APPLICATION OF SPIKING NEURAL NETWORKS AND THE BEES ALGORITHM TO CONTROL CHART PATTERN RECOGNITION

A thesis submitted to

Cardiff University,

for the degree of

Doctor of Philosophy

by

Shahnorbanun Sahran, B.Science (Hons), MSc

Manufacturing Engineering Centre
Cardiff University
United Kingdom

2007

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

ABSTRACT

Statistical process control (SPC) is a method for improving the quality of products. Control charting plays a most important role in SPC. SPC control charts are used for monitoring and detecting unnatural process behaviour. Unnatural patterns in control charts indicate unnatural causes for variations. Control chart pattern recognition is therefore important in SPC. Past research shows that although certain types of charts, such as the CUSUM chart, might have powerful detection ability, they lack robustness and do not function automatically.

In recent years, neural network techniques have been applied to automatic pattern recognition. Spiking Neural Networks (SNNs) belong to the third generation of artificial neural networks, with spiking neurons as processing elements. In SNNs, time is an important feature for information representation and processing. This thesis proposes the application of SNN techniques to control chart pattern recognition. It is designed to present an analysis of the existing learning algorithms of SNN for pattern recognition and to explain how and why spiking neurons have more computational power in comparison to the previous generation of neural networks.

This thesis focuses on the architecture and the learning procedure of the network. Four new learning algorithms are presented with their specific architecture: Spiking Learning Vector Quantisation (S-LVQ), Enhanced-Spiking Learning Vector Quantisation (NS-LVQ), S-LVQ with Bees and NS-LVQ with Bees. The latter two algorithms employ a new intelligent swarm-based optimisation called the *Bees*

Algorithm to optimise the LVQ pattern recognition networks. Overall, the aim of the research is to develop a simple architecture for the proposed network as well as to develop a network that is efficient for application to control chart pattern recognition. Experiments show that the proposed architecture and the learning procedure give high pattern recognition accuracies.

DEDICATION

THIS WORK IS ENTIRELY DEDICATED TO MY BELOVED FAMILY

TO MY HUSBAND: AZHAR MOHD KHALEL

who has greatly encouraged and supported me during my studies

TO MY CHILDREN: MUHAMMAD FIRHAN AZHAR

: LUKHMAN HAKIM AZHAR

: NUR KHAISUMAH AZHAR

: HILAL AZHAR

: UWAIS AZHAR

who always brings joy to my life

TO MY PARENTS: FATIMAH KHALIL KHAN & SAHRAN UTOH

who always pray for successfulness for my life

TO MY PARENTS-IN-LAW: HABIBAH ISMAIL & MOHD KHALEL MOHD DAHAN

who always wonderfully supported me throughout my education

ACKNOWLEDGEMENT

First, I thank ALLAH (My Lord) the all high, the all great and merciful who made it possible for me to complete this work.

I am privileged to have Professor D.T.Pham as my supervisor. The high standard of his research has always been an inspiration and a goal to me. I am deeply grateful to him for his consistent encouragement, invaluable guidance and strong support during the course of this study. His thoughtful advice and constant support extended to me will always be remembered. May ALLAH bless him and his family.

I wish to express my sincere thanks to the Cardiff University, especially the Manufacturing Engineering Centre (MEC) for the use of the facilities to pursue this research.

I am also very grateful to all the members of the MEC for their friendship and helps. Special thanks go to my fellow Ph.D. students Mr. Charles and the Bay Bees colony Mr. Afshin, Mr. Samch, Mr. Ebubekir, Mr. Marco, Mr. Zaidi and Miss Jaynee for their useful discussion and sincere help whenever I needed them. My deeply thanks to my dear friend Dr. Zarina, Mrs. Siti Aishah, Dr. Zakaria, Mrs. Yuhanis, Mr. Massudi and Mr. Yahya for their sincere help regarding programming C++.

I would also like to thank all the members of the Malaysian community in Cardiff for their support and friendship. I am very grateful and acknowledge the substantial financial support from Universiti Kebangsaan Malaysia (UKM) and Public Service

Department of Malaysia. Sincere thanks are also to all the members of staff of the Faculty of Information Science and Technology especially the Department of Industrial Computing, Universiti Kebangsaan Malaysia, who taught me and gave me the scientific base to continue my postgraduate studies.

I wish to express my heartfelt gratitude to my parents, Fatimah Khalil Khan and Sahran Utoh for all the love and consistent support they have given me. I also want to express my warmest thanks to my parents-in-law, Habibah Ismail and Mohd Khalel Mohd Dahan for their love and support in my studies. I also want to thank my grandmother and cousins. Finally yet importantly, I wish to sincerely thank my husband, Azhar Mohd Khalel and my children, Muhammad Firhan, Lukhman Hakim, Nur Khaisumah, Hilal and Uwais for their love, patience, support and understanding.

CONTENTS

ABSTRACT ii

DEDICATION iv

ACKNOWLEDGEMENT v

DECLARATION vii

CONTENTS viii

LIST OF FIGURES xiv

LIST OF TABLES xvii

ABBREVIATIONS xviii

Chapter 1. Introduction 1

1.1 Motivation 1

1.2 Research objectives 7

1.3 Thesis organisation 8

Chapter 2. Approaches to Control Chart Pattern Recognition 10

2.1 Pattern Recognition 10

2.2 Pattern Recognition Learning Algorithm 11

2.2.1 Unsupervised Learning 13

2.2.2 Supervised Learning 13

2.3 Control Chart Pattern Recognition 15

2.3.1 Control chart patterns simulator 22

2.3.2 Data pre-processing 23

2.3.2.1 Scaling 23

2.3.2.2 Coding in Spiking Networks 24

2.4 Current Trends in Control Chart Pattern Recognition Research 25

2.4.1 Statistical or Traditional Based CCPR 26

2.4.2 Artificial Intelligence (AI) Based CCPR 27

2.4.2.1 Fuzzy sets 27

2.4.2.2 Expert Systems 28

2.4.2.3 Artificial Neural Networks 32

2.5 Spiking Neural Networks 33

2.5.1 Biological Background 33

2.5.2 Neuronal Coding Scheme 38

2.5.3 Rate Codes 38

2.5.3.1 Rate as an Average over Time 39

2.5.3.2 Rate as a Spike Density (Average over Several Repetitions of the Experiment) 41

2.5.3.3 Rate as a Population Activity (Average over Several Neurons) 43

2.5.4 Temporal Coding 43

2.5.4.1 Time-to-first-spike Coding 45

2.5.4.2 Phase Coding 48

2.5.4.3 Correlations and Synchrony 50

2.5.5 Spiking Neuron Model 50

2.6 Optimisation Algorithm 52

2.7 Summary 53

Chapter 3. Spiking Learning Vector Quantisation (S-LVQ) 54

3.1 Preliminaries 54

3.2 LVQ Network Structure 55

3.3 The LVQ Algorithm 57

3.4 Learning Procedure in Standard LVQ Networks 58

3.5 Variants of LVQ Learning Procedures 59

3.5.1 LVQ2 59

3.5.2 LVQ with a Conscience 63

3.5.3 LVQ-X 64

3.6 Discussion 67

3.7 Current Trends in Spiking Neural Networks Research 69

3.7.1 Typical Spiking Neural Networks Architecture 69

3.7.2 A Review of Existing SNNs Learning Procedure 73

3.7.2.1 SNNs for Supervised Learning Procedure 73

3.7.2.1.1 Error Gradient Based Learning Procedures 74

3.7.2.1.2 Hebbian-based Supervised Learning procedures 75

3.7.2.2 SNNs for Unsupervised Learning Procedure 76

3.7.2.2.1 Weight-based Learning 76

3.7.2.2.2 Delay-based Learning 78

3.8 Discussion of SNNs 81

3.9 Motivation for Research 83

3.10 Proposed S-LVQ Algorithm 86

3.10.1 Network Structure 86

3.10.2 S-LVQ Learning Procedure 91

3.10.2.1 Boosting 93

3.10.2.1 Motivating 94

3.11 Setting the Weights, Delays and Threshold 100

3.12 Data Set 100

3.13 Empirical Evaluation of S-LVQ 101

3.14 Comparison with LVQ and its Variants 103

3.15 The Effect of Number of Hidden Neurons on S-LVQ 103

3.16 Summary 106

Chapter 4. Enhanced S-LVQ Network (NS-LVQ) 107

4.1 Previous Work 107

4.2 Motivation for Research 109

4.3 NS-LVQ Networks Architecture 110

4.4 Setting the Weights, Delays and Threshold 114

4.5 Pre-Process Weights 117

4.6 NS-LVQ Learning Procedure 118

4.7 Data Set 124

4.8 Empirical Evaluation of NS-LVQ 125

4.9 Comparison with S-LVQ and Traditional NNs 127

4.10 Learning Parameter (η) 128

4.10.1 Static Learning Rate 128

4.10.2 Adaptive Learning Rate 128

4.10.3 Static Vs Adaptive Learning Rate 132

4.11 Summary 132

Chapter 5. Optimisation of Spiking Neural Networks Using the Bees Algorithm 134

5.1 Preliminaries 134

5.2 Intelligent Swarm-based Optimisation Algorithms (SOAs) 135

5.3 The Basic Bees Algorithm 137

5.3.1 Honey Bees in Nature 137

5.3.2 Bees Algorithm 139

5.3.3 Characteristics of Bees Algorithm 139

5.4 Bees in Artificial Neural Network 143

5.5 Evolution Strategy (ES) in SNNs 147

5.6 Motivation for Research 148

5.7 Spiking Neural Networks with Proposed Bees Algorithm 149

5.7.1 Networks Structure 149

5.7.2 Optimising the Networks 150

5.7.3 Proposed Bees Algorithm 153

5.7.4 Spiking Networks Training Procedure 157

5.7.5 The Proposed Bees Algorithm Parameters 158

5.8 Data Set 158

5.9 Empirical Evaluation of Spiking Networks with Proposed Bees Algorithm 160

5.9.1 Comparison with Spiking Network without Bees Algorithm 162

5.10 Summary 162

Chapter 6 Conclusions and Future Work **164**

6.1 Contributions 165

6.2 Conclusions 169

6.3 Future Work 172

Appendix A 174

Appendix B 205

Appendix C 208

References 210

LIST OF FIGURES

Chapter 2

Figure 2.1 : A framework for pattern recognition

Figure 2.2 : A typical control chart indicating the process is in statistical control

Figure 2.3 : A typical control chart indicating the process is out of statistical control.

Figure 2.4 : Six main classes of control charts.

Figure 2.5 : Main components of an expert system

Figure 2.6 : Action potential in the visual cortex of a monkey

Figure 2.7 : Biological neuron

Figure 2.8 : Structure of a nerve

Figure 2.9 (a) : Definition of the mean firing rate via a temporal average

Figure 2.9 (b) : Gain function, schematic. The output rate ν is given as a function of the total input

Figure 2.10 : Definition of the spike density in the Per-Stimulus-Time Histogram (PSTH) as an average over several runs of the experiment

Figure 2.11 (a): A postsynaptic neuron receives spike input from the population m with activity A_m.

Figure 2.11(b): The population activity is defined as the fraction of neurons that are active in a short interval $[t, t + \Delta t]$ divided by Δt.

Figure 2.12 : Time to first spike

Figure 2.13 : Phase coding

Figure 2.14 : Correlation / synchrony coding

Chapter 3

Figure 3.1 : Standard learning vector quantisation network structure

Figure 3.2 : Flowchart for LVQ procedure

Figure 3.3 : Geometric representation of LVQ2 procedure

Figure 3.4 : Feed forward spiking neural network

Figure 3.5 : The proposed S-LVQ network structure

Figure 3.6 : Multi-synapse terminals for the S-LVQ spiking neural network

Figure 3.7 : A pseudo-code description for S-LVQ Algorithm

Figure 3.8 : A graph showing the result of the different pattern recognisers

Figure 3.9 : The classification accuracy for different number of hidden neurons

Figure 3.10 : The implementation of the proposed S-LVQ algorithm for control chart pattern recognition

Chapter 4

Figure 4.1 : A structure proposed for the NS-LVQ network

Figure 4.2 : Multi-synapse terminals for the NS-LVQ network

Figure 4.3 : A pseudo-code description for pre-process weight

Figure 4.4 : A pseudo-code description for NS-LVQ Algorithm

Figure 4.5 : Exponential decay function

Figure 4.6 : Linear decay function

Figure 4.7 : Adaptive vs. static learning for classification accuracy

Chapter 5

Figure 5.1 : Flowchart of the Bees Algorithm

Figure.5.2 : Graphical illustration of the Bees Algorithm

Figure 5.3 : Single synapse connection between two neurons for the proposed spiking neural network with the Bees Algorithm

Figure 5.4 : Pseudo-code of the proposed Bees Algorithm

Figure 5.5 : Graph illustrating the shrinking method for points that are near to the peak

Figure 5.6 : Classification accuracy of different pattern recognisers

LIST OF TABLES

Chapter 3

Table 3.1 : Performance of various LVQ pattern recognisers

Table 3.2 : Results of handwritten digit recognition

Table 3.3 : Details of the proposed S-LVQ network used for control charts

Table 3.4 : Representation of the output categories

Table 3.5 : Results of different pattern recognisers applied to control chart data set.

Table 3.6 : The effect of the number of hidden neuron

Chapter 4

Table 4.1 : Details of the proposed NS-LVQ network used for control charts

Table 4.2 : Representation of the output categories

Table 4.3 : Results of different pattern recognisers applied to control chart data set.

Table 4.4 : Testing accuracy for different values of static learning rate

Table 4.5 : Comparison of training accuracy based on two different types of adaptive learning rate.

Chapter 5

Table 5.1 : The parameters of the Bees Algorithm for LVQ, RBF and MLP for control chart pattern recognition

Table 5.2 : Performance of different pattern recognisers with Bees

Table 5.3 : Details of the proposed S-LVQ network used for control charts with the Bees Algorithm

Table 5.4 : Details of the proposed NS-LVQ network used for control charts with the Bees Algorithm

Table 5.5 : The parameters of the proposed Bees Algorithm

Table 5.6 : Results of different pattern recognisers

ABBREVIATIONS

SPC	: Statistical Process Control
DOE	: Design of Experiment
AS	: Acceptance Sampling
AI	: Artificial Intelligent
NN	: Neural Network
ANNs	: Artificial Neural Networks
LVQ	: Learning Vector Quantisation
S-LVQ	: Spiking Learning Vector Quantisation
NS-LVQ	: Enhanced- Spiking Learning Vector Quantisation
MLP	: Multi-Layer Perceptron
RBF	: Radial Basis Function
ART	: Adaptive Resonance Theory
SOM	: Self-Organising Maps
KNN	: k-Nearest Neighbour
CCPR	: Control Chart Pattern Recognition
ARL	: Average Run Length
CL	: Centre Line
UCL	: Upper Control Line
LCL	: Lower Control Line
X chart	: Control Chart for Individual
MR-chart	: Moving Range Chart
$\overline{X}$-chart	: Average Chart
R-chart	: Range Chart

σ -chart : Standard Deviation Chart

CUSUM : Cumulative Sum Chart

EWMA : Exponentially Weighted Moving Average ().

SNNs : Spiking Neural Networks

SRM : Spike Response Model

LIFN : Leakey Integrate and Fire Model

PSTH : Per-Stimulus-Time Histogram

PSP : Post-Synaptic Potential

WTA : Winner-Takes-All

SOAs : Swarm-Based Optimisation Algorithms

SI : Swarm Intelligence

ACO : Ant Colony Optimisation

GA : Genetic Algorithm

PSO : Particle Swarm Optimisation

BA : Bees Algorithm

ES : Evolutionary Strategies

CHAPTER 1

INTRODUCTION

1.1 Motivation

Quality and productivity are two essential factors for survival in a global economy experiencing tremendous developments in information technology. The quality of a product can be evaluated in several ways. It is often very important to differentiate these different dimensions of quality. Garvin (1987) provides an excellent discussion of eight components or dimensions of quality. These are as follows:

1) Performance- will the product do the intended job?

2) Reliability- how often does the product fail?

3) Durability- how long does the product last?

4) Serviceability- how easy is it to repair the product?

5) Aesthetics- what does the product look like?

6) Features- what does the product do?

7) Perceived quality- what is the reputation of the company or its product?

8) Conformance to Standards- is the product made exactly as the designer intended?

In this research, only the performance dimension will be addressed. Efficient process control is a key element in the maintenance and improvement of quality and productivity.

There are three major areas in statistical methods for quality control and improvement:

1) Statistical Process Control (SPC);

2) Design of Experiment (DOE);

3) Acceptance Sampling (AS).

This research specifically focuses on SPC. Statistical Process Control is a traditional technique to improve the quality of products, reduce rework, and scrap so that the quality and productivity expectations can be met. SPC primarily involves the implementation of control charts, which are used to detect any change in a process that may affect the quality of the output. Among the eight dimensions of quality, the performance dimension for control charts has been chosen as the focus. Control charts have been the most popular and widely used charts in industry for providing the capability for pattern recognition or pattern classification. Their applications have now moved far beyond manufacturing into engineering, environmental science, biology, genetics, epidemiology, medicine, finance, and even law enforcement and athletics [Lai, 1995; Montgomery, 1997; Ryan 2000].

The first control charts were developed by Shewhart in the 1920s [Shewhart 1931]. These simple Shewhart charts have dominated applications to date. Recently, control chart pattern recognition has received considerable attention in the literature, including applications to syntactic approaches, fuzzy-expert systems and artificial neural network models. Today's manufacturing enterprises need to adopt modern tools of quality engineering to maintain and improve their competitiveness in the

marketplace. One way to improve control chart procedures is to replace the SPC specialist with computers, which are able to mimic human-like intelligent behaviour. Although other improved control chart varieties give more powerful detection ability, such as the combined Shewhart-CUSUM scheme, they still have limitations. First, they still lack a pattern recognition capability [Guo and Dooley, 1993; Cheng 1995]. A review suggests that pattern recognition research has increased in importance as driven by the need for rapid interpretation and quick response to process deterioration within advanced manufacturing environments [Hwang and Hubele, 1993; Guh et al., 1999a]. Piplani and Hubele [Piplani and Hubele, 2001] noted that research into this area is relatively young. Second, the needs for robustness of control chart performance to violations of assumptions in control charting is increasing as in realistic situations data are auto-correlated. Third, as manufacturing complexity and uncertainty increase, SPC procedures become more demanding. There is a shortage of good SPC specialists as the skills required to implement proper control chart procedures develop over time, making use of the accumulated knowledge of the processes involved. In addition, a specialist's skills may vary from one machine, or plant, to another and involve human factors regarding learning ability, attitude and decision making aptitude. Fourth, the systems do not take effect automatically.

Recently, attention has focused on artificial intelligence (AI), a branch of computer science, which has shown great promise in dealing with difficult manufacturing problems. What makes AI techniques popular is their ability to learn from experience and to handle uncertain, imprecise (fuzzy) and complex information in a competitive environment that demands high quality. Among the available AI tools, neural networks is one of those which has attracted the most attention from researchers and

practitioners for the solving of many control chart issues such as pattern recognition. A clear definition of artificial neural networks (ANNs) is given by Hecht-Nielsen, [Hecht-Nielsen ,1990] in Pandya & Macy [Pandya & Macy ,1995].

> "*A neural network is a parallel, distributed information processing structure consisting of processing elements (which can possess a local memory and can carry out localized information processing operations) interconnected via unidirectional signal channels called connections. Each processing element has a single output*".

ANNs enhance this work by capturing automatically more meaning from the limited number of measurements that originally were collected for traditional control charts. The desired characteristics of a real-time SPC system in a highly automated and integrated manufacturing environment are accurate representation of the process without oversimplification and adaptability to new changes [Jacobs and Luke, 1993]. Previous researchers [Hwang, 1992; Pham and Oztemel, 1994; Cheng, 1997; Spoerre and Velasco, 2001] proposed ANNs as a potential solution to SPC pattern recognition problems.

A number of researchers have demonstrated the utility of neural networks in identifying process non-randomness, such as shifts, cycles or trends, in quality control charts. Neural network techniques have greatly extended and enhanced traditional approaches. One promising aspect of neural network pattern recognition is that the system simultaneously can be trained to identify a variety of unnatural patterns (Guh and Tannock 1999). The application of ANN to SPC can also be beneficial when prior knowledge about the probability distribution of the process data is unknown. ANN

has the ability to extract regularities in datasets without any a priori assumptions if the available data is enough for efficient training.

The new generation of artificial neural networks has attracted research efforts from the domains of artificial intelligence and pattern recognition because they offer the prospect of describing much better the actual output of a biological neuron. Networks of spiking neurons (SNNs) are the third generation of neural network models. The different generations of ANNs based on neural network computational units can be defined as follows.

The first generation employs McCulloch-Pitts neurons as computational units. These are also referred to as perceptrons or threshold-gates. They give rise to a variety of neural network models such as multi-layer perceptrons, Hopfield nets, and Boltzmann machines. A characteristic feature of these models is that they can only give digital output. In fact, they are universal for computations with digital input and output, and every Boolean function can be computed by some multi-layer perceptron with a single hidden layer.

