the tendency of the correlations of the key transformer parameters to vary. It has been found that the ratio between the index numbers of the faulty and normal transformers is sufficiently large that it could be used to report the variation of the transformers condition. The index number could therefore be used as an alarm for the purposes of fault monitoring.

As the data from the normal and faulty transformers were not observed from identical transformers and they were sampled within a limited operating range, the sensitivity of PCA as applied to condition monitoring could not

be investigated. The quantitative analysis of this method requires more data, obtained from a variety of different transformer conditions, in order to provide an accurate assessment of the condition variation. However, by contrast to the conventional correlation analysis methods which have been applied, for example, for monitoring of transformer partial discharges, the PCA provides a more systematic and comprehensive approach to analysis of correlation variations occurring as the transformer condition changes. Although the PCA cannot be used directly for fault diagnosis, the information extracted by the PCA could form a maximum-information space for further classification leading to an identification of the details of transformer condition variations.

Overall, the studies described in this thesis have shown several different methodologies that have been used to develop systems for real world time series prediction and classification problems. Initial studies used multilayer backpropagation neural networks for data extraction. However, although the neural networks trained successfully they could not interpret new data consistently.

Principal component analysis and the wavelet transform were used to identify key data variables and reduce data dimensionality. Although artificial neural networks could be trained successfully using the pre-processed data, interpretation of test data by the neural networks was still difficult and inconsistent. One method that may prove useful and important for analysis of these data is learning expert systems. The application of this methodology to the problems described in this thesis has been illustrated in Chapter 5. Clearly the

work described in this thesis provides the basis for a variety of future studies to continue the development of intelligent monitoring and prediction systems.

# References


[Ade90] H. Adeli. *Knowledge Engineering Vol. 1, Fundamentals.* McGraw-Hill Publishing Company, 1990.

[BBPB96] P. Bozzola, G. Bortolan, F. Pinciroli, and C. Brohet. A hybrid neuro-fuzzy system for ecg classification of myocardial infarction. *IEEE Computers in Cardiology*, 1996.

[BCD+95] J. Buckheit, S. Chen, D. Donoho, I. Johnstone, and J. Scargle. About wavelab. Stanford University Web Site, November 1995.

[BDG96] A. Bruce, D. Donoho, and H-Y. Gao. Wavelet analysis. *IEEE Spectrum*, October 1996.

[BH96] B. Burke Hubbard. *The World According to Wavelets.* A.K. Peters, 1996.

[BL88] D.S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, (321-355), 1988.

[BP71] J.S. Bendat and A.G. Piersol. *Random Data: Analysis and Measurement Procedures.* Wiley-Interscience, 1971.

[Bri88] E.O. Brigham. *The Fast Fourier Transform And Its Applications.* Prentice-Hall Inc, 1988.

[Buh95] J.M. Buhmann. *The Handbook of Brain Theory and Neural Networks*, chapter Data Clustering and Learning. MIT Press, 1995.

[Cai61] E.R. Caianiello. Outline of a theory of thought processes and thinking machines. *Journal of Theoretical Biology*, 1:204–235, 1961.

[CCG91] S. Chen, C.F.N. Cowan, and P.M. Grant. Orthogonal least squares algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, 2(2), March 1991.

[CCM96] E.S. Chng, S. Chen, and B. Mulgrew. Gradient radial basis function networks for nonstationary and nonlinear time series prediction. *IEEE Transactions on Neural Networks*, 7(1), January 1996.

[Chu92] C.K. Chui. *An Introduction to Wavelets.* Academic Press Inc, 1992.

[CK96] A. Cohen and J. Kovačević. Wavelets: The mathematical background. *Proceedings of the IEEE*, 84(4), April 1996.

[Dau74] S. Daultrey. *Principal Component Analysis.* Geo Abstracts Ltd, 1974.

[Dau88] I. Daubechies. Orthonormal basis for compactly supported wavelets. *Comm. Pure Appl. Math.*, 41:906–966, 1988.

[Dau92] I. Daubechies. Ten lectures on wavelets. *Society for Industrial and Applied Mathematics*, 1992.

[DKK96] Diamantaras and S.Y. K.I. Kung. *Principal Component Neural Networks.* Wiley, New York, 1996.