The second generation is based on computational units that apply to a weighted sum (or polynomial) of the inputs an "activation function" with a continuous set of possible output values, such as the sigmoid function or the linear saturated function. Typical examples for networks from this second generation are feedforward and recurrent sigmoidal neural nets, as well as networks of radial basis function units. The characteristic features of these models are that they can compute arbitrary Boolean functions; furthermore, they can compute certain Boolean functions with fewer gates than neural networks from the first generation. In addition, neural networks from this second generation are able to compute functions with analogue input and output. In

fact, they are universal for analogue computations in the sense that any continuous function with a compact domain and range can be well approximated by a network of this type with a single hidden layer. The second generation also support learning algorithms, which are based on gradient descent, such as back propagation.

The third generation incorporates spiking neurons (or "integrate and fire neurons") as computational units. There exist a number of variations of this model, which are described and compared in a recent survey [Gerstner, 1995]. Spiking neuron models are high level models in which biological neurons are considered as homogeneous processing units. Models for spiking neurons based on temporal coding are the Spike Response Model (SRM), and the Leakey Integrate and Fire Model (LIFN) [Maass, 1997a]. This research adapts the Spike Response Model (SRM) as the model [Bialek, Rieke, de Ruyter and Warland, 1991]. SNNs have a similar architecture to traditional neural networks. SNNs use spikes as the basis for information processing and are based on temporal coding. The characteristic features of SNNs are firstly that they are substantially more realistic as compared with the previous two models. In particular, they describe much better the actual output of a biological neuron, and hence they allow us to investigate on a theoretical level the possibilities for using time as a resource for computation and communication. Secondly, the timing of individual or single spikes plays the key-role in both computation and communication in SNNs. Thirdly, the output of all the spiking neurons is spikes of the same dimension. In other words, the output of a spiking neuron consists of the set of points at the time when a neuron fires. Fourthly, the timing of a single spike is considered, but not the dimension. Lastly, a spiking neuron can be viewed as a digital or analogue computational element, depending on the type of temporal coding that is used [Maass & Bishop, 2001].

1.2 Research Objectives

The overall aim of this research was to design and develop spiking neural network systems as a powerful pattern recognition tool for control chart data. These systems should be able to recognise patterns in control chart data in an efficient and effective way. Moreover, the systems should be reliable and with a simple architecture. Accordingly, they should be able to achieve superior performance in terms of classification accuracy. To achieve the overall aim of the research, the following objectives were set:

- To perform a detailed analysis of existing spiking neural network techniques for classification learning, with particular emphasis on supervised learning, and to assess their appropriateness for control chart pattern recognition application.

- To develop a simulator to create and to perform a detailed analysis of spiking neuron networks on control chart pattern recognition.

- To improve the overall performance of spiking neuron networks including:

  - To improve the implementation of existing learning algorithms that has been considered as significantly suitable for identified problems.

  - To develop new learning algorithms that are computationally efficient for control chart pattern recognition accuracy.

  - To adopt a simple architecture such as learning vector quantization (LVQ) for spiking neuron networks.

- To develop an effective method of training spiking neural networks using an optimisation algorithm.

1.3 Thesis Organisation

The remainder of the thesis is organised as follows.

Chapter 2 defines the classification learning problem, presents a framework for viewing approaches to it, discussing in some detail spiking neuron networks algorithms and reviews other artificial neural networks approaches. Current trends and recent developments in spiking neural networks research are also presented.

Chapter 3 gives a detailed description of previous work on learning vector quantisation and its application to control chart pattern recognition. This chapter also discusses the potential of spiking neural networks as a pattern recogniser for control charts. A simple network structure similar to that of an LVQ network [Pham and Liu, 1995] is utilised. Accordingly, a new learning algorithm, called spiking learning vector quantisation (S-LVQ), is proposed for control chart pattern recognition. An empirical evaluation of the proposed algorithm is also reported and discussed.

Chapter 4 is an enhancement of the network presented in chapter 3. Based on the S-LVQ network, a simpler network structure is proposed. This chapter describes a new method for establishing pre-process weight and its advantages are discussed. A study of static and adaptive learning parameters is also presented. Finally, the chapter gives the results of experiments carried out to demonstrate the performance of the proposed structure.

Chapter 5 proposes the use of the Bees Algorithm, a new optimisation tool, for training spiking neural networks. The chapter presents a detailed description of the algorithm and its application to the control chart pattern recognition problem.

Chapter 6 summarises the contributions and conclusions of the thesis and proposes directions for further research.

CHAPTER 2

APPROACHES TO CONTROL CHART PATTERN RECOGNITION

2.1 Pattern Recognition

An informal definition for pattern recognition is telling things apart. Otherwise, pattern recognition is the automatic transformation of data X_i (observation, features) into a set of symbols C_i (classes). Pattern recognition (also known as pattern classification) is a field within the area of computer science and can be defined as "the act of taking in raw data and taking an action based on the category of the data" [Duda R.O et al, 2001]. In other words, pattern recognition is a process of extracting information from an unknown data stream or signal and assigning it to one of the prescribed classes or categories [Haykin, 1999]. It uses methods from statistics, machine learning and other areas. Typical applications are automatic speech recognition, classification of text into several categories (e.g. spam or non-spam email messages), the automatic recognition of handwritten postal codes on postal envelopes, or the automatic recognition of images of human faces. The last three examples form the subtopic image analysis of pattern recognition that deals with digital images as input to pattern recognition systems. Pattern recognition techniques include Neural Networks, Hidden Markov Models, and Bayesian Networks. The fundamentals of various aspects of pattern recognition can be found in Theodoridis and Koutroumbas [Theodoridis and Koutroumbas, 2006].

This chapter gives an overview of pattern recognition approaches in general and of control chart pattern recognition (CCPR) specifically. The chapter is organised as follows: section 2 formally defines the pattern recognition learning problem and presents a framework for viewing approaches to it. The framework is presented in Figure 2.1; section 3 describes in more detail control chart pattern recognition as this thesis is concentrated on control chart applications; section 4 reviews current trends and recent developments in control chart pattern recognition research; a review of the spiking neural networks approach in which this thesis is most interested is presented in section 5; section 6 concludes the chapter with a summary of some of the key research issues in CCPR.

2.2 Pattern Recognition Learning Algorithm

Specifically, pattern recognition has three types of learning algorithms. These are unsupervised, supervised and reinforcement learning algorithms. However, reinforcement learning can be regarded as a special form of supervised learning. The detailed descriptions of these learning algorithms will be given in the next subsection. Usually, any given type of network architecture can be employed in any of these three major learning algorithms. The next two subsections describe these three learning algorithms and identify the most suitable application for each algorithm.

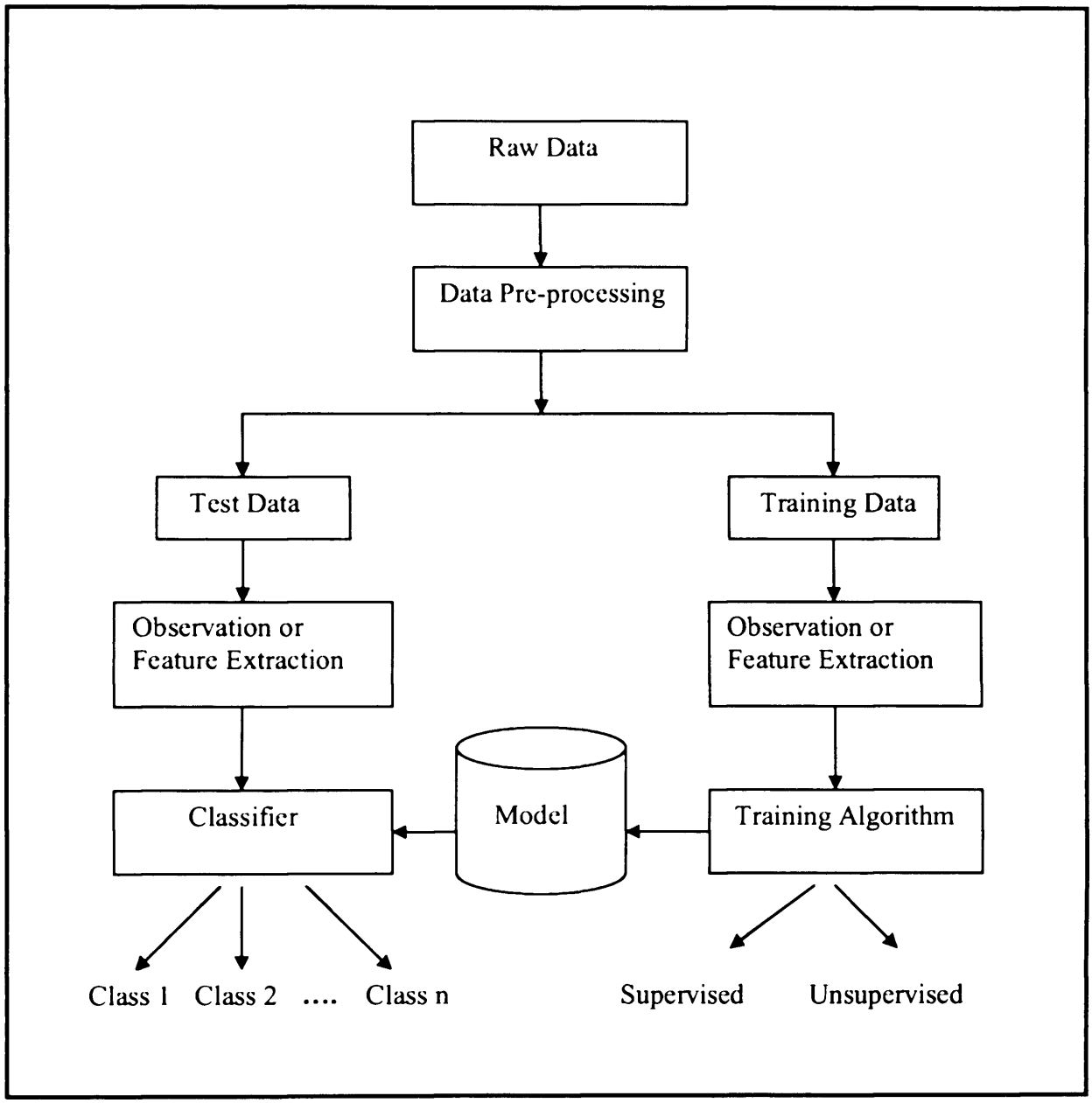

Figure 2.1: A framework for pattern recognition

2.2.1 Unsupervised Learning

In classification learning, a learning algorithm is given a sample of pre-classified examples from the problem domain called the training set. Each example is described by a vector of attributes. An attribute is either nominal or continuous. The algorithm learns a model that is used to predict the class of future examples.

Learning methods can be divided into supervised and unsupervised schemes based on whether or not a dedicated target function for prediction has been defined. In unsupervised methods, such a function is not available and the goal is grouping or clustering instances based on some similarity or distance measure. The unsupervised scheme is more suitable for application where there are insufficient examples.

2.2.2 Supervised Learning

In supervised learning, there is either a nominal or continuous-valued target function to be predicted. The former case is referred to as classification and the latter as regression or continuous prediction. In this thesis, only methods for supervised classification learning will be addressed.

If the examples in the training set are presented and used all at once, learning is said to be batch or off-line. If the examples are presented one at a time, and the concept definition evolves over time as successive examples are incorporated, learning is said to be incremental or on-line. This thesis concentrates on on-line learning. The main goal of a

classification learning system is to produce a classifier that will assign previously-unseen examples (i.e., examples not in the training set) to the corresponding classes with high accuracy. In other words, given a set of example pairs $(x, y), x \in X, y \in Y$, the aim is to find a function f in the allowed class of functions that matches the examples and to *infer* the mapping implied by the data.

The accuracy of a classifier is defined as the probability that it will correctly classify a new, unlabelled example (i.e., examples in a test set). Ideally, given a complete description of an example (i.e., the values of all its attributes), its class should be determined unambiguously. However, in some instances, process data is available every second or every few minutes on hundreds of process variables and product characteristics. Consequently, examples may appear with an erroneous class value, or with erroneous attribute values, or with both. These errors may stem from a diversity of sources, including the limitations of measuring instruments, and human error while typing examples into a computer. All of these phenomena, referred to collectively as noise, limit the achievable accuracy in a pattern recognition or a pattern classification problem. The degree of robustness of a learning system with respect to noise is one of its most important characteristics. It also occurs often in practice that the values of certain attributes for certain example are simply not available. These are called missing values and again a practical control chart pattern recognition system must be able to handle them.

Generally, the applications that use the algorithm of supervised learning are pattern recognition (also known as classification) and regression (also known as function approximation). The supervised learning paradigm is also applicable to sequential data (e.g., for speech and control chart recognition). This thesis concentrates on supervised learning algorithms.

2.3 Control Chart Pattern Recognition

Many quality characteristics can be expressed in terms of a numerical measurement. Examples include dimensions such as length or width, temperature, and volume. Such a quality characteristic that is measured on a numerical scale is called a variable. Control charts for variables are used extensively. The control charts for variable data are: control chart for individual (X); moving range chart (MR-chart); average chart ($\overline{X}$-chart); range chart (R-chart); median chart; standard deviation chart (σ-chart); cumulative sum chart (CUSUM); exponentially weighted moving average (EWMA). Two of the most commonly used control charts in industry for variable data are the X-bar charts and the range charts (R-charts). These two control charts were adopted in this study. The success of quality improvement depends on two major factors:

1) The quality of data available;
2) The effectiveness of the techniques used for analyzing the data.

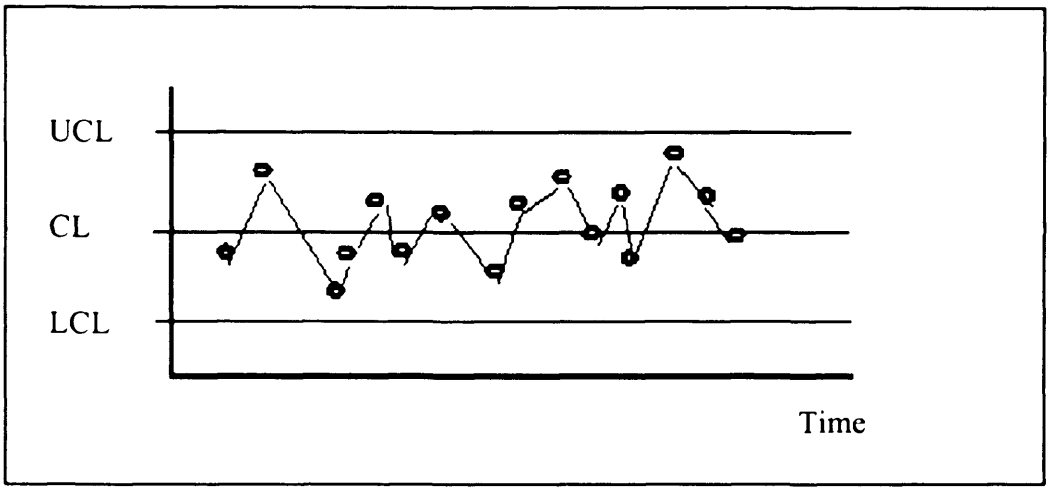

Figure 2.2: A typical control chart; control chart indicates the process is in statistical control.

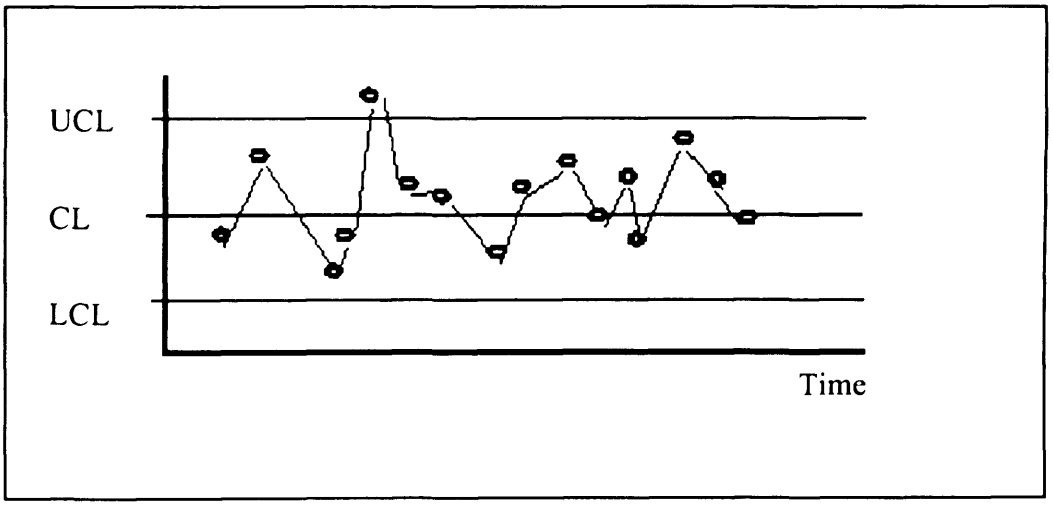

Figure 2.3: A typical control chart; control chart indicates the process is out of statistical control.

Generally, control charts are a graphical display of a quality characteristic that has been measured from a sample versus the sample number or time. The chart contains a centre line (CL) that represents the average value and the upper (UCL) and lower (LCL) lines allow variation limits (natural variation limits) of the quality characteristic under consideration.

Figure 2.2 and Figure 2.3 show a typical control chart for a process in statistical control and a process out of statistical control respectively.

These limits, usually taken as the mean value plus and minus three standard deviations, represent the boundaries of the range for unavoidable (inherent) variations as follows:

$$UCL = \mu + \frac{3\sigma}{\sqrt{n}} \qquad (1)$$

$$LCL = \mu - \frac{3\sigma}{\sqrt{n}} \qquad (2)$$

Three standard deviations are used because there is a high probability (99.73%) that a sample measurement will fall within this range if the process is in control (the quality characteristic is assumed to be normally distributed based on the central limit theorem). Proper construction and interpretation of these charts is very important. Careful construction of the charts and a capability analysis will determine the inherent variation of a process which is in control and capable of producing the products to meet customer specifications. After the charts have been constructed, they are employed for on-line process monitoring.

Control rules are used to detect out-of-control situations considering the very recent history of a process. A bare X-bar chart only indicates when to look for disturbances. It does not indicate where to look for them, or what types of disturbances are present. In order to avoid the occurrence of such situations, instead of waiting for them to happen and finding out afterwards, it is necessary to monitor the long term behaviour of the process. This can be done by observing the patterns contained in the control charts for the process. It is important to detect the out-of-control situation as well as to recognise the shape of an unnatural pattern. The nature of control chart patterns can reveal any impending out-of-control situations.

Thus the problem of monitoring a process to predict possible abnormalities reduces to that of recognizing control chart patterns, which is the subject of this research. Assumptions made for control chart performance are that the data is normally distributed and that the data is independent.

A control chart may indicate an out-of-control condition either when one or more points fall beyond the control limits or when the plotted points exhibit some non-random pattern of behaviour. Certain types of pattern may also indicate an out-of-control condition. For example, consider the control charts in Figure 2.3. Although all points fall within the control limits, the points do not indicate statistical control because their pattern is very non-random in appearance.

Such patterns may indicate a problem with the process, such as operator fatigue, raw material deliveries, and so forth. Although the process is not really out of control, the yield may be improved by elimination or by reduction of the sources of variability causing those patterns.

The problem is one of pattern recognition. That is, of recognizing systematic or non random patterns on the control chart and identifying the reason for this behaviour. In many cases, the pattern of the plotted points will provide useful diagnostic information on the process, and this information can be used to make process modifications that reduce variability (the goal of SPC). There are six main classes of patterns in control charts, normal, cycle, upward trend, downward trend, upward shift, and downward shift, as illustrated in Figure 2.4.

Specifically, control chart pattern recognition is a process of recognising an unknown CCP and assigning it to one of the prescribed classes. CCP includes previous data, and does not rely merely on the last data. Normally, patterns of the same category share common properties.