[Duv89] M. Duval. Dissoloved gas analysis: It can save your transformer. *IEEE Electrical Insulation Magazine*, 5(6):22–27, 1989.

[ESM94] K. Esbensen, S. Schönkopf, and T. Midtgrarrd. *Multivariable Analysis in Practice.* Camo As, 1994.

[FTK96] O. Fukada, T. Tsuji, and M. Kaneko. Pattern classification of time-series eeg signals using neural networks. *IEEE International Workshop on Robot and Human Communication*, 1996.

[Gab54] D. Gabor. Communication theory and cybernetics. *IRE Transactions on Circuit Theory*, CT-1:19–31, 1954.

[GM84] A. Grossmann and J. Morlet. Decomposition of hardy functions into square integrable wavelets of constant shape. *SIAM Journal of Mathematical Analysis*, 15:723–726, 1984.

[Gol89] D.E. Goldberg. *Genetic Algorithms in Search Optimisation and Machine Learning.* Addison-Wesley Publishing Company Inc, 1989.

[Gon86] M. Gondran. *An Introduction to Expert Systems.* McGraw-Hill Book Company (UK) Limited, 1986.

[Gra95] A. Graps. An introduction to wavelets. *IEEE Computational Science and Engineering*, 2(2), 1995.

[Has95] M.H. Hassoun. *Fundamentals of Artificial Neural Networks.* The MIT Press, 1995.

[Hay94] S Haykin. *Neural Networks, A Comprehensive Foundation.* Prentice-Hall Inc., 1994.

[Heb49] D.O. Hebb. *The Organisation of Behaviour: A Neuropsychological Theory.* Wiley, 1949.

[HKP91] J. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the Theory of Neural Computing.* Addison-Wesley Publishing Co., 1991.

[Hol62] J.H. Holland. Outline for a logical theory for adaptive systems. *Journal of the Association of Computing Machinery*, 3:297–314, 1962.

[Hol65] J.H. Holland. *Electronic Information Handling*, chapter Some Practical Aspects of Adaptive Systems Theory, pages 209–217. Spartan Books, 1965.

[Hol71] J.H. Holland. *Associative Information Processing*, chapter Processing and Processors for Schemata, pages 127–146. American Elsevier, 1971.

[Hol75] J.H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Washington Press, 1975.

[Hop82] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the USA.*, 79:2554–2558, 1982.

[HR78] J.H. Holland and J.S. Reitman. *Pattern Directed Inference Systems*, chapter Cognitive Systems Based on Adaptive Algorithms, pages 313–329. Academic Press, 1978.

[HRWL83] F. Hayes-Roth, D.A. Waterman, and D.B. Lenat. *Building Expert Systems.* Addison-Wesley Publishing Company Inc, 1983.

[HY+97] Y.C. Huang, H.Y. Yang, et al. Developing a new transformer fault diagnosis system through evolutionary fuzzy logic. *IEEE Transactions on Power Delivery*, 12(2):761–767, 1997.

[II97] R.A. Ibata and M.J. Irwin. Discrete classification with principal component analysis: Discrimination of giant and dwarf spectra in k stars. *The Astronomical Journal*, 113(5), May 1997.

[JB94] T. Jackson and R. Beale. *Neural Networks: An Introduction.* Institute of Physics Publishing, 1994.

[Joh] C.M. Johnson. Signal classification using wavelets and neural networks. *SPIE*, 2762.

[Jol72a] I. T. Jollife. Discarding variables in a principal component analysis. i: Artificial data. *Applied Statistics*, 21, 1972.

[Jol72b] I. T. Jollife. Discarding variables in a principal component analysis. ii: Real data. *Applied Statistics*, 22, 1972.

[Jou97] J. Joutsensalo. A class of neural networks for independent component analysis. *IEEE Transactions on Neural Networks*, 8(3), May 1997.

[Kin94] K.E. Kinnear, Jr. *Advances in Genetic Programming.* The MIT Press, 1994.

[Kos93] B. Kosko. *Fuzzy Thinking.* Flamingo, 1993.

[KOW+97] J. Karhunen, E. Oja, L. Wang, A. Vigário, and J. Joutsensalo. A class of neural networks for independent component analysis. *IEEE Transactions on Neural Networks*, 8(3), May 1997.