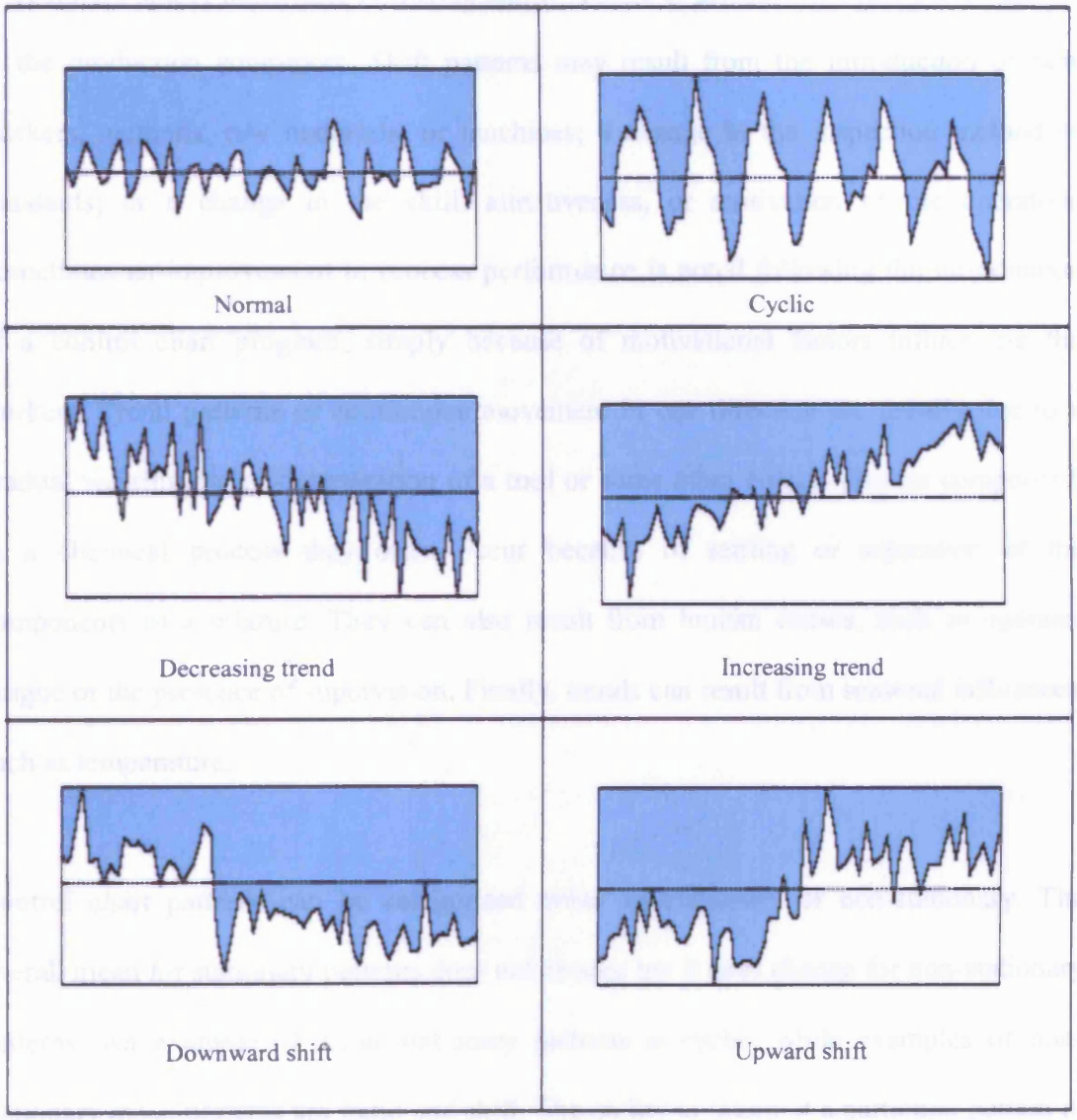

Figure 2.4: Six main classes of control charts.

Cyclic patterns occasionally appear on the control chart. Cyclic patterns may result from systematic environmental changes such as temperature, operator fatigue, regular rotation of operators and/or machines, or fluctuation in voltage or pressure or some other variable in the production equipment. Shift patterns may result from the introduction of new workers, methods, raw materials, or machines; a change in the inspection method or standards; or a change in the skill, attentiveness, or motivation of the operators. Sometimes an improvement in process performance is noted following the introduction of a control chart program, simply because of motivational factors influencing the workers. Trend patterns or continuous movement in one direction are usually due to a gradual wearing out or deterioration of a tool or some other critical process component. In a chemical process they often occur because of settling or separation of the components of a mixture. They can also result from human causes, such as operator fatigue or the presence of supervision. Finally, trends can result from seasonal influences, such as temperature.

Control chart patterns can be categorised either as stationary or non-stationary. The overall mean for stationary patterns does not change but it does change for non-stationary patterns. An example of mean stationary patterns is cyclic, while examples of non-stationary mean patterns are trend and shift. The ability to interpret a particular pattern in terms of assignable causes requires experience and knowledge of the process. That is, one must not only know the statistical principles of control charts, but also have a good understanding of the process.

2.3.1 Control chart patterns simulator

The following expressions were used to generate the different patterns for a control chart. This data set is used in this thesis. The total number of generated patterns is 1500. The training set employed 1002, and the testing set 498.

1. Normal patterns:

$$y(t) = \mu + r(t)\,\sigma \tag{3}$$

2. Cyclic patterns:

$$y(t) = \mu + r(t)\,\sigma + a\sin(2\pi\,t/T) \tag{4}$$

3. Increasing or decreasing trends:

$$y(t) = \mu + r(t)\,\sigma \pm gt \tag{5}$$

4. Upward or downward shifts:

$$y(t) = \mu + r(t)\,\sigma \pm ks \tag{6}$$

where

μ = mean value of the process variable being monitored

σ = standard deviation of the process

a = amplitude of cyclic variations (taken as 15 or less)

g = magnitude of the gradient of the trend (taken as being in the range 0.2 to 0.5)

k = parameter determining the shift position (k=0 before the shift position; k=1 at the shift position and thereafter).

r = normally distributed random number (between -3 and 3)

s = magnitude of the shift (taken as being in the range of 7.5 to 20)

t = discrete time at which the pattern is sampled (taken as being within the range 0 to 59).

T = period of a cycle (taken as being in the range 4 to 12 sampling intervals).

$y(t)$ = sample value at time t.

This pattern simulator is taken from Pham and Oztemel [Pham and Oztemel, 1994].

2.3.2 Data pre-processing

In data pre-processing, the most important transformation techniques are 1) standardization; 2) zoning; 3) scaling and 4) using continuous (as opposed to binary) representation. In this thesis, before the data were presented to the networks, two steps of data pre-processing were implemented: 1) scaling and 2) coding in spiking networks.

2.3.2.1 Scaling

Although the input data to any node can theoretically take any value, restricting it to fall within a fixed range produces more efficient training. Scaling is a transformation that is devised according to each individual application to modify the input data into a fixed range. The most important issue in scaling is the range of output values dictated by the scaling transformation. Different types of scaling transformation may operate over different ranges of values [Zorriassatine and Tannock, 1998]. There are two advantages of scaling described by Swingler [Swingler, 1996]. Firstly, it takes care of the distribution of the training data and the effect of outliers, and secondly, it ensures that errors or variations of different variables contribute the same proportion to the change in network weights. In this thesis, by applying a scaling method mentioned below, the original inputs

were scaled to fall as continuous values between 0 and 1. The actual data sets were scaled values of $y(t)$. Scaling was performed using the following expression:

$$\bar{y}(t) = \frac{y(t) - y_{min}}{y_{max} - y_{min}} \qquad (7)$$

where

$\bar{y}$ = scaled pattern value (in the range 0 to 1)

y_{min}= minimum allowed value (taken as 35)

y_{max}= maximum allowed value (taken as 125)

This scaling method is taken from Pham and Oztemel [Pham and Oztemel, 1994] with some modification on the minimum and maximum allowed value.

2.3.2.2 Coding in Spiking Networks

In traditional neural networks, the essential information is encoded in the firing rates, which are averaged over time. Unlike the traditional neural networks, spiking neural networks use the timing of single spikes generated by a neuron to encode information. The scheme of coding used here is called temporal coding. It utilizes the timing of individual spikes. More details concerning temporal coding are given in section 5. The scaled input data will then be mapped over a number of time steps, referred to as the input time window. The coding resolution affects the performance of the network. Increasing the number of steps in the input time window will increase the precision. However, this will decrease the computational efficiency of the model. The precision of the temporal code should therefore be selected in such a way as to attain adequate

accuracy with optimal computational efficiency. In this thesis, experiments for control chart data set revealed that an input time window with 100 units is adequate. A simple linear mapping scheme is as follows:

Input time window = (100 - (100 * (scaled data))) unit.

2.4 Current Trends in Control Chart Pattern Recognition Research

Control chart pattern recognition research has been making significant progress in many directions. A review by Hwang and Hubele and also Guh [Hwang and Hubele, 1993; Guh et al., 1999a] noted that this area of research has increased in importance, driven by the need for rapid interpretation and quick response to process deterioration within advanced manufacturing environments. Piplani and Hubele [Piplani and Hubele, 2001] claimed that research into this area is relatively young. This section examines two of the most popular directions that have major impact on CCPR and discusses some current problems. The two directions are statistical or traditional based CCPR and automated and intelligent CCPR.

2.4.1 Statistical or Traditional Based CCPR

Generally, statistical based CCPR can be divided into four types: i) syntactic or structural matching, ii) template matching, iii) statistical testing, and iv) heuristic algorithm. Pham and Wani [Pham and Wani, 1997] applied heuristic techniques in their work on feature-based control chart pattern recognition. They reported here that the manual process of

obtaining a good set of heuristics is extremely laborious. Cheng [Cheng, 1989] proposed syntactic pattern recognition. Cheng and Hubele [Cheng and Hubele, 1996] proposed a mathematical pattern recognition algorithm drawn from syntactic (structural) and statistical (discriminate) approach. The problem is that the application of such an algorithm would require trial and error experiments to define the parameters for the algorithm and their sensitivities to the patterns. Generally, traditional techniques for control chart pattern recognition rely on assumptions requiring prior process knowledge.

2.4.2 Artificial Intelligence (AI) Based CCPR

Recently, research issues have been closely related to advances in pattern recognition technology. AI in pattern recognition has attracted a lot of research interest in time series data sequencing, especially in control chart problems. Artificial intelligence is a science that has defined its goal as concerned with designing intelligent computer systems, that is, making machines do things that would require intelligence if done by humans – understanding language, learning, reasoning, solving problems, and so on. Various AI techniques have been implemented for application in control chart pattern recognition including fuzzy sets, expert systems, and neural networks.

2.4.2.1 Fuzzy sets

The fuzzy set theory approach is a powerful means of representing and handling uncertainty (imprecise information) and is particularly useful when an inexpensive solution is sought [Schalkoff, 1997]. It was pioneered by Lotfi Zadeh in 1965. The use of

fuzzy rules provides a way of exploiting the tolerance for imprecision to achieve tractability, robustness, and a low solution cost. Kahraman [Kahraman et al., 1995] reported the use of triangular fuzzy numbers in the tests for unnatural SPC patterns. No results on the proposed method were given. Chang and Aw [Chang and Aw, 1996] developed a neural fuzzy control chart for detecting and classifying process mean shifts. Recently, Wang and Rowlands [Wang and Rowlands, 1999; 2000] explored the use of fuzzy logic to represent zones in the control chart for detecting runs. Their results confirmed the feasibility of the technique for control chart interpretation.

2.4.2.2 Expert Systems

An expert system is a system that employs human knowledge captured by a computer to solve problems that ordinarily require human expertise [Turban, 1995]. The heart of expert systems is the domain knowledge (knowledge about a particular problem or situation). Therefore, expert systems are also referred to as knowledge-based systems. In an expert system, the domain knowledge is usually represented in two forms: it is either at the level of know-how where underlying fundamentals are not detailed (shallow knowledge); or at a level where its theoretical and scientific fundamentals are deeply expressed (deep knowledge). There are several ways of representing either type of knowledge in an expert system. The three most popular methods are rules, frames, and semantic networks.

The four main components of an expert system are:

(i) Knowledge base. This contains knowledge about the problem domain. It can comprise rules, rule sets, frames, classes, and procedures.

(ii) Inference engine. This manipulates the stored knowledge to produce solutions to problems. The inference engine in a rule-based expert system scans the knowledge base, selecting and applying appropriate rules. Inference can proceed in different ways according to different control procedures.

(iii) User interface or explanation module. This handles communication with the user in a "natural" language. A set of general facilities to be provided by a user interface module is documented by Zahedi [Zahedi, 1990].

(iv) Knowledge acquisition. This assists with the development of the knowledge base by facilitating the capture and encoding of the domain knowledge. The main principles and strategies for knowledge acquisition may be found in Cullen and Bryman [Cullen and Bryman, 1988]. The main components of an expert system are illustrated in Figure 2.5. Expert systems have been applied in SPC to automate control chart selection, construction, and analysis.

Some of the traditional methods for CCPR have been implemented using the expert system technique. Swift [Swift, 1987] has described a knowledge-based control chart pattern recogniser. The system employs a statistical hypothesis and is designed for off-line use. A drawback of the system is that it assumes an in-control state always follows an out-of-control state whereas, in practice, once a process has gone out-of-control, it is unlikely to return to an in-control state without corrective intervention.

Similar systems have been reported for control chart pattern recognition using templates [Cheng, 1989; Cheng and Hubele, 1989] or control theory [Love and Simaan, 1989; Simaan and Love, 1990] instead of statistical hypotheses. Pham and Oztemel [Pham and Oztemel, 1992a] have described an on-line control chart pattern recogniser utilising heuristic rules and statistical hypothesis. Swift and Mize [Swift and Mize, 1995] used statistical significance tests as interpretative rules to determine the pattern variation. Generally, they reported promising results, and noted the feasibility of the expert system for control chart pattern recognition.

Researchers have shown that expert systems are a powerful tool for knowledge gathering, knowledge retrieval, and decision making. Some drawbacks of the system are its limited use for pattern recognition, particularly in a dynamic environment, and that it is time consuming to train the expert systems with all possible patterns. Furthermore, as mentioned above, expert systems require human experts to provide all the possible rules. This may create difficulty when recognising patterns that have not been encountered previously.

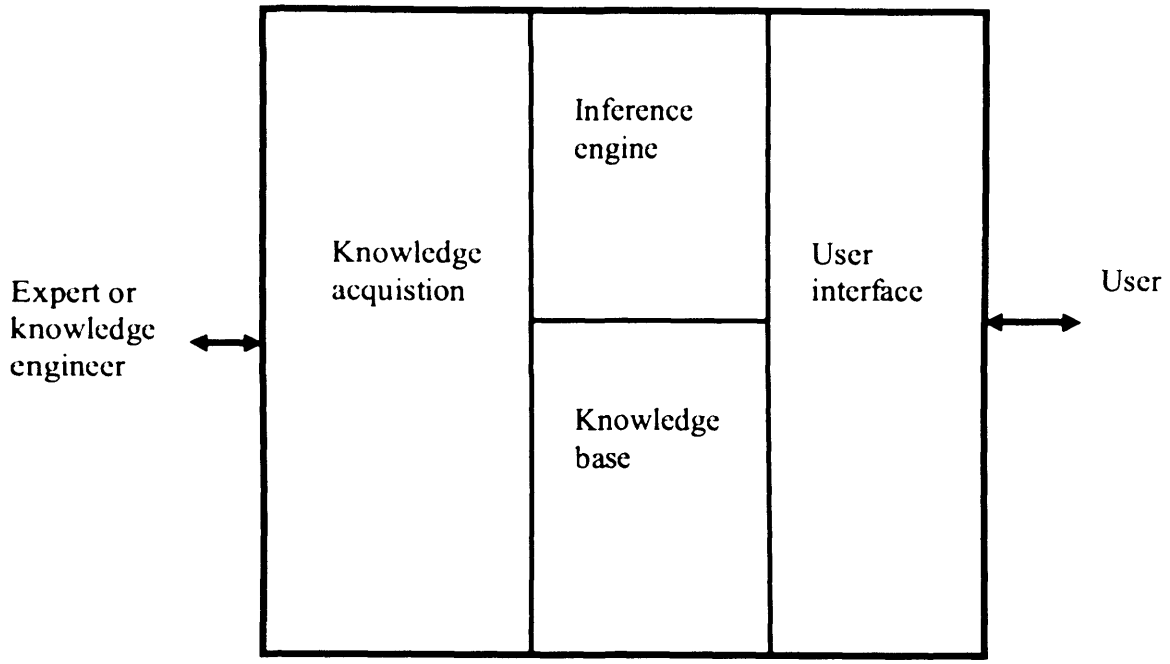

Figure 2.5: Main components of an expert system

The inflexibility of expert systems has limited their effectiveness in recognising control chart patterns, particularly within changing and dynamic manufacturing environments. Artificial neural networks were found to be a promising tool to overcome this limitation.

2.4.2.3 Artificial Neural Networks

The pattern recognition and classification capabilities of neural networks have been shown to be better than those of traditional techniques [Lippmann, 1989]. Neural networks first appeared in the late 1980s. An artificial neural network is a massively parallel-distributed processor that has the ability to learn, recall, and generalise knowledge [Haykin, 1999]. A great deal of research in neural networks for pattern recognition has focused on classification learning, the main aim of which is to increase the accuracy of correct classification. Neural network-based pattern recognisers perform identification and classification with minimum process knowledge, requiring only examples of how different patterns are classified. Such pattern recognisers are able to generalise from given examples. This enables arbitrary patterns to be readily classified.

Sutton, Pham and Zhang [Sutton, 1992; Pham and Oztemel, 1994; Zhang et al., 1995] provide more detailed information regarding the application of NNs to manufacturing in general. The principle reason for applying NNs to SPC is to automate SPC chart interpretation. Accurate representation of the process without oversimplification, and adaptability to new changes, were among the features highlighted by Jacobs and Luke [Jacobs and Luke, 1993] as the desired characteristics of a real-time SPC system within a

highly automated and integrated manufacturing environment. NNs can potentially satisfy these requirements.

Researchers have applied various examples of NN architecture to pattern recognition. Existing popular NN architectures are: Multi-Layer Perceptron (MLP), Radial Basis Function (RBF), Learning Vector Quantization (LVQ), Adaptive Resonance Theory (ART), Auto-Associative NNS, and Kohonen Self-Organising Maps (SOM). As with NN architecture, there are also many rules for NN learning. Hwang and Hubele [Hwang and Hubele, 1993a, 1993b] have applied NNs with Back-propagation architecture. They used the Average Run Length (ARL) as the performance criterion. Pham and Oztemel [Pham and Oztemel, 1993a, 1993b] applied BPN with a hybrid structure and used classification accuracy (%) as the performance criterion. In 1994, Pham and Oztemel [Pham and Oztemel, 1994] applied the structure of LVQ-X. They used classification accuracy as the performance criterion. Hwang and Chong [Hwang and Chong, 1995] used ART1 mod architecture, but they used modified ARL as the performance criterion. Yang and Yang [Yang and Yang, 2002] proposed a new supervised LVQ for control charts based on a fuzzy-soft competitive learning network. They used classification accuracy as the performance criterion. Generally, all the researchers reported promising results. Among the existing NN architectures, LVQ structures have a very simple architecture. In this thesis, LVQ structures will be of most interest and will be discussed in detail in Chapter 3.

2.5 Spiking Neural Networks

Experimental evidence from the past few years indicates that many biological neural systems use the timing of single spikes (temporal coding) for very rapid speed information processing. It is considered that the timing of the first spike contains most of the relevant information needed for processing. As a result, very recently, researchers' attention has shifted to spiking neurons. This research is concerned with spiking neuron networks as the ANN technique for control chart pattern recognition. Spiking neuron networks have a similar architecture to traditional neural networks, have spiking neurons as processing units, transmit information by spike (pulses), and use spikes as the basis for information processing.

Spiking neural networks are networks of spiking neurons, which represent an entirely new generation of artificial neurons. The next subsection introduces SNNs, including the biological background, coding scheme, and neuron models.

2.5.1 Biological Background

Research from the past hundred years has shown that the brain is comprised of neurons. The most pertinent structures in neurons are axons, dendrites, the cell body, and synapses. The axons carry signals away from the cell body to other neurons. A neuron receives connections from thousands other neurons. Most of these contacts take place on the neuron dendrites trees; however they can also exist on the soma or the axon of the

neuron. The morphology of the dendrites tree plays an important role in the integration of the synaptic inputs and it influences the way the neuron processes information and computes [Mel, 1993]. The strengths of the charges received by a neuron on its dendrites are added together through a nonlinear process of spatial and temporal summation [Koch, 1999]. The dendrites receive stimuli and carry it to the cell body. The cell body is separated from the surrounding medium by a selectively permeable membrane. There is an electric potential which is also known as action potential associated with the concentration of charged ions inside the cell. When the cell receives a signal, the signal may cause it to either increase or decrease the potential.

If the action potential exceeds a certain threshold, the neuron fires, sending signals to every other neuron to which it is connected through a synapse. Synapses play an important role in neuronal information processing. Immediately after a neuron fires, its potential is drastically lowered, which prevents it from repeatedly firing in some circumstances. Figure 2.6, Figure 2.7 and 2.8 show an action potential exceeding the threshold θ, a biological neuron, and the structure of a nerve respectively.

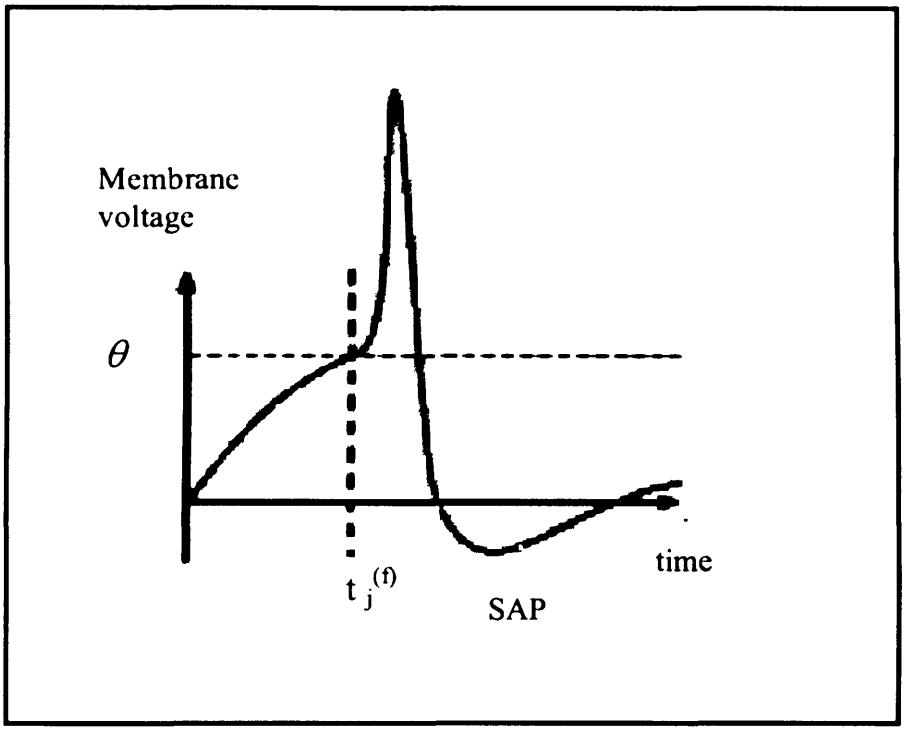

Figure 2.6: Action potential in the visual cortex of a monkey

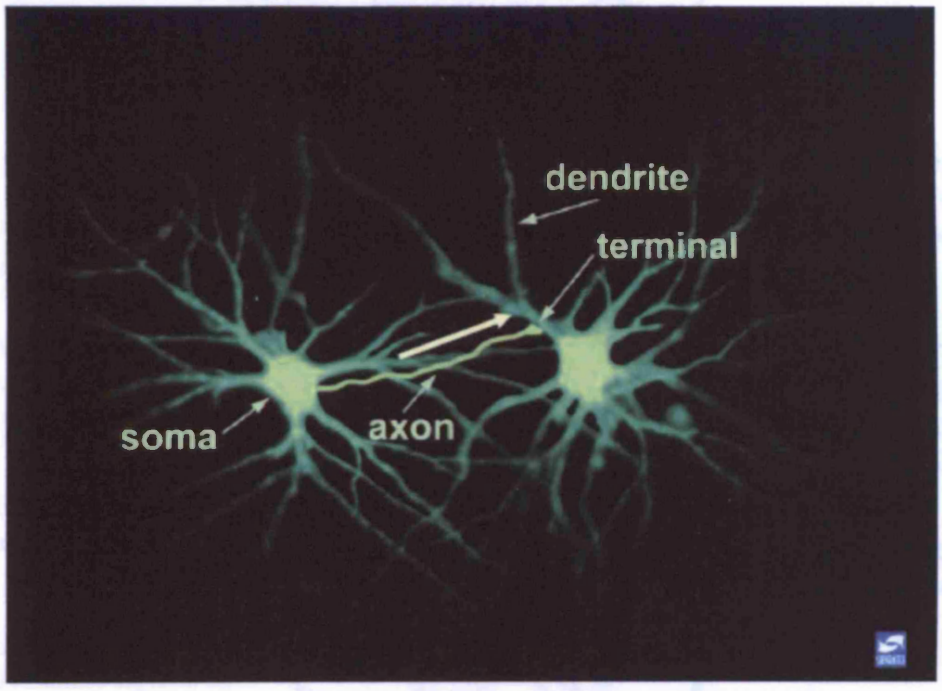

Figure 2.7: Biological neuron

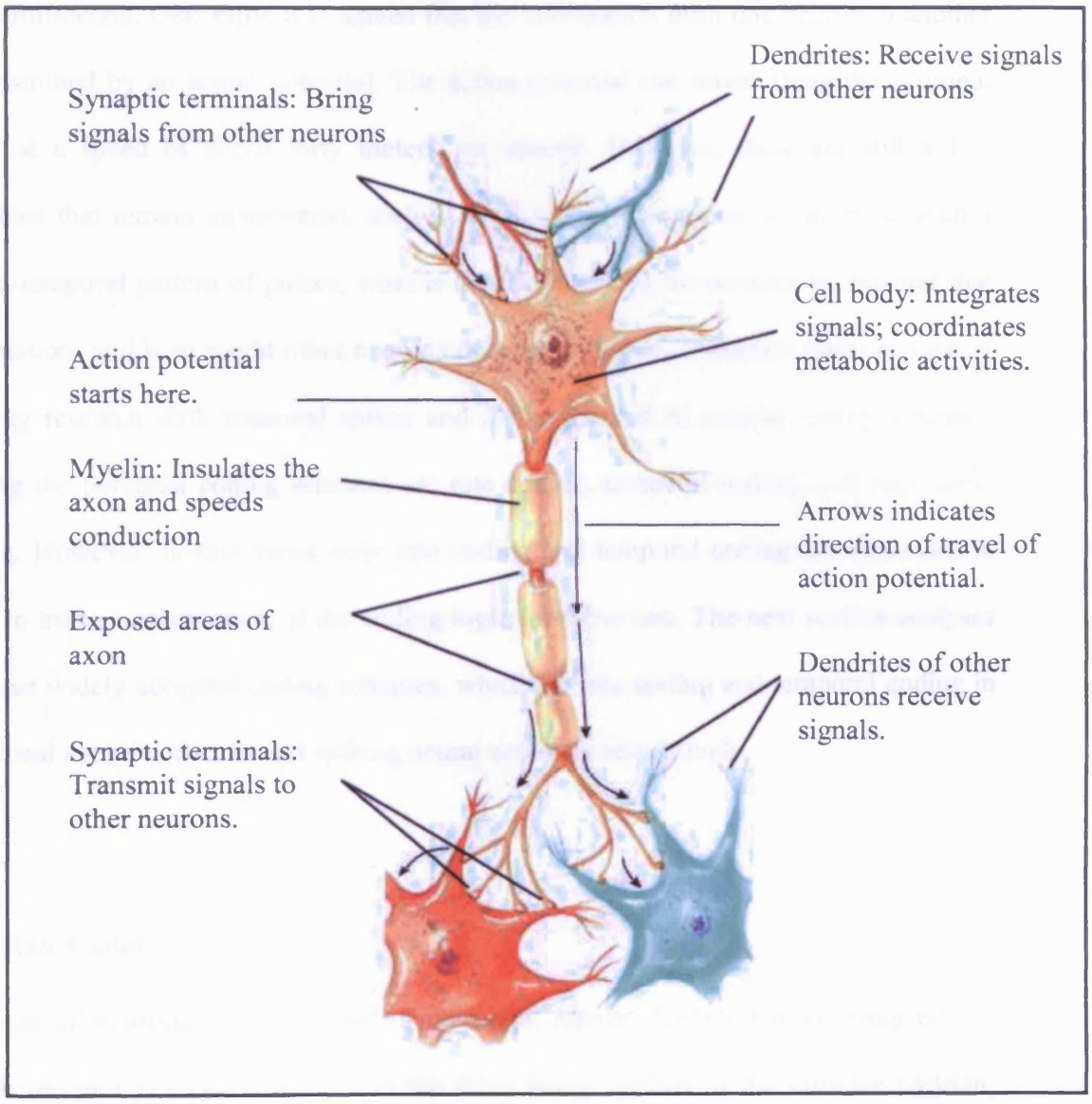

Figure 2.8: Structure of a nerve

2.5.2 Neuronal Coding Scheme

The mammalian brain contains more than 10^{10} densely packed neurons that are connected to an intricate network. In every small volumes of cortex, thousands of spikes are emitted each millisecond. Generally, it is agreed that the information from one neuron to another is transmitted by an action potential. The action potential can travel along the neuronal fibres at a speed of about forty meters per second. However, there are still a few questions that remain unanswered, such as what is the information contained in such a spatial-temporal pattern of pulses, what is the code used by the neurons to transmit that information, and how might other neurons decode the signal. Therefore, there is a lot of ongoing research with neuronal spikes and it has resulted in several coding schemes. Among the potential coding schemes are rate coding, temporal coding, and population coding. However, in this thesis only rate coding and temporal coding are discussed in order to make a comparison of the coding logic for these two. The next section analyses the most widely accepted coding schemes, which are rate coding and temporal coding in traditional neural networks and spiking neural networks respectively.

2.5.3 Rate Codes

In a seminal contribution more than 75 years ago, Adrian showed that the firing rate of stretch receptor neurons is related to the force being applied to the muscles [Adrian, 1926]. As a result, early neural network models interpreted the output of artificial neurons as an abstraction of the firing rate or rate coding in their biological counterparts. In a general way, rate coding is transferring information by means of the firing rate of a

neuron. There are three definitions of rate coding which refer to three different averaging procedures: 1) An average over time. 2) An average over several repetitions of the experiment. 3) An average over a population of neurons.

2.5.3.1 Rate as an Average over Time

This is the first and most commonly used definition of a firing rate, referred to as a temporal average. Rate as a spike count is essentially the spike count in an interval of duration T divided by T. Figure 2.9 (a) and (b) illustrates this coding. The length T of the time window is set by the experimenter and depends on the type of neuron from which it records and on the stimulus. In practice, to get sensible averages, several spikes should occur within the time window. This definition of rate has been successfully used in many research activities, particularly in experiments on sensory or motor systems. A classical example is the experiment on a stretch receptor in a muscle spindle [Adrian, 1926].

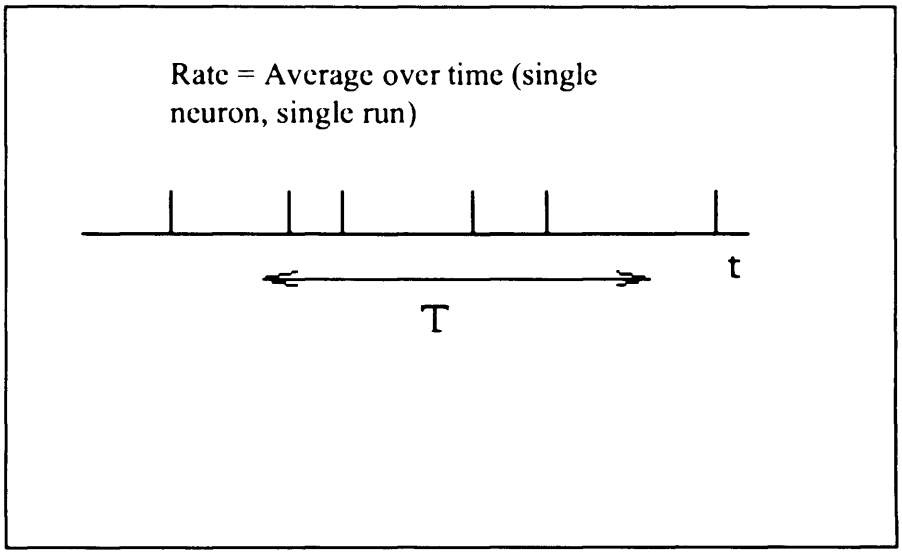

Figure 2.9 (a): Definition of the mean firing rate via a temporal average.

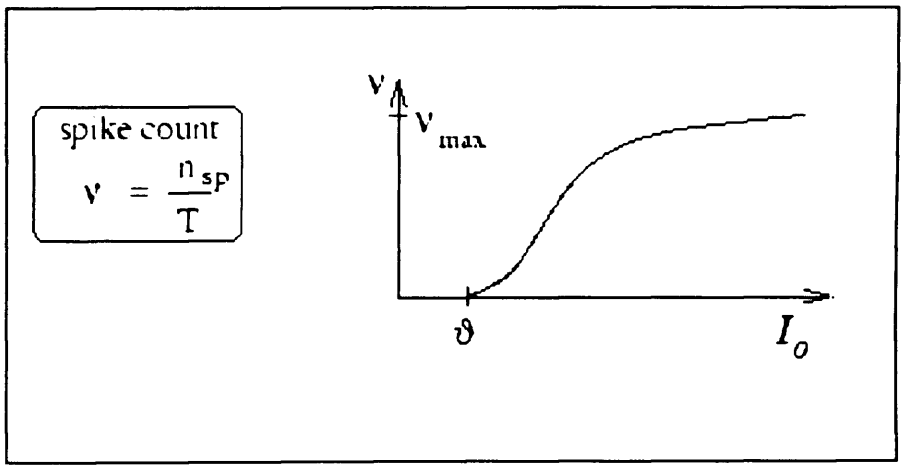

Figure 2.9 (b): Gain function, schematic. The output rate ν is given as a function of the total input.

2.5.3.2 Rate as a Spike Density (Average over Several Repetitions of the Experiment)

This is a coding rate based on the average of spikes over several observations with the same stimulation. The same stimulation sequence is repeated several times and the neuronal response is reported in a Per-Stimulus-Time Histogram (PSTH). Figure 2.10 shows the PSTH. The time t is measured with respect to the start of the stimulation sequence and Δt is typically in the range of one or a few milliseconds. The spike density measure is a useful method for evaluating neuronal activity, particularly in the case of time-dependent stimuli.

The obvious problem with this approach is that it cannot be the decoding scheme used by neurons in the brain. Consider, for example, a frog which wants to catch a fly. It cannot wait for the insect to fly repeatedly along exactly the same trajectory. The frog has to base its decision on single 'run'.Each fly, and each trajectory, is different.

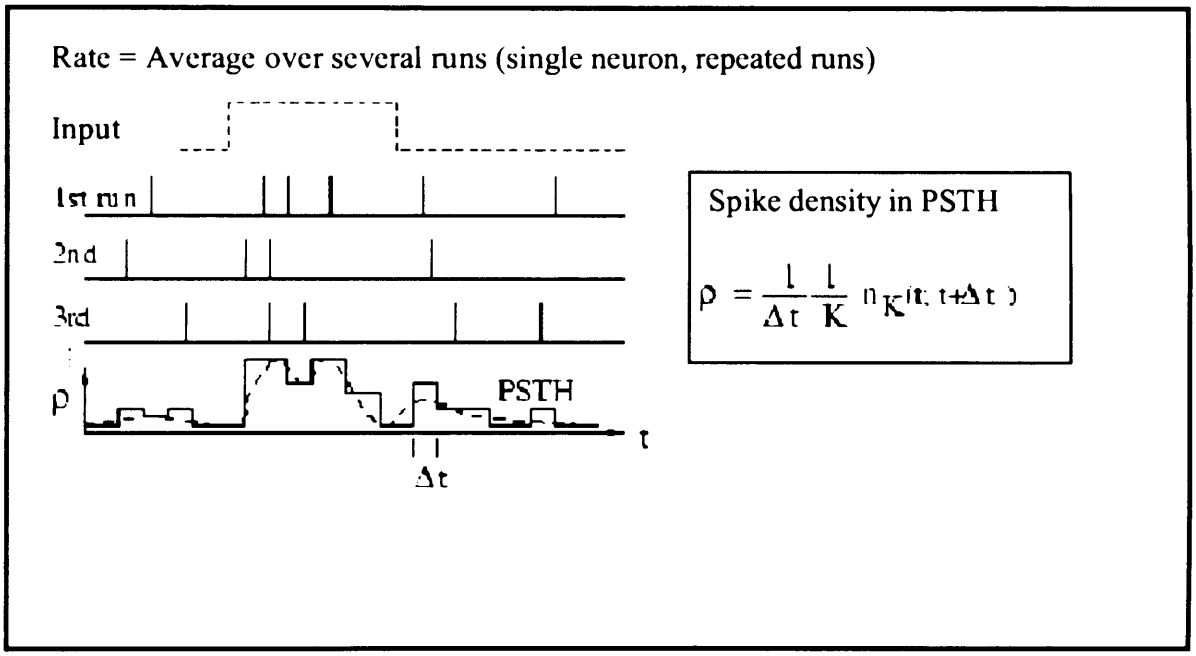

Figure 2.10: Definition of the spike density in the Per-Stimulus-Time Histogram (PSTH) as an average over several runs of the experiment.

2.5.3.3 Rate as a Population Activity (Average over Several Neurons)

The number of neurons in the brain is huge. The brain often has many neurons with similar properties which respond to the same stimuli. For example, neurons in the primary visual cortex of cats and monkeys are arranged in columns of cells with similar properties [Hubel, 1988; Hubel and Wiesel, 1962]. In particular, all neurons in the population should have the same pattern of input and output connections. Figure 2.11(a) and (b) show the population activity.

A potential problem with this coding is that it formally requires a homogeneous population of neurons with identical connections, which is hardly realistic. Real populations will always have a certain degree of heterogeneity both in their internal parameters and in their connectivity patterns. Rate as a population activity may, however, be a useful coding principle in many areas of the brain.

2.5.4 Temporal Coding

The classical point of view that neurons transmit information exclusively via modulations of their mean firing rates [Shadlen and Newsome, 1998; Mazurek and Shadlen, 2002; Litvak et al., 2003] seems to be at odds with the growing empirical evidence that neurons can generate spike-timing patterns with millisecond temporal precision *in vivo* [Chang et al., 2000; Tetko and Villa, 2001] and *in vitro* [Mao et al., 2001; Ikegaya et al., 2004].

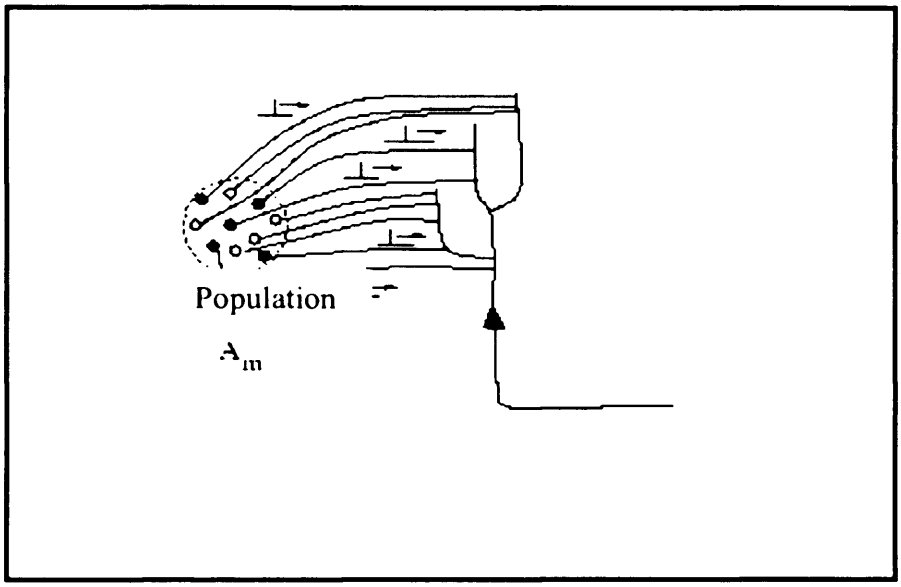

Figure 2.11(a): A postsynaptic neuron receives spike input from the population m with activity A_m.

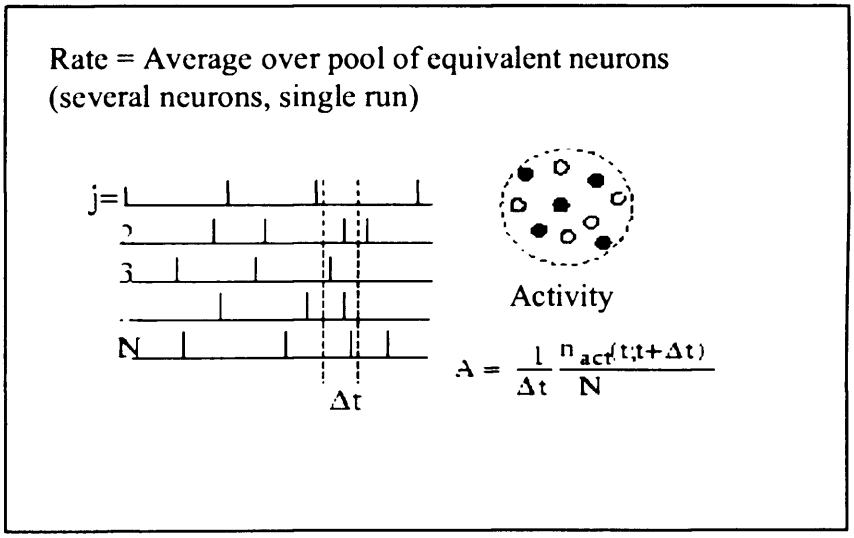

Figure 2.11(b): The population activity is defined as the fraction of neurons that are active in a short interval $[t, t + \Delta t]$ divided by Δt.

Patterns can be found in the firing sequences of single neurons [Reinagel and Reid, 2002] or in the relative timing of spikes of multiple neurons [Chang et al., 2000] forming a functional neuronal group [Edelman, 1993]. Activation of such a neuronal group can be triggered by stimuli or by behavioural events [Villa et al., 1999; Riehle et al., 1997]. In temporal coding, information is transmitted by the timing of each spike. These findings have been widely used to support the hypothesis of "temporal coding" in the brain [Abeles, 2002; Diesmann et al. 1999]. Within temporal coding, several variations exist by considering the relations between spikes and other neurons. The main consideration in this relationship is whether or not the individual action potentials and individual neurons encode independently, or if the correlations between different spikes from the same or several neurons carry significant information.

2.5.4.1 Time-to-first-spike Coding

Time-to-first-spike is a potential coding strategy that is based on temporal coding which counts only the first spike of each neuron. In this thesis, first spike was chosen as the coding strategy as it is reliable and easy to implement. Moreover, there are few discussions from previous research that support the idea of using first spike.

Firstly, in a realistic situation, it is quite common that a neuron may abruptly receive a new input at time t_0. This means that a neuron might be driven by an external stimulus which is suddenly switched on at time t_0. Consider the following situation which happens in the retina. When someone looks at a picture, their gaze jumps from one point to the next.

After each saccade, there is a new visual input at the photo receptors in the retina. Information about the time t_0 of a saccade would easily be available in the brain. Then imagine a code where for each neuron the timing of the first spike to follow t_0 contains all information about the new stimulus. Hence, all following spikes would be irrelevant. Alternatively, each neuron emits exactly one spike per saccade and is shut off by inhibitory input afterwards. The time gap between a reference signal and the first spike is enough to pass the information. It is also clear that, in such a scenario, only the timing conveys information and not the number of spikes. This coding strategy is certainly an idealization which formally analyzed by Wolfgang Maass [Maass, 1997b].