[Koz92] J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press, 1992.

[Koz94] J.R. Koza. *Genetic Programming II Automatic Discovery of Reusable Programs.* The MIT Press, 1994.

[KP] G.S. Kapogiannopoulos and M. Papadakis. Character recognition using a biorthogonal discrete wavelet transform. *SPIE*, 2825.

[LB95] Y. LeCun and Y. Bengio. *The Handbook of Brain Theory and Neural Networks*, chapter Pattern Recognition. MIT Press, 1995.

[Lin88] R. Linsker. Self-organisation in a perceptual network. *Computer*, 21:105–117, 1988.

[LJWR00] H.F. Lewis, L. Jiang, Q.H. Wu, and Z. Richardson. Key variables identification and interdependence analysis in power transformers. *UPEC 2000 Conference Proceedings*, 2000.

[Low95] D. Lowe. *Radial Basis Function Networks*, chapter The Handbook of Brain Theory and Neural Networks. MIT Press, 1995.

[Lyn89] P.A. Lynn. *An Introduction to the Analysis and Processing of Signals.* MacMillam Education Ltd., third edition, 1989.

[Mal89a] S. Mallat. A compact multiresolution representation: The wavelet model. *Trans. Amer. Math. Soc.*, 315:69–88, 1989.

[Mal89b] S. Mallat. Multiresolution approximation and wavelets. *Trans. Amer. Math. Soc.*, 315:69–88, 1989.

[Mal89c] S. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *Proc. IEEE Trans. on Pattern Analysis and Machine Intelligence*, 11(7):674–693, July 1989.

[MCKDV97] Y. Mallet, D. Coomans, J. Kautsky, and O. De Vel. Classification using adaptive wavelets for feature extraction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(10), October 1997.

[Mey92] Y. Meyer. Wavelets: Algorithms and applications. *Society for Industrial and Applied Mathematics*, 1992.

[Min61] M.L. Minsky. Steps towards artificial intelligence. *Proceeding of the Institute of Radio Engineers*, 49:8–30, 1961.

[MP43] W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activivty. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[MP69] M.L. Minsky and S.A. Papert. *Perceptrons.* MIT Press, 1969.

[MS+96] H. Miao, M. Sforma, et al. A new logic-based alarm analyzer for on-line operational environment. *IEEE Trans. on Power Systems*, 11(3):1600–1606, 1996.

[Nil65] N.J. Nilsson. *Learning Machines: Foundations of Trainable Pattern-Classifying Systems.* McGraw-Hill, 1965.

[nn:92] *Neural Networks Toolbox Users Guide.* The Math Works Inc., 1992.

[Oja82] E. Oja. A simplified neuron as a principal component analyzer. *Journal of Mathematical Biology*, 15:267–73, 1982.

[Oja89] E. Oja. Neural networks, principal components, and subspaces. *International Journal of Neural Systems*, 1(1), 1989.

[Oja95] E. Oja. *The Handbook of Brain Theory and Neural Networks*, chapter Principal Component Analysis, pages 753–6. MIT Press, 1995.

[Plu95] M.D. Plumbley. Lyapunov functions for convergence of principal component analysis. *Neural Networks*, 8(1), 1995.

[Pola] R. Polikar. The wavelet tutorial, part i. WWW.

[Polb] R. Polikar. The wavelet tutorial, part ii. WWW.

[Polc] R. Polikar. The wavelet tutorial, part iii. WWW.

[Pold] R. Polikar. The wavelet tutorial, part iv. WWW.

[RHHD56] N. Rochester, J.H. Holland, L.H. Habit, and W.L. Duda. Tests on a cell assembly theory of the action of the brain using a large digital computer. *IRE Transactions on Information Theory*, IT-2:80–93, 1956.

[RHW86] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[RM86] D.E. Rumelhart and J.L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press, 1986.

[Rog78] R.R. Rogers. Ieee and iec codes to interpret faults in transformers, using gas in oil analysis. *IEEE Trans. Electrical Insulation*, EI-13(5):349–354, 1978.

[Ros58] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organisation in the brain. *Psychological Review*, 65:386–408, 1958.