Secondly, in a slightly different context, coding by first spikes has also been discussed by S. Thorpe [Thorpe et al., 1996]. He argues that the brain does not have time to evaluate more than one spike from each neuron per processing step. Therefore, the first spike should contain most of the relevant information. Using information-theoretic measures on their experimental data, several groups have shown that most of the information about a new stimulus is indeed conveyed during the first 20 or 50 milliseconds after the onset of the neuronal response [Optican and Richmond, 1987; Kjaer et al., 1994; Tovee et al., 1993; Tovee and Rolls, 1995].Figure 2.12 shows time-to-first-spike coding strategy.

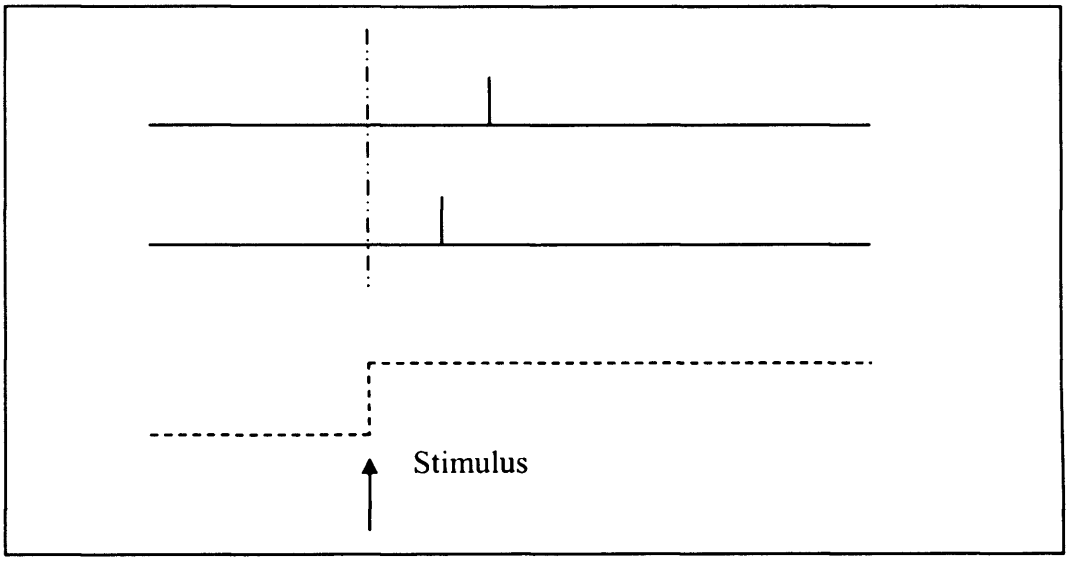

Figure 2.12: Time to first spike

An early spike, which will result in a small time gap, could signal a strong stimulation, and a later spike would signal a weaker stimulation. A coding scheme based on the time to first spike is certainly an idealization and simple. These advantages mean that it is applied widely in analytical studies.

2.5.4.2 Phase Coding

A coding scheme with time to first spike codes the information by means of the time gap between a neuron spike and a static reference signal. Compared to time-to-first-spike coding, a periodic signal is used as reference in phase coding. Oscillations of some global variables such as the population activity are quite common in some areas of the brain (hippocampus, olfactory). These oscillations could serve as an internal reference signal for coding purposes. The concept of coding by phases has been studied by several different groups, not only in model studies [Hopfield, 1995], but also experimentally [O'Keefe and Recce, 1993]. There is evidence that the phase of a spike during an oscillation in the hippocampus of the rat conveys information on the spatial location of the animal which is not accounted for by the firing rate of the neuron alone. Figure 2.12 shows phase coding.

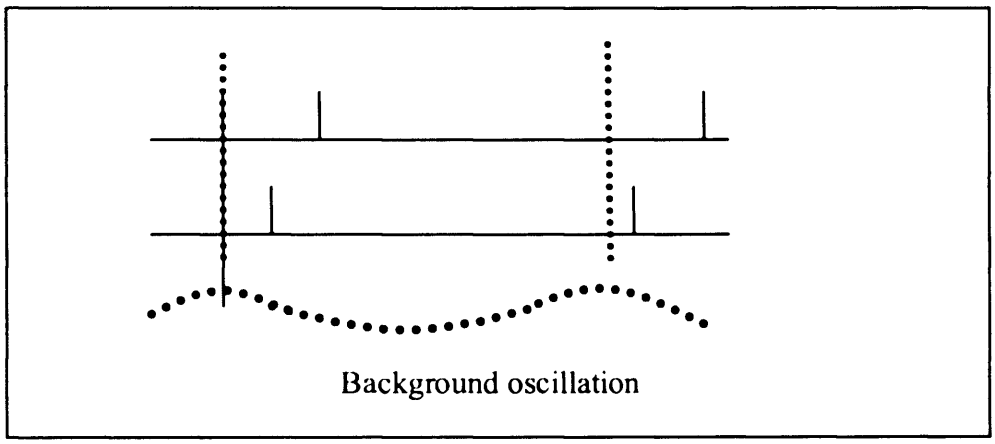

Figure 2.13: Phase coding

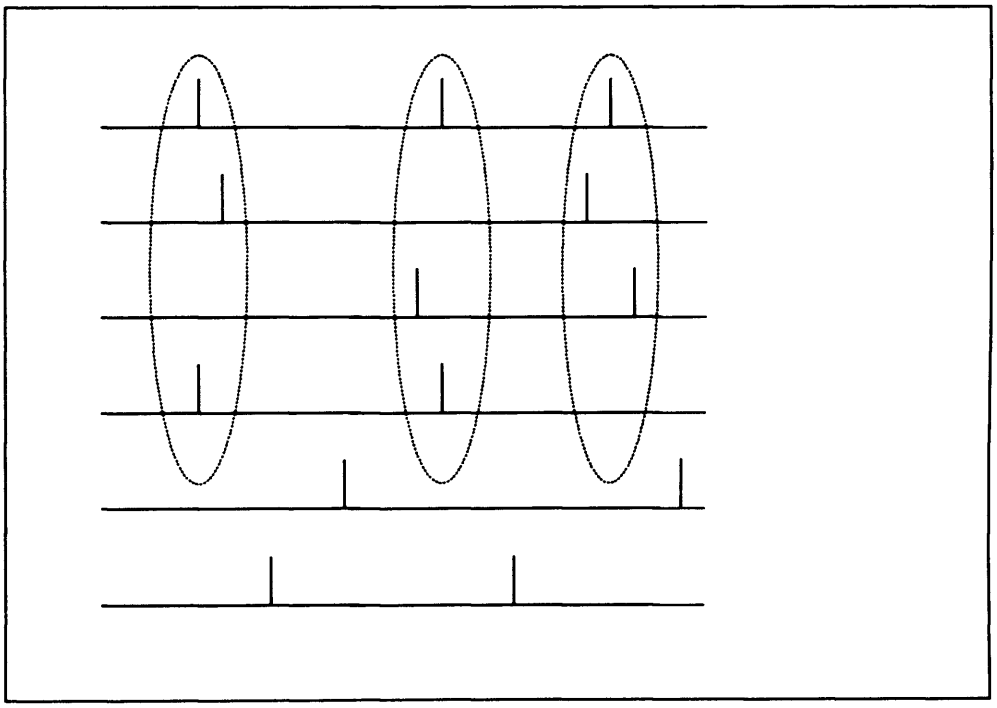

Figure 2.14: Correlation / synchrony coding

2.5.4.3 Correlations and Synchrony

Correlations and synchrony coding use spikes from other neurons as the reference signal for a pulse code. Synchrony between a pair and a group of neurons could signify special events and convey information which is not contained in the firing rate of the neurons. One famous idea is that synchrony could mean 'belonging together' [Milner, 1974; Malsburg, 1981]. Consider for example a complex scene consisting of several objects. It is represented in the brain by the activity of a large number of neurons. Neurons which represent the same object could be 'labelled' by the fact that they fire synchronously. Figure 2.14 shows correlations and synchrony coding.

2.5.5 Spiking Neuron Model

There are several models which describe the neuronal activity in the brain for various level abstractions. Spiking neuron models are high level models in which biological neurons are considered as homogeneous processing units. There are two models for spiking neurons based on temporal coding. First, the Spike Response Model (SRM) [Gerstner and Van Hemmen, 1994]. Second, the Leaky Integrate-and-Fire Model (LIFM), [Maass, 1997a]. However, this research will adopt SRM only. The study of spiking neural networks as the tool for pattern recognition is mainly motivated by the desire to develop more realistic neuron models and to automate the systems, as the SNNs have more computational power [Maass, 1996] in comparison with the traditional neural network model. Consequently, the system should give higher accuracy in pattern recognition or classification.

SRM is basically a generalised leaky integrate-and-fire model. The leaky integrate-and-fire model describes the biophysical mechanisms of the neuron mainly by means of its membrane potential. In addition, this model gives much importance to the time lag from the last firing event. The SRM model is the basis for the SNN proposed in this thesis. The model describes the state of a neuron j at time t by the state variable $u_j(t)$ [Maass, 2001a]. Let F_i be the inputs the neuron j receives from pre-synaptic neurons $i \in \Gamma_j$, where $\Gamma_j = \{i \mid \text{pre-synaptic to } j\}$. In general, pre-synaptic neurons are the input spikes data and j neurons are basically the hidden neurons or called post-synaptic neurons. In a typical network, a neuron would have several pre-synaptic neurons and each could present several input spikes. The effect of an input spike given at $t_i^{(f)}$ to the neuron j at time t, $(t > t_i^{(f)})$ will be $w_{ji}\varepsilon_{ji}\left(t - t_i^{(f)}\right)$, where ε_{ji} is called the spike response function. The input spikes can either increase or decrease the state variable u_j. An output spike will be generated when u_j exceeds a threshold value θ at some time t. In this thesis, t is the simulated time. All the previous output spikes of neuron j will be F_j where $F_j = \{\left(t_j^{(f)}\right) 1 \le f \le n\} = \{t \mid u_j(t) = \theta \text{ and } u_j'(t) > 0\}$ at a particular time t. Here, $t_j^{(f)}$ is the time where the state variable u_j crosses the threshold value from below. A neuron potential (referred to post-synaptic neuron) will be set to a very low value immediately after a particular firing event. This phenomenon will return the neuron potential to its normal state after a significant amount of time. This is known as the refractoriness of a neuron. This state variable u_j (t) can be defined with equation:

$$u_j(t) = \sum_{t_j^{(f)} \in} \eta_j\left(t - t_i^{(\ )}\right) + \sum_{i \in \Gamma_j} \sum_{t_i^{(f)} \in} w_{ji} \varepsilon_{ji}\left(t - t_i^{(f)}\right)$$

where η_j is the function to reflect the refractoriness of the neuron j, the strength of the connection between the neuron i and j is represented by w_{ji} and ε_{ji} represent the spike response function, which can be either excitatory or inhibitory. An excitatory spike response function will increase the potential of the receiving neuron while an inhibitory spike response function with negative effect will decrease the potential of the receiving neuron.

2.6 Optimisation Algorithm

Many complex multi-variable optimisation problems cannot be solved exactly within polynomial bounded computation times. This has generated interest in search algorithms that find near-optimal solutions in reasonable running times. The algorithm described in this paper is a search algorithm capable of locating good solutions efficiently. The algorithm is inspired by the food foraging behaviour of honey bees and could be regarded as belonging to the category of "intelligent" optimisation tools [Pham et al., 2006]. This thesis proposes a modified version of the basic Bees Algorithm, specifically for determining the neighbourhood range and presents an application of the new algorithm to spiking neural networks. The algorithm is also among the first applications to control chart pattern recognition. The aim of applying the Bees Algorithm in the proposed networks here is for optimising the network topology to the size of the network, the time

needed for the optimisation and the classification accuracy. This algorithm is presented in detail in chapter 5.

2.7 Summary

This chapter has given background information on pattern recognition and learning algorithms with attention focused on control chart pattern recognition. The basic concepts of control chart pattern recognition have been described and the three main types of learning algorithms available have been presented. This chapter has also outlined a number of algorithms of each type and discussed their performance for control chart pattern recognition. Finally, recent directions in research approaches have been presented.

CHAPTER 3

S-LVQ: A SPIKING LEARNING VECTOR QUANTISATION ALGORITHM

3.1 Preliminaries

Learning vector quantisation (LVQ) networks as originally proposed by Kohonen [Kohonen, 1984] are known good neural classifiers which provide fast and accurate results for many applications. LVQ is a widely used approach to classification. It is applied in a variety of practical problem areas including medical image and data analysis, for example in speech recognition and in control chart pattern recognition. This is a supervised version of vector quantization. Classes are predefined and the data are labelled. The goal is to determine a set of prototypes that best represent each class. In vector quantization, it is assumed that there is a codebook which is defined by a set of M prototype vectors. (M is chosen by the user and the initial prototype vectors are chosen arbitrarily).

The first part of this chapter, from section 3.2 to section 3.6, is organised as follows: section 3.2 introduces the general structure of LVQ networks; a review of LVQ algorithm is then given; this is followed by a detailed description of the existing learning procedure in LVQ networks including the standard LVQ and variants of its learning procedures; finally, section 3.6 outlines the control chart pattern recognition problem using LVQ and previous work addressing it.

3.2 LVQ Network Structure

An LVQ network comprises three layers of neurons: an input buffer layer, a hidden layer, and an output layer. The structure of an LVQ is shown in Figure 3.1. The input layer carries out no information processing and only conveys the input patterns to the network. The hidden layer (also known as the Kohonen layer) performs actual information processing. The output layer yields the category of the input pattern. The network is fully connected between the input and hidden layers and partially connected between the hidden and output layers. Each output neuron is linked to a different cluster of hidden neurons. The hidden layers to output layer connections have their values fixed to 1. The weights of the connections between the input and hidden layers constitute the components of the reference vectors (one reference vector is assigned to each hidden neuron).

The reference vectors' values are modified during the training of the network. Both the hidden neurons and the output neurons have binary outputs. When a Kohonen neuron wins the competition, it is turned 'on' (its activation value is made equal to 1) while others are automatically switched 'off' (their activation values are set to 0). This, in turn, makes the output neuron connected to the activated Kohonen neuron or to the cluster of hidden neurons that contains the winning neuron switch 'on' (emits a '1') and the rest switch 'off' (emits a '0'). The output neuron that produces a '1' gives the class of the input pattern. Each output neuron is dedicated to a different class.

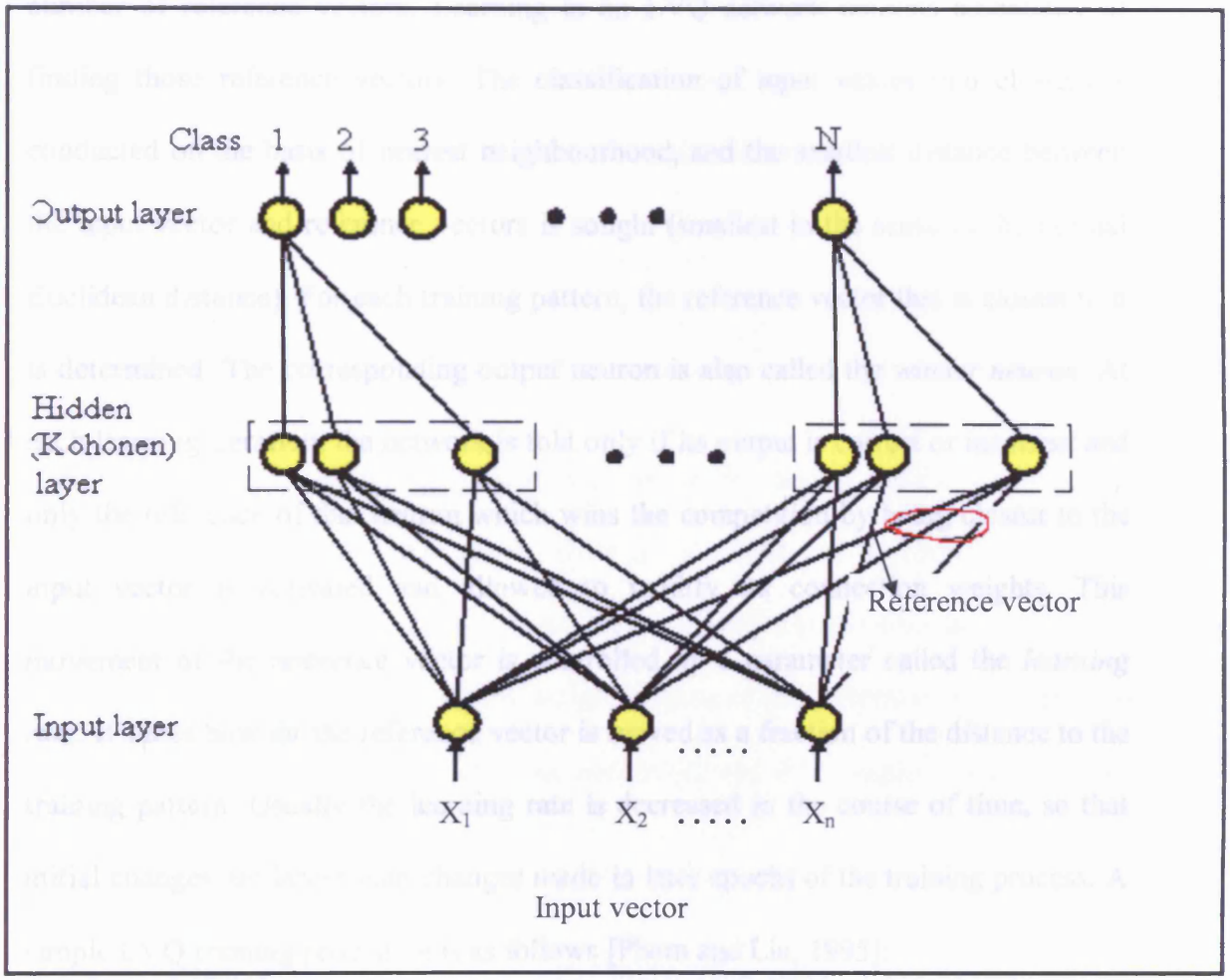

Figure 3.1: Standard learning vector quantisation network structure

3.3 The LVQ Algorithm

LVQ, as its name indicates, is based on vector quantisation, which is the mapping of an n-dimensional vector into one belonging to a finite set of representative vectors. That is, vector quantisation involves clustering input samples around a predetermined number of reference vectors. Learning in an LVQ network consists essentially of finding those reference vectors. The classification of input values into clusters is conducted on the basis of nearest neighbourhood, and the smallest distance between the input vector and reference vectors is sought (smallest in the sense of the normal Euclidean distance). For each training pattern, the reference vector that is closest to it is determined. The corresponding output neuron is also called the *winner neuron*. At each learning iteration, the network is told only if its output is correct or incorrect and only the reference of that neuron which wins the competition by being closest to the input vector is activated and allowed to modify its connection weights. This movement of the reference vector is controlled by a parameter called the *learning rate*. It states how far the reference vector is moved as a fraction of the distance to the training pattern. Usually the learning rate is decreased in the course of time, so that initial changes are larger than changes made in later epochs of the training process. A simple LVQ training procedure is as follows [Pham and Liu, 1995]:

(i) Initialise the weights of the reference vectors;

(ii) Present a training input pattern to the network;

(iii) Calculate the (Euclidean) distance between the input pattern and each reference vector;

(iv) Update the weight of the reference vector that is closest to the input pattern, that is, the reference vector of the winning hidden neuron. If the latter belongs to the cluster connected to the output neuron in the class that the input pattern is known to belong to, the reference vector is moved closer to the input pattern. Otherwise, the reference vector is moved away from the input pattern;

(v) Return to (ii) with a new training input pattern and repeat the procedure until all training patterns are correctly classified (or a stopping criterion is met).

3.4 Learning Procedure in Standard LVQ networks

Good performance in an LVQ network depends on the correct number of reference vectors being assigned to each category, their initial values, and the choice of a proper learning rate and stopping criterion. In general, the Euclidean distance is adopted as a basic rule of competition between the weight vectors of the reference vectors and the input vector. The distance d_i between the weight vectors W_i of neuron i and the input vector X is given by:

$$d_i = \| W_i - X \| = \sqrt{(\sum_j (W_{ij} - X_j)^2)} \qquad (8)$$

Where W_{ij} and X_j are the j th components of W_i and X, respectively. As mentioned in section 3.3, the neuron which has the minimum distance wins the competition and is permitted to change its connection weights in each learning iteration. The learning formula for updating the reference vector is given as follows:

If the winning neuron is in the correct category, then;

$$W_{new} = W_{old} + \lambda(X - W_{old}) \quad (9)$$

and if the winning neuron is in the incorrect category, then;

$$W_{new} = W_{old} - \lambda(X - W_{old}) \quad (10)$$

In equations (9) and (10), λ is the learning rate (usually, $0<\lambda<1$), which decreases monotonically with the number of iterations. The implication of the learning rule expressed in equations (9) and (10) is that the weight or reference vector is updated to be close to the input vector if it represents the input pattern, and is pushed away if it does not. Figure 3.2 summarised the features of a standard LVQ network.

3.5 Variants of LVQ Learning Procedures

3.5.1 LVQ2

LVQ2 was also developed by Kohonen [Kohonen et al. 1988, Kohonen 1990]. It is usually employed after acceptable results have been obtained by applying the standard procedure. LVQ2 refines the solution boundary between regions where misclassifications have occurred. The learning algorithm modifies simultaneously two reference vectors w_1 and w_2 in each learning iteration if:

(i) w_1 and w_2 are the closest and next closest neighbours of the input vector, where w_1 is in the incorrect category and w_2 is in the correct category, and

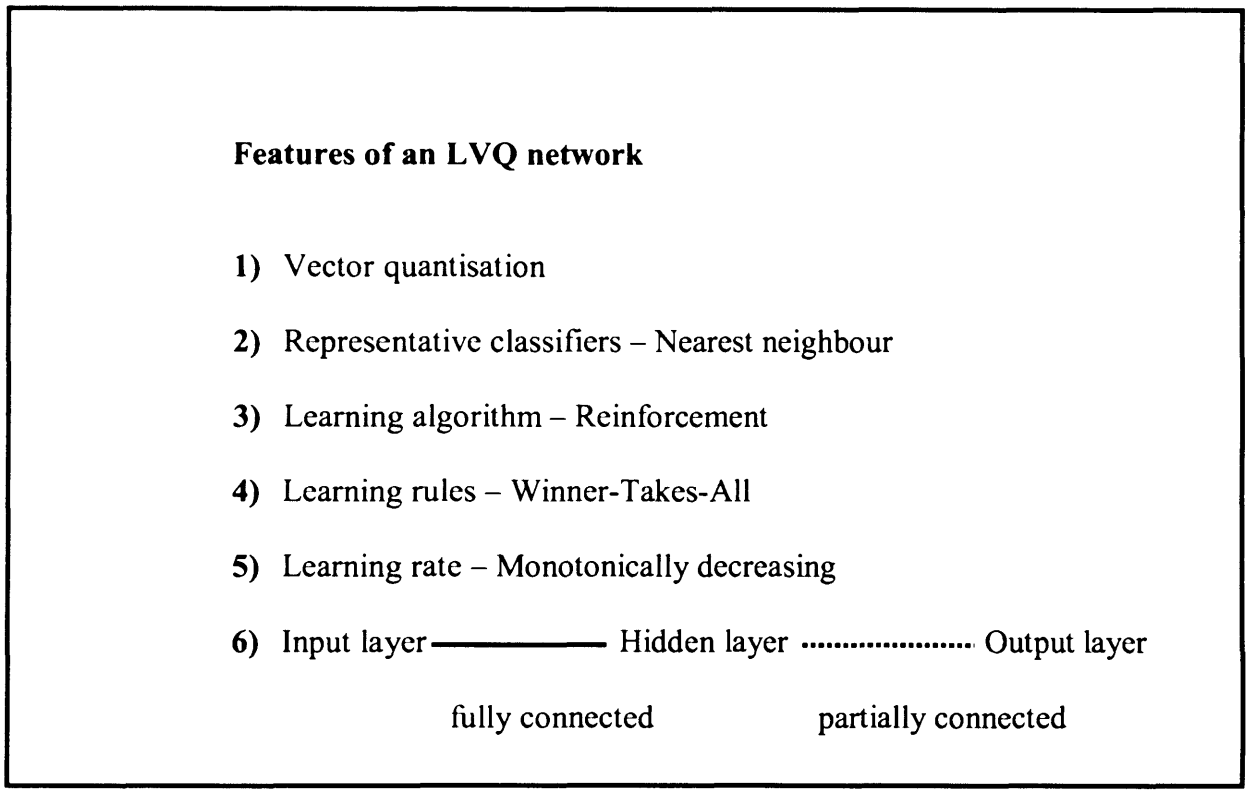

Figure 3.2: Features of an LVQ network

(ii) The input vector x falls inside a window located centrally between w_1 and w_2.

The learning formula for updating the reference vector is given as follows:

$$W_{1new}=W_{1old}-\lambda(X-W_{1old}) \tag{11}$$

$$W_{2new}=W_{2old}-\lambda(X-W_{2old}) \tag{12}$$

where W_{1new} and W_{2new} are the new reference vectors. The learning rate λ is a monotonically decreasing function of time. Other connection weights of the network remain untouched. This weight modification procedure is represented geometrically in Figure 3.3.

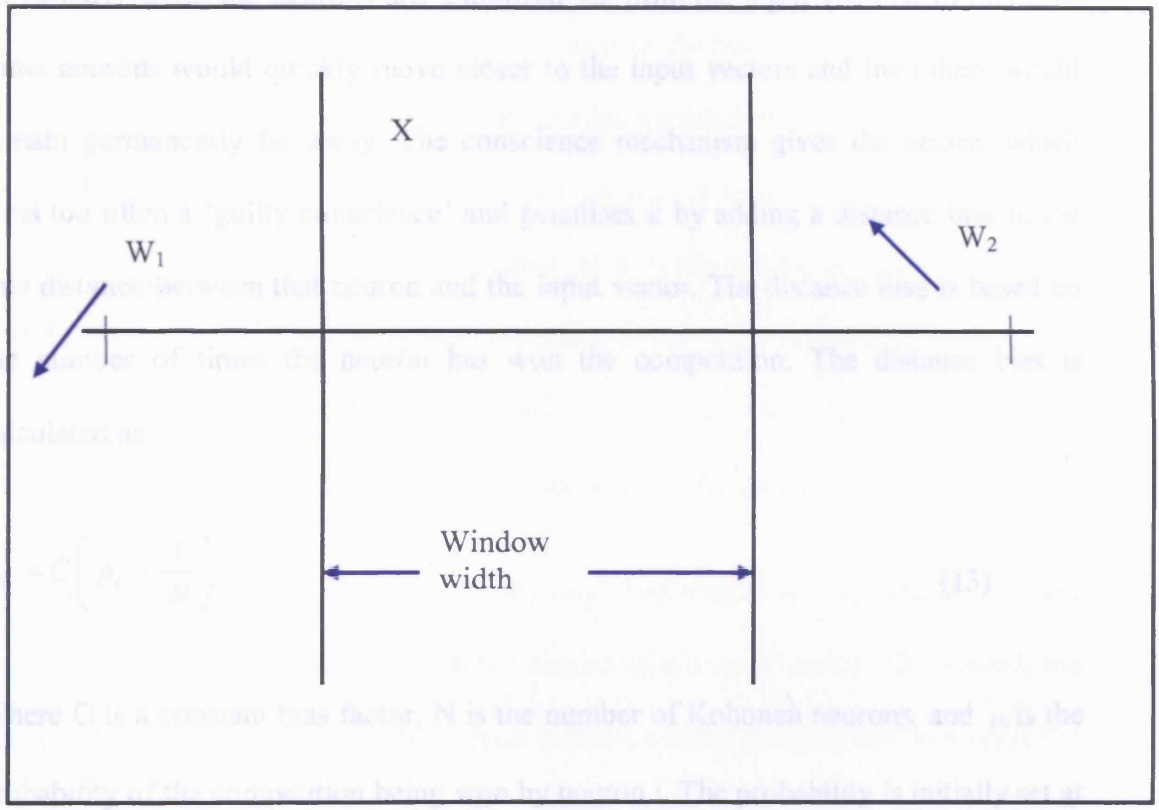

Figure 3.3: Geometric representation of LVQ2 procedure

3.5.2 LVQ with A Conscience

This version of LVQ was originally developed by DeSieno [DeSieno, 1988] to avoid the problem that some neurons tend to win too often while others are always inactive. The standard LVQ algorithm can suffer from this type of problem. This happens particularly when the neurons are initialized far from the input vectors. In this case some neurons would quickly move closer to the input vectors and the others would remain permanently far away. The conscience mechanism gives the neuron which wins too often a 'guilty conscience' and penalises it by adding a distance bias to the true distance between that neuron and the input vector. The distance bias is based on the number of times the neuron has won the competition. The distance bias is calculated as:

$$b_i = C\left(p_i - \frac{1}{N}\right) \tag{13}$$

Where C is a constant bias factor, N is the number of Kohonen neurons, and p_i is the probability of the competition being won by neuron i. The probability is initially set at 1/N but is then updated according to the following equation:

$$p_{i\,new} = p_{i\,old} + B\left(y_i - p_{i\,old}\right) \tag{14}$$

Where B is a constant selected to prevent random fluctuations in the data, y_i=1 if neuron i wins the competition, and 0 otherwise. The new distance $d_{i\,new}$ of neuron i from the input vector is calculated as:

$$d_{i\,new} = d_{i\,old} + b_i \tag{15}$$

The competition is carried out with the new distances, and the same weight updating procedure is then applied as for the standard LVQ.

3.5.3 LVQ-X

The training procedure employed is largely based on the learning procedure originally developed by Kohonen [Kohonen, 1990]. As mentioned before, in the two existing LVQ models, only one weight vector is updated at each learning iteration. In LVQ2, however, two weight vectors are updated at a time, which happens under rare circumstances. In LVQ-X, the extended version of the LVQ learning procedure, two reference vectors are updated in most iterations, resulting in a decrease in the learning time and an increase in the generalisation capability of the system.

The main idea of LVQ-X is to modify two candidate weight vectors. The first, called the "global winner", is the weight vector nearest to the input vector. The second, the "local winner", is the weight vector which is in the correct category and nearest to the input vector in that category. If the global winner is not in the correct category then it is pushed away from the input vector and at the same time the local winner is brought closer. This gives a chance for the correct neuron to win the competition in the next iteration. Obviously, if the global winner is also the local winner then only one weight vector needs to be updated.

In this case, the weights are modified as follows:

$$W_{new} = W_{old} + \lambda(X - W_{old}) \quad (16)$$

where λ is the learning rate. If the global winner is different from the local winner then:

$$W_{new} = W_{old} - \lambda(X - W_{old}) \quad (17) \text{ and}$$

$$W_{new} = W_{old} + \lambda(X - W_{old}) \quad (18)$$

Pham and Oztemel [Pham and Oztemel, 1994] claim that numerical comparison showed that LVQ-X has a better classification accuracy within a shorter training time than LVQ and its two variants, of LVQ2 and of LVQ with a conscience mechanism. Moreover, dependency on the initial values of the weight vectors is virtually eliminated and the performance of the network is almost the same for the cases of arbitrary initial weight values and initial weight values taken from the training set. A comparison of the various LVQ models is given in Table 3.1. The proposed LVQ-X network achieves even better classification accuracy than that obtained with the MLP network.

Pattern recogniser	**Number of training epochs**	**Learning performance (%)**	**Test performance (%)**
LVQ (Standard)	70	95.18	92.31
LVQ2	4	94.31	89.62
LVQ (Standard) + LVQ2	74	96.18	92.61
LVQ (with a conscience mechanism)	70	95.98	92.71
LVQ-X	20	100.0	97.70

Table 3.1: Performance of various LVQ pattern recognisers

3.6 Discussion for LVQ

Experiments showed that in standard LVQ some neurons may win too often while others are always inactive. This means that only a few neurons have been learning, hence resulting in poor initial performance. This situation may occur when a network cannot learn the complete set of training patterns when the weights are randomly initialised. The accuracy levels achieved are below 60%. To solve this initialisation problem, some of the patterns in the training set were assigned as initial weight vectors. The number of patterns that can be learned had increased. However, the network still did not achieve 100% learning. The classification accuracy levels of the network after 70 training epochs are 95.18% for the training data and 92.3% for the test data.

Experiments again showed that the initialisation procedure problem occurs in LVQ2. Patterns from the training set were assigned as the initial values of the weight vectors since the network cannot learn with random initial weight values. Furthermore, after only four training epochs the LVQ2 learning procedure was no longer applicable because of the conditions set out as mentioned above in section 3.5.1. The overall accuracy levels of the network are 94.31% for the training set and 89.62% for the test set after 4 training epochs.

Applying LVQ2 to a network using the standard LVQ algorithm showed some improvement in the accuracy level. The accuracy levels achieved after the network is trained for 70 epochs are 96.18% for the training set and 92.61% for the test set.

Applying a conscience mechanism to the standard LVQ model increases the learning capability of the network. It also helps to reduce the dependence on using training examples as initial weights. Results showed that an LVQ module with a "conscience" can learn 95.98% of the training set and correctly classify 92.71% of the test set following 70 training epochs.

Pham and Oztemel [Pham and Oztemel, 1994] reported that at the end of 10 training epochs, the network of LVQ-X can correctly classify 99.39% of the training set and 96.30% of the test set. After 20 training epochs, the overall recognition accuracy level increases to 100% for the training set and 97.70% for the test set in a shorter training time. Despite the good classification performance of LVQ-X, at the same time, two weights at most need to be modified for LVQ-X. The first is called the "global winner", which is the one globally nearest to the training vector but not necessarily in the correct category. The second is called the "local winner" and is the one nearest to the training vector in the correct category. Pham and Oztemel {Pham and Oztemel, 1994] claimed that this approach gives an opportunity for the correct neuron to win in the next iteration.

It could therefore be concluded that most of the existing LVQ algorithms were originally designed to tackle the problem that some neurons may win too often while others are always inactive, thus reducing the dependency on using training examples as initial weights. Moreover, a significant drawback of many of these techniques is the poor classification accuracy.

The second part of this chapter reviews current trends in SNNs research. The review includes the network architecture and it existing learning procedure. Section 3.8 discusses the pattern recognition problem using SNNs and previous work addressing it. SNNs research has been making significant progress in many directions.

This section examines two of the most important directions and discusses some current problems. The two directions are those based on supervised learning and on unsupervised learning.

3.7 Current Trends in Spiking Neural Networks Research

3.7.1 Typical Spiking Neural Networks Architecture

Spiking neural networks have a similar architecture to traditional neural networks. Elements that differ in the architecture are the numbers of synaptic terminals between each layer of neurons and also the fact that there are synaptic delays. Several mathematical models have been proposed to describe the behaviour of spiking neurons, such as the Hodgkin-Huxley model [Hodgkin and Huxley, 1952], the Leaky Integrate-and-Fire model (LIFN) [Maass, 1997] and the Spike Response Model (SRM) [Bialek, Rieke, de Ruyter and Warland, 1991]. Figure 3.4 shows the network structure as proposed by Natschlager and Ruf [Natschlager and Ruf, 1998].

This structure consists of a feedforward fully connected spiking neural network with multiple delayed synaptic terminals. The different layers are labelled H, I, and J for the input, hidden, and output layer respectively as shown in Figure 3.4. The adopted spiking neurons are based on the Spike Response Model to describe the relationship between input spikes and the internal state variable. Consider a neuron j, having a set D_j of immediate pre-synaptic neurons, receiving a set of spikes with firing times t_i , $i \in D_j$. It is assumed that any neuron can generate at most one spike during the simulation interval and discharges when the internal state variable reaches a threshold. The dynamics of the internal state variable $x_j(t)$ are described by the following function:

$$x_j(t) = \sum_{i \in D_j} w_{ij}\, y_j(t) \tag{19}$$

$y_j(t)$ is the un-weighted contribution of a single synaptic terminal to the state variable which described a pre-synaptic spike at a synaptic terminal k as a PSP of standard height with delay d^k.

$$y_i^k = \varepsilon(t - t_i - d^k) \tag{20}$$

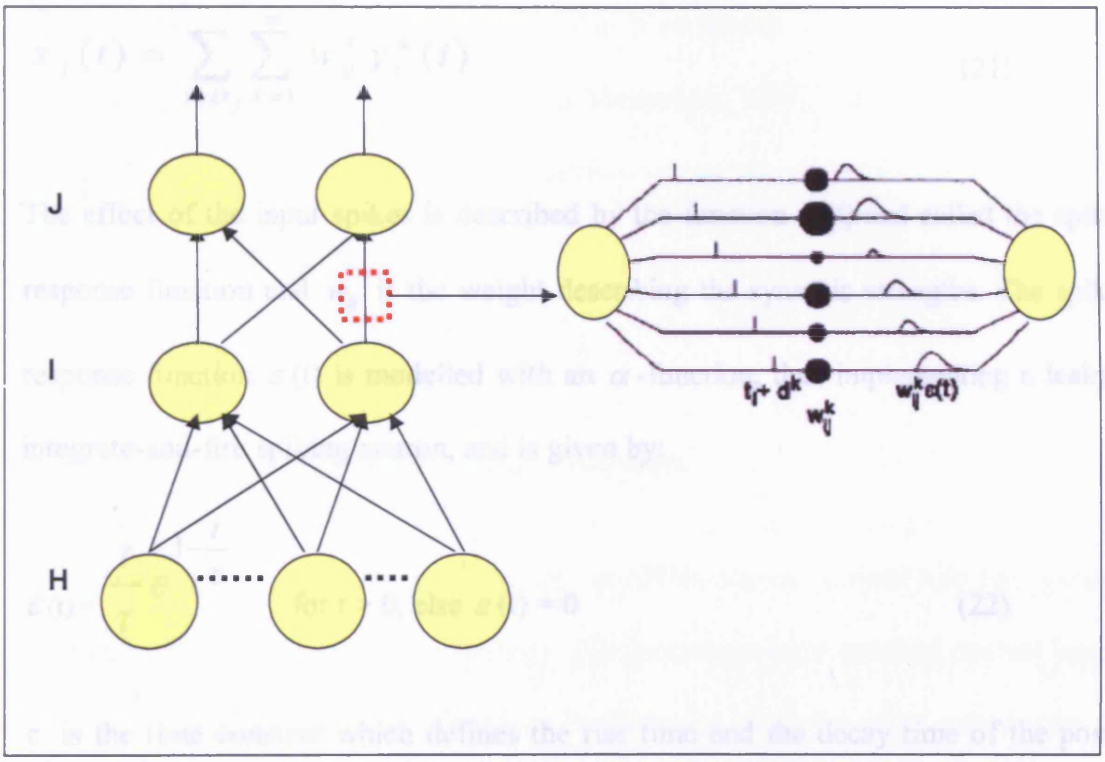

Figure 3.4: Feedforward spiking neural networks

The time t_i is the firing time of pre-synaptic neuron i, and d^k the delay associated with the synaptic terminal k. Considering the multiple synapses per connection case, the state variable $x_j(t)$ of neuron j receiving input from all neurons i is then described as the weighted sum of the pre-synaptic contributions as follows:

$$x_j(t) = \sum_{i \in D_j} \sum_{k=1}^{m} w_{ij}^k y_i^k(t) \qquad (21)$$

The effect of the input spikes is described by the function ε(t) and called the spike response function and w_{ij} is the weight describing the synaptic strengths. The spike response function ε(t) is modelled with an α-function, thus implementing a leaky-integrate-and-fire spiking neuron, and is given by:

$$\varepsilon(t) = \frac{t}{\tau} e^{1-\frac{t}{\tau}} \qquad \text{for } t > 0, \text{ else } \varepsilon(t) = 0 \qquad (22)$$

τ is the time constant which defines the rise time and the decay time of the post-synaptic potential (PSP). The individual connection, which is described in [Natscchlager and Ruf, 1998], consists of a fixed number of m synaptic terminals. Each terminal serves as a sub-connection that is associated with a different delay and weight, see Figure 3.4. The delay d^k of a synaptic terminal k is defined as the difference between the firing time of the pre-synaptic neuron and the time when the post-synaptic potential starts rising. The threshold θ is a constant and is the same for all neurons in the network.

3.7.2 A Review of Existing Spiking Neural Networks Learning Procedure

Network architectures based on spiking neurons that encode information in the individual spike times have yielded, amongst other things, a self-organising map akin to Kohonen's SOM [Ruf and Schmitt, 1998], and networks for unsupervised clustering [Bohte et al., 2000; Natschlager and Ruf, 1998]. The principle of coding input intensity by relative firing time has also been applied successfully to a network for character recognition [Buonomano and Merzenich, 1999]. Recently, a review of a spiking neural network model based on supervised learning for spike time coding has been carried out [Ponulak, 2005].

3.7.2.1 SNNs for Supervised Learning Procedure

Generally, supervised learning procedures in SNNs are categorised into two groups based on the underlying training strategy. The groups are error gradient descent based models and Hebbian rule based models. Furthermore, since the input information in SNNs can be encoded either in the connection weights or delays, the two models can also be grouped based on the encoding strategy. The following subsection gives details of the above mentioned learning models.

3.7.2.1.1 Error Gradient Based Learning Procedures

Bohte et al. [Bohte et al., 2000] proposed a network of spiking neurons that encodes information in the timing of individual spike times. They derive a supervised learning rule, *SpikeProp*, akin to traditional error-backpropagation. They utilise a fully connected feedforward spiking neural network. Each connection between two neurons corresponds to sixteen sub-connections. Each sub-connection is characterised with a different delay and weight. In Bohte et al's work, using this algorithm, they demonstrate how networks of spiking neurons with biologically reasonable action potentials can perform complex non-linear classification in fast temporal coding just as well as rate-coded networks.

A drawback of Bohte et al.'s work is that a large set of weights have to be adjusted, so the size of the network increases drastically with the number of neurons. A simpler learning procedure might help to reduce this problem. The presented *Spikeprop* algorithm was reinvestigated in Schrauwen and Campenhout [Schrauwen and Van Campenhout, 2004], Moore [Moore, 2002], and Jianguo and Embrechts [Jianguo and Embrechts, 2001]. Schrauwen and Campenhout proposed an improvement on Bohte et al.'s model. They proposed to adopt connection delays; a time constant, and the neuron's threshold instead of adapting only the connection weights. In Moore's work [Moore, 2002], the weights were initialized with a value that led the network to successful training in a similar number of iterations as in Bohte's work, but with high learning rates. However, this conflicts with Bohte's work as he argued that the approximation of the threshold function implies that only small learning rates can be used. Jianguo and Embrechts [Jianguo and Embrechts, 2001] improved the proposed model by Bohte et al. by adding a momentum term to the learning rule.