[Ros60] F. Rosenblatt. On the convergence of reinforcement procedures in simple perceptrons. Technical Report VG-1196-G-4, Cornell Aeronautical Laboratory, Buffalo, NY., 1960.

[Ros92] S. Rose. *The Making of Memory.* Bantam Press, 1992.

[San89] T.D. Sanger. Optimal unsupervised learning in a single-layer

linear feedforward neural network. *Neural Networks*, 2:459–473, 1989.

[Sel85] P.S. Sell. *Expert Systems - A Practical Introduction.* MacMillan Publishers Ltd, 1985.

[SLL+] G. Székely, T. Lindblad, C. Lindsey, M. Minerskjöld, and G. Sekhniaidze. Wavelets and signal processing. *SPIE*, 2760.

[sp:92] *Signal Processing Toolbox Users Guide.* The Math Works Inc., 1992.

[Str80] G. Strang. *Linear Algebra and its Applications.* Academic Press, second edition edition, 1980.

[Str89] G. Strang. Wavelets and dilation equations: A brief introduction. *Siam Review*, (31), 1989.

[Str90] F.G. Stremler. *Introduction to Communication Systems.* Addison Wesley, third edition, 1990.

[Str93] G. Strang. Wavelet transforms versus fourier transforms. *Bulletin of the American Mathematical Society*, 28(2), April 1993.

[Str94] G. Strang. Wavelets. *American Scientist*, (82), April 1994.

[Tay56] W.K. Taylor. *Electrical Simulation of Some Nervous System Functional Activities*, volume 3. Butterworth, 1956.

[Tur52] A.M. Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society*, Series B(237):5–72, 1952.

[Ued95] M. Ueda. *Wavelets: An Elementary Introduction and Examples.* UCSC-CRL, 1995.

[Utt56] A.M. Uttley. A theory of the mechanism of learning based on the computation of conditional probabilities. *Proceedings of the 1st International Conference of Cybernetics*, 1956.

[Utt79] A.M. Uttley. *Information transmission in the nervous system.* Academic Press, 1979.

[VM91] B. Vidaković and P. Müller. Wavelets for kids. WWW, 1991.

[vN56] J. von Neumann. *Automata Studies*, chapter Probabilistic and the Synthesis of Reliable Organisms from Unreliable Components, pages 43–98. Princeton University Press, 1956.

[Was93] P.D. Wasserman. *Advanced Methods in Neural Computing.* Van Nostrand Reinhold, 1993.

[WBLH69] D.J. Willshaw, O.P. Buneman, and H.C. Longuet-Higgins. Non-holographic associative memory. *Nature*, (222):960–962, 1969.

[WC63] S. Winograd and J.D. Cowan. *Reliable Computation in the Presence of Noise.* MIT Press, 1963.

[Wei94] L.G. Weiss. Wavelets and wideband correlation processing. *IEEE Signal Processing Magazine*, January 1994.

[WH60] B. Widrow and M.E. Hoff Jr. Adaptive switching circuits. *IRE WESCON Convention Record*, 1960.

[Wid62] B. Widrow. *Self-Organising Systems*, chapter Generalisation and Information Storage in Networks of Adaline 'Neurons', pages 435–461. Sparta, 1962.

[Wie48] N Wiener. *Cybernetics: Or, Control and Communication in the Animal and the Machine.* Wiley, 1948.

[Wol78] S. Wold. Cross-validatory estimation of the number of components in factor and principal component analysis. *Technometrics*, 20, November 1978.

[Xu95] Xu. Robust principal component analysis by self-organising rules based on statistical physics approach. *IEEE Transactions on Neural Networks*, 6(1), January 1995.

[ZB92] Q. Zhang and A. Benveniste. Wavelet networks. *IEEE Transactions on Neural Networks*, 3(6), November 1992.

[Zha97] Q. Zhang. Using wavelet network in nonparametric estimation. *IEEE Transactions on Neural Networks*, 8(2), March 1997.

[ZL95] Q. Zhang and Y-W. Leung. Energy function for the one-unit oja algorithm. *IEEE Transactions on Neural Networks*, 6(5), September 1995.

[ZWML95] J. Zhang, G.G. Walter, Y. Miao, and W.N.W. Lee. Wavelet neural networks for function learning. *IEEE Transactions on Signal Processing*, 43(6), June 1995.