3.7.2.1.2 Hebbian-based Supervised Learning Procedures

Hebbian learning is much more biologically realistic. There are numerous examples in biological modelling studies where Hebbian-based learning has been implemented. Hebb's rule states that synaptic strength will be increased if the post-synaptic neuron and the pre-synaptic neuron are both highly active at the same time. According to the Supervised Hebbian Learning, it is assumed that learning rules apply to all synaptic inputs of the learning neuron and the post-synaptic neuron receives an additional "teaching" input *I(t)*. This additional input could either arise from a second group of neurons or from the intracellular current injection. The role of *I(t)* is to increase the probability that the neuron fires at or close to the desired firing time. Maass [Maass, 1997b] proposed a monosynaptic learning rule which trains a single synapse with temporarily encoded inputs. The network is activated for a time period during each learning cycle. An important assumption in this model is that the potential rise in the neuron due to an incoming spike is linear. However, in a practical situation where neurons have several incoming connections and receive inputs from each connection, it is difficult to find the effect induced by a single connection. Ruf and Schmitt [Ruf and Schmitt, 1997] suggested a Hebbian-based supervised learning model which encodes the information in the connection weights. It was suggested also that the connections in Maass's work can be trained in parallel through a normalisation technique.

3.7.2.2 SNNs for Unsupervised Learning Procedure

Hopfield [Hopfield, 1995] presents a model of spiking neurons for discovering clusters in an input space akin to Radial Basis Functions. Extending Hopfield's idea, Natschlager and Ruf [Natschlager and Ruf, 1998] proposed a learning algorithm that performs unsupervised clustering in spiking neural networks using spike-times as input. This model encodes the input patterns in the delays across its synapse and is shown to reliably find centers of high-dimensional clusters. Generally, there are two kinds of model for the unsupervised learning:

(i) Encodes the input vectors in the connection weights;

(ii) Encodes the input vectors in the connection delays.

Basically, both of the models use Hebbian-based self-organised weight adaptation.

3.7.2.2.1 Weight-based Learning

The model discussed below encodes the input information in the connection weights. Ruf and Schmitt [Ruf and Schmitt, 1998] proposed a model for self-organisation in a network of spiking neurons which encodes the input information in connection weights. The effect of weight normalisation also has been studied in several implementations of the SOM. It was discovered that, after a certain number of learning cycles, approximately the same degree of topology preservation could be achieved regardless of whether or not the weights were normalised.

In contrast to the standard formulation of the SOM, their work has the additional advantage that the winner among the competing neurons can be determined quickly and locally by using lateral excitation and inhibition. These lateral connections also

constitute the neighbourhood relationship among the neurons. In order to realise cooperation among neurons in the neighbourhood of the winning neuron, two simple measures were implemented as follows:

(i) Neurons which are topologically closer were assigned with strong excitatory lateral connections and remote neurons were assigned with strong inhibitory lateral connections. Through these connections the neurons closer to the winning neuron are encouraged to fire while the other neurons are discouraged from firing;

(ii) Neurons which fire temporally closer to the winning neuron are encouraged more than the neurons which fire later after the winning neuron.

Ruf and Schmitt [Ruf and Schmitt, 1998] in this work proposed a self-organising rule as specified in equation (23) below:

$$\delta w_{ji} = \eta \frac{T_{out} - t_j}{T_{out}} \left(x_i - w_{ji} \right) \tag{23}$$

where δw_{ji} is the modification for the connection weight w_{ji} at some learning cycle, ($\eta > 0$) is the learning rate and is slowly decreased during learning, and $x_i = (x_1, \ldots, x_m)$ is the input vector presented to the network. Synaptic modification in equation (23) is based on the firing time t_j of the output neuron. T_{out} is an upper limit to specify the applicability of the rule among the competing neurons. Synaptic connections of those neurons which fire before T_{out} are updated, while others remain unchanged. The term ($\frac{T_{out} - t_j}{T_{out}}$), defines the effect on neighbouring neurons based

on the spike time. Since the winner is the one which fires first, then ($\frac{T_{out} - t_j}{T_{out}}$) will return a higher value, while returning low values for late neurons. Generally, their work showed that the spiking neural network model along with temporal coding is capable of preserving the topology of the input space in a fashion similar to Kohonen's self-organising map.

3.7.2.2.2 Delay-based Learning

In delay-based learning, the input information is encoded in the connection delays through adapting the connection weights. Hebbian-based learning modifies the connection weights based on the time difference between the pre-synaptic and post-synaptic firing of a neuron. Through the weight adaptation strategy, suitable delayed connections are selected while pruning the unwanted connections. In unsupervised learning, the rules applied enhance the strength of some connections while weakening others. This results in some selected delayed connections with high strength and the rest with very low or null strength. This helps to encode the input information effectively in the connection delays. Hopfield [Hopfield, 1995] was the first to introduce this and showed that the input spike patterns can be stored in the delays across the synapses.

His work was supported by Gerstner's [Gerstner et al., 1996] work which established that an encoded delay pattern will balance the differences of the firing times of the input neurons such that the delayed input spikes reach the output neurons at almost the same time, enabling them to fire. This type of learning is claimed to be more like

the learning in a Radial Basis Function (RBF) network. This approach is supported by many neurobiological findings reported in Habberly's, and O'Keefe and Reece's works [Habberly, 1985; O'Keefe and Reece, 1993]. Gerstner et al. [Gerstner et al., 1996] performed a modelling study through computer simulations on the barn owl's auditory system. A Hebbian-based unsupervised learning mechanism was proposed to train a single integrate-and-fire neuron with multisynapse (several sub-connections) connections for each single connection. Each sub-connection is characterised with a weight and a delay. The learning rule proposed here enhances the strength of the connections which are repeatedly active shortly before a postsynaptic spike event. Connections which are active shortly after the postsynaptic event are weakened. This learning rule then selects connections with suitable delays from a distribution of connections with different delays. The learning rule also selects the correct delays from two independent groups of inputs, for example, from the left and right ear.

Natschlager and Ruf [Natschlager and Ruf, 1998] extended the approach reported in Gerstner's et al. work [Gerstner et al., 1996]. In contrast with Gerstner's et al. work, here the firing times of the output neurons are considered where the firing or non-firing of a neuron was taken into consideration. The network architecture deployed in their work is a two layered fully connected feedforward network.

The model proposed in Natschlager and Ruf [Natschlager and Ruf, 1998] was further improved by Bohte et al., [Bohte et al., 2002], in order to increase the precision, capacity, and clustering capability of the specified spiking neural network model. This was achieved through a population coding scheme and the model's performance was proved with clustering several real world data sets. Previous research by Bohte et al.,

[Bohte et al., 2002] on unsupervised learning used the Winner-Takes-All learning rule to modify the weights between the source neurons and the neuron first to fire in the target layer, using a time-variant version of Hebbian learning. The firing time of an output neuron reflects the distance of the evaluated pattern to its learned input pattern. The first neuron to fire is chosen as the winner. If the start of the post-synaptic potential (PSP) at a synapse slightly precedes a spike in the target neuron, the weight of this synapse is increased, as it exerts significant influence on the spike-time by virtue of a relatively large contribution to the membrane potential. Earlier and later synapses are decreased in weight, reflecting their lower impact on the target neuron's spike time. For a weight with delay d^k from neuron i to neuron j, Bohte et al used equation (24) to update the weights;

$$\Delta w_{ij}^{k} = \eta \left[L(\Delta T) = \eta(1-b) e^{-\frac{(\Delta T - c)^2}{\beta^2}} + b \right] \qquad (24)$$

Where the parameter b determines the effective integral over the entire learning window β sets the width of the positive learning window, and c determines the position of this peak. The value of ΔT denotes the time difference between the onset of a PSP at a synapse and the time at which the spike is generated in the target neuron. The weight of a single terminal is limited by a minimum and maximum value of 0 and w_{max} respectively. In their experiments, ΔT is set to [0-9] (ms) and delays d^k to 1-15 (ms) in 1 ms intervals (m=16). The parameter values used by Bohte et al. for the learning function $L(\Delta T)$ were set to: b = -0.2, c = -2.85, β =1.67, η=0.0025 and

w_{max} =2.75. To model the post-synaptic potentials, they used an α -function with τ =3.0 (ms) as in equation (22) in section 3.7.1.

3.8 Discussion of SNNs

Generally, the research work on supervised learning with SNNs discussed above applied a fully connected feedforward network (back-propagation) with multi-synapse connections. The most usual arrangement has 12 to 16 sub-connections, with delays of up to 15 (ms). The combination of this type of architecture with multi-synapse connections will increase the network complexity and training time since a large set of weights have to be adjusted. Lippman [Lippmann, 1991] has carried out a critical overview of neural network pattern classifiers. He presented the results of handwritten digit recognition experiments using Multi-Layer Perceptron, k-Nearest Neighbour, and Radial Basis Function classifiers as shown in table 3.2. The result showed that Back-propagation requires a longer training time compared with the other three classifiers. Research work on unsupervised learning mostly applied the Kohonen's network with delay-based learning as discussed above. Previous research on unsupervised learning used the Winner-Takes-All learning rule to modify the weights between the source neurons and the neuron first to fire in the target layer, using a time-variant version of Hebbian learning.

The remainder of this chapter is organised as follows: section 3.9 explains in depth LVQ and SNN networks' pattern detection capability which motivate the research into this area; section 3.10 details the proposed supervised SNN learning model with an LVQ structure, the so called S-LVQ, for the application to control chart pattern recognition; section 3.13 presents the pattern recognition results obtain using the

	Back-Prop	KNN	RBF
Error rate (no rejections)	5.15%	5.14%	4.77%
Free parameters	5,472	11,016,000	371,000
Training time (hours)	67.7	0.0	16.5
Classification time (secs/char)	0.14	6.22	0.24

Table 3.2: Results of handwritten digit recognition

proposed S-LVQ networks; lastly, section 3.14 discusses some interesting findings on the proposed learning model. This chapter concludes with a summary of all the three parts.

3.9 Motivation for Research

Control chart patterns normally contain a random noise element. Therefore, it would be difficult to classify these patterns using simple heuristic rules with fixed and well-defined detection limits. Experiments have shown that the noise filtering and generalisation capabilities of neural networks make them suitable for this classification task. This is clearly demonstrated by the good identification results presented in table 3.1. Although the standard LVQ network has a relatively poor performance, after a slight modification, it achieves the best classification performance in a short training time. Lippman [Lippman, 1991] reported that characteristics which often differ dramatically across classifiers include classification time, training and adaptation time, ease of implementation, memory requirements, rejection accuracy, and usefulness of outputs as estimated by Bayes' probability. Among the most attractive features of LVQ is the natural way in which it can be applied to multi-class problems and its simple learning rule. The reason for the focus on LVQ networks is the proven strength of their classification abilities [Baig et al., 2001].

Previous research on ANNs showed that most practical applications of ANNs are based on computational models involving the propagation of continuous variables from one processing unit to the next using the concept of mean firing rates. The

concept of mean firing rates has been applied successfully during the last 80 years. It dates back to the pioneering work of Adrian [Adrian, 1926] who showed that the firing rate of stretch receptor neurons in muscles is related to the force applied to the muscle. In the following decades, measurement of firing rates became a standard tool for describing the properties of all types of sensory or cortical neurons [Mountcastle, 1957; Hubel and Wiesel, 1959], due partly to the relative ease of measuring rates experimentally.

It is clear, however, that an approach based on a temporal average neglects all the information possibly contained in the exact timing of the spikes. It is therefore no surprise that the firing rate concept has been criticized repeatedly and is the subject of an ongoing debate [Abeles, 1994; Bialek et al., 1991; Hopfield, 1995; Shadlen and Newsome, 1994; Softky, 1995; Rieke et al., 1996]. Although ANNs are considered to be one of the most powerful and flexible computational models known today, recent research has found that ANNs are not powerful enough as a biological counterpart due to their more simplified approach and coding of information [Zador A M, 2000; Maass W, 1997]. In recent years, more and more neurobiological experimental evidence has accumulated showing clearly that biological neural networks, which communicate through pulses, use the timing of these pulses to transmit information and to perform computation. In addition, SNNs are deemed computationally more powerful than common ANNs formalisms on the basis of extensive theoretical work by Maass [Maass W, 1996].

Together, these realisations have stimulated or motivated a significant growth of research activity in the area of pulsed neural networks. These range from neurobiological modelling and theoretical analyses, to algorithm development and

hardware implementations. Generally, pattern recognition problems are involved with research activity in algorithm development. Much of the research into learning algorithms for pulsed neural networks has been focused on unsupervised learning and most of these existing learning algorithms adjust the synaptic weights based on the adaptation of a Hebbian rule. According to Ammar [Ammar et al., 2003] the first supervised training was suggested by Bohte [Bohte et al., 2000] where the classical back propagation, which is a gradient descent based algorithm, is adapted to temporal coding, and an approximation of the post-synaptic potential is assumed to allow derivation.

However, a large set of weights has to be adjusted since a connection between two neurons corresponds to sixteen sub-connections, so the size of the network increases drastically with the number of neurons. Therefore, more research into supervised learning for pulsed neural networks is essential. In addition, a more efficient supervised learning algorithm is needed for the better exploitation of pulsed neural networks models.

Together with the advantages of LVQ, a new approach for supervised training called the "Spiking Learning Vector Quantisation (S-LVQ)" algorithm is proposed in this chapter to address this problem. The proposed S-LVQ uses spiking neurons instead of the common neurons in LVQ. It uses the motivation concepts in behavioural neuroscience [Berridge, 2004], instead of a penalty, to encourage a neuron to be a winner. It is expected that these concepts, together with the new updating weights rule proposed in this chapter, will help to increase the classification performance. The proposed S-LVQ updates the weights of the winning neurons and of its neighbours in the same cluster simultaneously. Furthermore, S-LVQ gives more concentration to the

neurons in the correct category and in the same cluster, as this is more practical. This will help to fully use the number of hidden neurons in each cluster in the network. This concentration might lead to a decrease in the number of inactive neurons in the network.

The remainder of this chapter is organised as follows: section 3.10 describes in detail the proposed S-LVQ that consists of the suggested network structure and the supervised learning algorithm; section 3.12 describes the training and testing data sets in control charts; in section 3.13 an empirical evaluation of the method is presented; section 3.14 provides general discussion about an interesting finding on the effect on classification accuracy of various hidden neurons; finally, a summary of the findings of the chapter is given.

3.10 Proposed S-LVQ Algorithm

3.10.1 Network Structure

This thesis proposes a new architecture for spiking learning vector quantisation for control chart pattern recognition. Generally, the proposed architecture uses the structure of an LVQ network with some modification in the connection. It consists of spiking neurons instead of common neurons. It is a feedforward network of spiking neurons which is fully connected between the input and hidden layers. It has multiple delayed synaptic terminals (m) and is partially connected between the hidden and output layers, with each output neuron linked to a different cluster of hidden neurons. An individual connection consists of a fixed number of m synaptic terminals, where each terminal serves as a sub-connection that is associated with a different delay and

weight between the input and hidden layers. The weights of the synaptic connections between the hidden and output neurons are fixed at 1. Experiments were carried out with a number of network structures with different parameters and learning procedures. The network finally adopted had 60 input neurons in the input layer, which means the input patterns consisted of the 60 most recent mean values of the process variable to be controlled. One input neuron was therefore dedicated for each mean value. There were six output neurons, one for each pattern category, and 36 hidden neurons (as in LVQ). Table 3.3 shows the details of the networks used. At the beginning of training, the synaptic weights were set randomly between 0 and +1. The input vector components were scaled between 0 and 1. Using a temporal coding scheme, the input vector components were then coded by a pattern of firing times within a coding interval, and each input neuron was allowed to fire no more than once during this interval. In this work, the coding intervals ΔT were set to [0-100] ms and the delays d^k to {1,, 15}[ms] in 10 ms intervals. The available synaptic delays were therefore 1-16 (ms). These parameters were chosen experimentally to produce the best results. The post-synaptic (PSP) was defined by a α-function with a constant time τ=120 (ms). Input vectors were presented sequentially to the network together with the corresponding output vectors identifying their categories as shown in Table 3.3. Unlike the network structure and its variants used in the standard LVQ, as shown in Figure 3.1 and Figure 3.2 respectively, the proposed structure has different features:

(i) It uses spiking neurons instead of the common neurons;

(ii) It uses multi-synapse terminals instead of single reference vector between input layer and hidden layer;

(iii) For each reference vector each multi-synapse terminal has delay and weight instead of weight only.

The difference between spiking neurons and conventional neurons that enable spiking neurons to outperform conventional neurons is that they represent a more plausible model of real biological neurons, since spiking neurons consider time as an important feature for information representation and processing. Another important feature is that the models of SNNs are much more nonlinear and that more parameters are considered than in the conventional networks. Hence, a network of spiking neurons appears to be an interesting tool for investigating temporal neural coding and for exploiting its computational potential in a much more sophisticated manner than offered by conventional networks.

Another difference in the application of the network structure is the multiple synapse approach. The existence of multiple synapses is biologically plausible [Wolf, Zhao and Roberts, 1998], since in brain areas like the neocortex a single pre-synaptic axon makes several independent contacts with the post-synaptic neuron. The practical aspects of this theory have been discussed recently and using this approach for neural computation has already been demonstrated [Wei and Fahn, 2002; Nager, Storck, and Deco, 2002; Natschlager, Maass, and Zador, 2001; Maass and Zador, 1999]. Instead of a single synapse, with its specific delay and weight, this synapse model consists of many sub-synapses, each one with its own weight and delay as shown in Figure 3.5. The use of multiple synapses enables an adequate delay selection using the learning rule that is presented in the next section. For each multiple synapse connecting input neurons to hidden neurons, the resulting PSP is given by equation (21) in sub section 3.7.1.

Number of inputs = 60	Number of outputs = 6
Number of hidden neuron for each output category = 6	Initial weights range = 0 to 1
Scaling range = 0 to 1	Coding interval = 0 to 100
Learning rate = 0.0075	Delay intervals = 15 (ms) in 10 (ms) interval
Synaptic delays = 1 to 16 (ms)	Time constant = 120 (ms)

Table 3.3: Details of the proposed S-LVQ network used for control charts

Pattern	Outputs					
	1	2	3	4	5	6
Normal	1	0	0	0	0	0
Increasing trend	0	1	0	0	0	0
Decreasing trend	0	0	1	0	0	0
Upward shift	0	0	0	1	0	0
Downward shift	0	0	0	0	1	0
Cycle	0	0	0	0	0	1

Table 3.4: Representation of the output categories

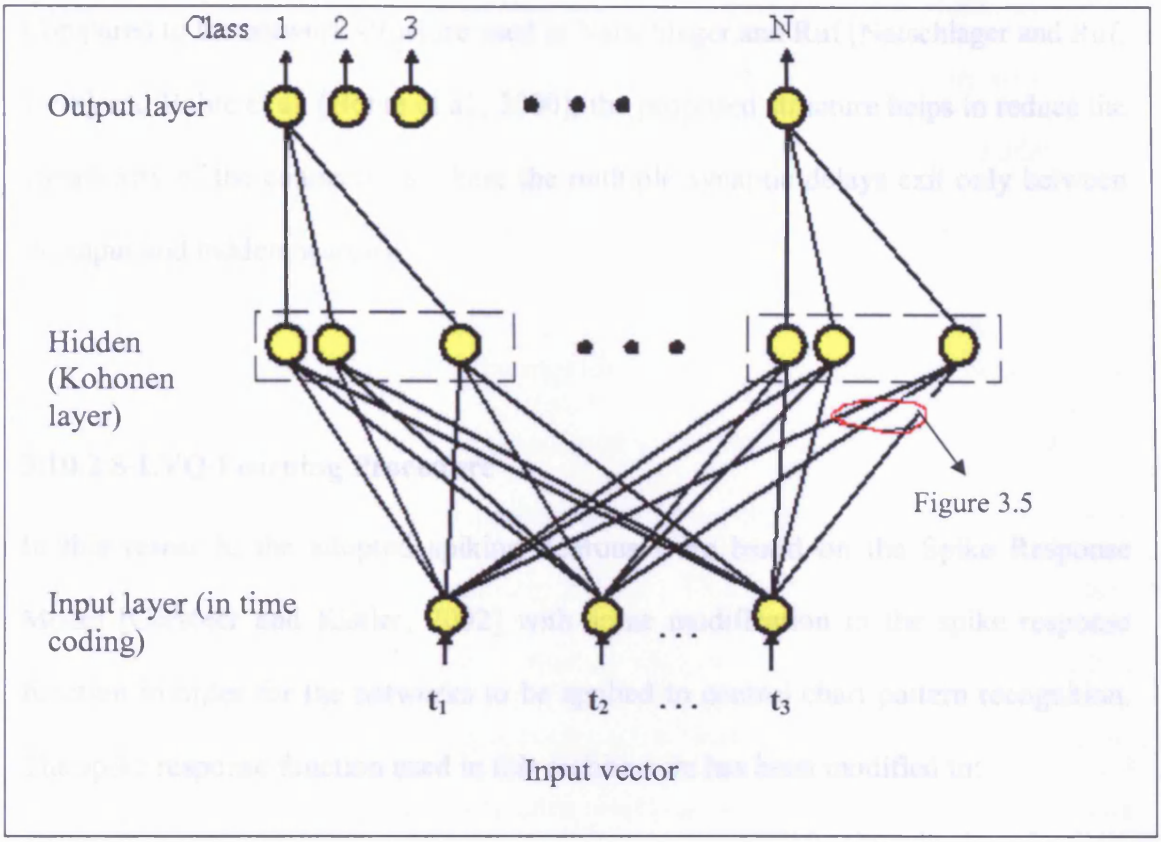

Figure 3.5: The proposed S-LVQ network structure

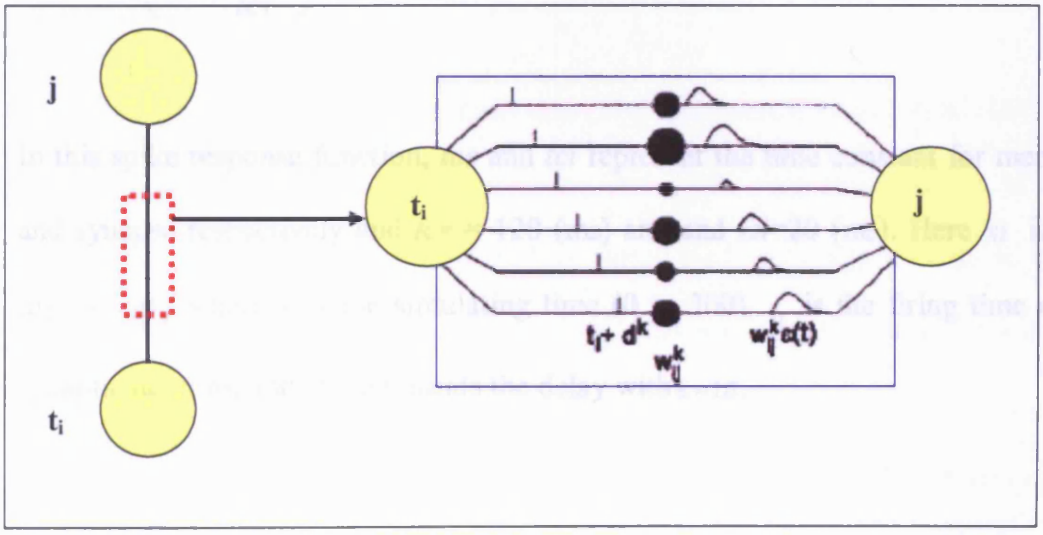

Figure 3.6: Multi-synapse terminals for the S-LVQ spiking neural network

Compared to the network structure used in Natschlager and Ruf [Natschlager and Ruf, 1998] and Bohte et al. [Bohte et al., 2000], the proposed structure helps to reduce the complexity of the connections where the multiple synaptic delays exit only between the input and hidden neurons.

3.10.2 S-LVQ Learning Procedure

In this research, the adopted spiking neurons were based on the Spike Response Model [Gerstner and Kistler, 2002] with some modification to the spike response function in order for the networks to be applied to control chart pattern recognition. The spike response function used in this architecture has been modified to:

$$\varepsilon(t) = \left(\frac{1}{1 - \frac{tce}{tci}} \right) \left(e^{-\frac{(1+st)}{tci}} - e^{-\frac{(1+st)}{tce}} \right) \qquad (25)$$

In this spike response function, *tce* and *tci* represent the time constant for membrane and synapse respectively and *tce* = 120 (ms) and and *tci*=20 (ms). Here st is equal to $(t - t_i - d^k)$ where t is the simulating time (0 to 300), t_i is the firing time of pre-synaptic neurons, and d^k represents the delay with $k = 16$.

With this proposed spike response function, the spiking neural network technique worked well for control chart data. Bohte et al [Bohte, 2000] have stated that "Depending on the choice of suitable spike response functions, one can adapt this model to reflect the dynamics of a large variety of different spiking neurons."

There are, at most, two weights to be modified for the LVQ-X and the variant of the LVQ as discussed in section 3.6. As discussed in sections 3.3 to section 3.6, the main problem with standard LVQ is that only the winning neuron is permitted to modify the connection weights in each learning iteration. This is the so-called "winner-take-all" competition. This will result in a situation where some neurons may win too often while others are always inactive. Although each cluster of neurons has more than one neuron, most of the time only one neuron seems to be contributing or playing a role during the learning. This means that only a few neurons have been learning and the potential of the other neurons, especially the neurons in the cluster which belong to the correct category, is wasted.

To address this problem, a new supervised LVQ architecture based on an SNN model learning algorithm is proposed. Moreover, due to the time-series nature of control chart data, an SNN with temporal coding would more naturally be able to learn to process the data and detect any patterns in it. This is expected to result in a compact classifier that is easy to train. There are two methods applied in the proposed network (S-LVQ): firstly, to boost the neuron potential; secondly, to motivate the neurons.

3.10.2.1 Boosting

Generally, one spike is not enough to cause a post-synaptic neuron to fire. Biological neurons typically must have their potentials raised by around 20mV, while individual spikes change this potential by a few millivolts at most [Maass, 1997]. Therefore, to determine a potential for a neuron at a certain time t, it is necessary to integrate the response function for each spike encountered before time t. This could be a daunting task as spikes accumulate over a simulation. This might result in M potential winners, corresponding to the top M external inputs. In general, when the external inputs are close in magnitude, M tends to be larger. If $M > 1$, the selection of the actual winner is strongly influenced by the initial states (membrane potential). For some initial states, the winner is the first neuron to spike, and the computation is done at the first spike of the network [Jin and Seung, 2002]. Experiments performed on control chart data sets have confirmed this biological phenomenon in the proposed network. This means that there is more than one potential winner and the selection of the actual winner is strongly influenced by the initial states (membrane potentials).

To address this problem, a boosting technique is applied to the proposed learning rule. With supervised learning, the classes of the training patterns are known. When a training pattern is presented to the network, a bias value may be injected to boost the initial state of the spiking neuron's potential in the cluster that belongs to the class of that training pattern. This bias value is selected after a few experiments to give a reliable value. The selected value will produce a reasonable distance for the particular cluster from the other clusters in the network. The best value found for control chart data with the proposed S-LVQ network was in the ranges of 0.5 to 1.0. The value chosen is fixed for all the spiking neuron clusters in the network. The effectiveness of

this boosting technique depends upon how efficiently the updating weights are implemented and upon the regularity of the data to which the training is applied.

3.10.2.1 Motivating

Concepts of motivation are vital to progress in behavioural neuroscience. Motivational concepts help one to understand what limbic brain systems are chiefly evolved to do, i.e., to mediate psychological processes that guide real behaviour. Motivational concepts are needed to properly understand how real brains generate real behaviour [Berridge, 2004]. Epstein [Epstein, 1982] suggested that three additional criteria are needed to distinguish truly motivated behaviour. These criteria are as follows:

(1) Flexible goal directedness. This means behavioural demonstration that the target was a true goal, shown by flexible learning and coordinated appetitive behaviour aimed at obtaining the goal, both changing appropriately when the alteration of circumstances necessitate new strategies to obtain the goal. It means ruling out both simple forms of learning and simple drive activation of behaviour.

(2) Goal expectation. This means expecting to find something interesting.

(3) Affect. This means that real motivation is always accompanied by affective reactions to the goal itself.

These three criteria are used as a guideline to measure the effectiveness of the motivation concepts implemented in the proposed learning rule. This will be supported by empirical evaluation of the proposed learning rule.

In the proposed learning rule, the goal is to learn a set of firing times at the output layer, for a given set of temporal input patterns. The learning rule is easy to implement. The proposed S-LVQ updates not only the weights of the winning neuron but also the other spiking neurons in the same cluster simultaneously. Moreover, instead of penalising the winner as implemented in previous work, the winner and the spiking neurons in the same cluster are motivated.

The main idea or the goal expectation here is that the spiking neurons in the same cluster with the winner are as motivated as the candidates to be the winner in the next iteration. By motivating the other spiking neurons in the same cluster with the winner, their weight vectors which are in the correct category with the winner will get closer to the input vector in that category. Obviously, this approach gives a chance to the other spiking neurons in the same category with the winner to win the competition in the next iteration. In this way, most probably, all the spiking neurons in the same cluster will be activated. Clearly, this method gives more concentration to the spiking neurons in the same cluster with the winning neuron.

Based on the goal and its expectation mentioned above, the learning rule should give an affective reaction which acts on the goal itself. As more spiking neurons become activated, then the performance of the learning algorithm will become better.

A pseudo-code description for updating the weight of S-LVQ is given in Figure 3.6. The details of the pseudo-code are as follows.

Procedure Training For S-LVQ Algorithm

1) Define the network structure ;

2) Initialise the weights and the delays;

3) Encode the input data into temporal coding;

4) Present a training input pattern to the network;

5) **For** each t (simulation time) **Do**

i) **Update the synapse potential ;**

ii) **Update the output (boosting);**

While (t < time window)

6) Find the winner;

7) Update the weights as the following:

For Category = 1 **Do**

Process = 1 End For

For Category = 2 **Do**

Process = 2 End For

Continue until

For Category = n **Do**

Process = n End For

8) Return to (4).

Figure 3.7: A pseudo-code description for S-LVQ Algorithm

The algorithm requires the network structure definition. This involves deciding on the number of inputs and outputs, the number of sub-connections, the learning rate, the time constant, and the threshold parameter. The chosen values for these parameters were mentioned in sub-section 3.10.1 and in table 3.3. SNN is a complex network model with a number of parameters which control its functionality. Tuning the network by assigning appropriate values for these parameters is essential for the smooth functioning of the network and for obtaining optimum performance. There is a lack of clear guidelines regarding the selection of these parameters. Hence, in this work, these parameters were found by analysing the preliminary results obtained with some initial trial values.

In step 2, initialising the network plays an important role in the learning process. The important parameters here are the weights and delays of the interconnections between the input neurons and the output neurons. The chosen values for these two parameters have been mentioned in sub-section 3.10.1 and table 3.3 In the proposed model the connection weights are assigned only with positive values and all the connections are realised as excitatory with positive spike response function $\varepsilon(t) \in [0.1]$. This is to ensure that the effect of most of the input parameters contributes positively to the output.

In step 3, another important aspect of implementing the model is effected, the temporal coding for the continuous input values. Precision of the temporal code should be selected in a way as to attain adequate accuracy with optimal computational efficiency. Experiments have revealed that an input time window with 100 units is adequate in the case of control chart data sets for this learning rule.

In step 4, a training input pattern will be presented to the network sequentially.

In step 5, the simulation time (t) is selected as from 0 to 300 units. For each t, the network is updating the synapse potential for the training input pattern. This is then followed by updating the output. The potential neurons of the class belonging to that particular training input are raised here.

In step 6, the system will determine the first neuron to fire. The first neuron to fire is determined as the winner and will specify the class of the input vector.

In step 7, the winning neuron and other neurons in the same cluster with the winner will be modified or updated. If the winning neuron is in category (class) 1, then proceed with process 1. Process 1 involves updating that winning neuron and also other neurons in category 1 only. If the winning neuron is in the correct category and $\Delta T > 0$, the neurons will be updated using equation (26). If $\Delta T < 0$, equation (27) is used, which is mentioned below. If the winning neuron is in the incorrect category for $\Delta T > 0$ or $\Delta T < 0$, then equation (28) is used. This process will be continued for other categories.

In step 8, the system will return to step 4 with a new training input pattern and will repeat the procedure until all training patterns are correctly classified (or a stopping criterion is met).

In this research, the unsupervised learning equations in (24) in sub section 3.7.2.2.2 were modified to create a supervised learning equation using the following update equations. Learning is achieved through adapting the weights in the network connections to encode the input information.

If the winner is in the correct category, then:

$$w_{new} = w_{old} + dw \quad \text{where}$$

$$dw = \left(\eta \left(\frac{1}{\beta\lambda} \right) e^{-\left(\frac{\Delta T}{2\beta^2} \right)} \right) for \Delta T > 0 \text{ or} \qquad (26)$$

$$dw = \left(-\eta \left(\frac{1}{\beta\lambda} \right) e^{-\left(\frac{\Delta T}{2\beta^2} \right)} \right) for \Delta T < 0 \qquad (27)$$

If the winner is in the incorrect category, then:

$$w_{new} = w_{old} - dw \quad \text{where}$$

$$dw = \left(\alpha\eta \left(\frac{1}{\beta\lambda} \right) e^{-\left(\frac{\Delta T}{2\beta^2} \right)} \right) \qquad \text{for} \quad \Delta T > 0 \quad or \quad \Delta T < 0 \qquad (28)$$

In the simulation, the parameter values for the learning function $L(\Delta T)$ were set to: η = 0.0075, β = 35, $\lambda = \sqrt{(2 * (22/7))}$, ΔT = [0-100], α = 0.8, w_{new} is the new value for the weight, and w_{old} is the old weight value. The parameter η is the constant learning rate. Parameter β sets the width of the positive part of the learning window and ΔT denotes the time difference between the onset of a PSP at a synaptic terminal and the time of the spike generated in the winning output neuron. Parameter α was used because in supervised learning there is prior information about the training sets.

3.11 Setting the Weights, Delay, and Threshold

In this chapter, in order to realise a neuron as an integrator, a time constant that is longer than the time input window is selected. In addition, the threshold θ value should be reliable in order to get better accuracy. A suitable threshold value for this experiment could be the number of inputs multiplied by the number of multi-synapses multiplied by the average of the connection weights, and all this multiplication divided by a constant number. In this experiment, the corresponding value is 10. The weights range is 0 to 1 and the delay interval is 10 for each of the synapses.

3.12 Data Set

Experiments have shown that the ability of the networks to generalise is affected significantly by the quality of data available and by the effectiveness of the techniques used for analysing the data. The process simulator designed to create the required

training data set has been described in Chapter 2. There were 1500 data sets generated using this simulator. From these data sets, 1002 were used as training patterns (167 patterns in each category) and 498 (83 patterns in each category) were used for the testing patterns. The patterns were sequentially applied to the network.

3.13 Empirical Evaluation of S-LVQ

This section presents an empirical evaluation of the control chart pattern recognition performance with the S-LVQ algorithm. Two criteria were used to evaluate the performance of the tested algorithm, namely, number of training epochs and classification accuracy. The number of epochs was taken as the total number of epochs to get the best performance during the training and testing process.

Classification accuracy is generally the most important criterion in control chart pattern recognition performance. It is defined as the percentage of instances from the test set that were correctly classified when the network developed from the corresponding training set was applied. The accuracy level was calculated using the following equation.

$$Accuracy\ (\%) = \frac{Number\ of\ patterns\ correctly\ classified}{Total\ number\ of\ patterns\ tested}\ X\ 100$$

The results obtained with the proposed architecture and the supervised learning procedure for control chart pattern recognition are presented in Table 3.5, together with the results obtained with an LVQ network and its variants. The results in Table 3.7 are presented graphically in Figure 3.7.

Pattern recogniser	Number of training epochs	Learning performance (%)	Test performance (%)
LVQ (Standard)	70	95.18	92.31
LVQ2	4	94.31	89.62
LVQ (Standard) + LVQ2	74	96.18	92.61
LVQ (with a conscience mechanism)	70	95.98	92.71
LVQ-X	20	100.0	97.70
S-LVQ	40	99.85	98.28

Table 3.5: Results of different pattern recognisers applied to control chart data set.

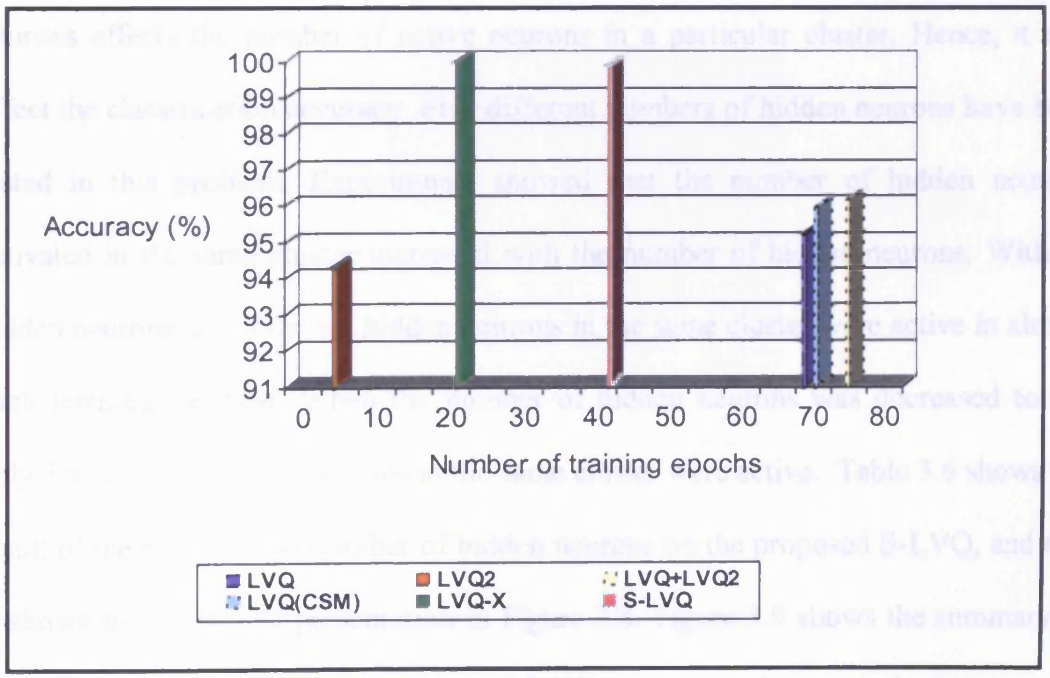

Figure 3.8: Graph showing the result of the different pattern recognisers

3.14 Comparison with LVQ and its Variants

S-LVQ was compared with LVQ and its variant as mentioned in section 3.5. Experimental results have demonstrated that the proposed S-LVQ gives better performance compared to the other five LVQ pattern recognisers. With only 40 training epochs, it can reach 98.28% of classification accuracy. Although LVQ-X has fewer numbers of training epochs, the classification accuracy is only 97.70%. Overall, S-LVQ gives the best performance among the other pattern recognisers.

3.15 The Effect of Number of Hidden Neurons on S-LVQ

The primary factors which controlled the behaviour of the LVQ network were the number of hidden neurons, the learning rate, and the training time. In the proposed S-LVQ, some interesting findings have been made. In S-LVQ, the number of hidden neurons effects the number of active neurons in a particular cluster. Hence, it will affect the classification accuracy. Five different numbers of hidden neurons have been tested in this problem. Experiments showed that the number of hidden neurons activated in the same cluster increased with the number of hidden neurons. With 36 hidden neurons, all of the six hidden neurons in the same cluster were active in almost each learning iteration. When the number of hidden neurons was decreased to 24, only 3 or 2 of the hidden neurons in the same cluster were active. Table 3.6 shows the result of the effect of the number of hidden neurons on the proposed S-LVQ, and this is shown in a graphical presentation in Figure 3.8. Figure 3.9 shows the summary of the implementation of the proposed S-LVQ algorithm for control chart pattern recognition.

Number of hidden neurons	Number of active neurons	Learning performance (%)	Test performance (%)
12	Less than 2	88.73	87.22
18	Less than 3	92.02	90.33
24	Less than 4	96.88	94.25
36	All neurons in the same cluster	99.85	98.28
42	All neurons in the same cluster	99.97	98.30

Table 3.6: The effect of the number of hidden neuron

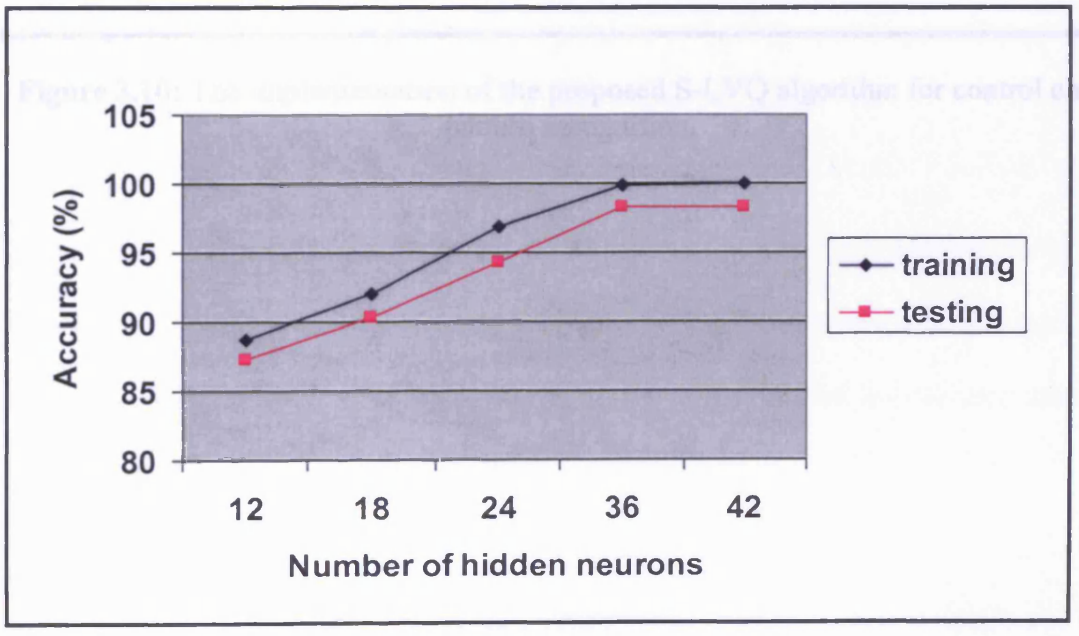

Figure 3.9: The classification accuracy for different number of hidden neurons

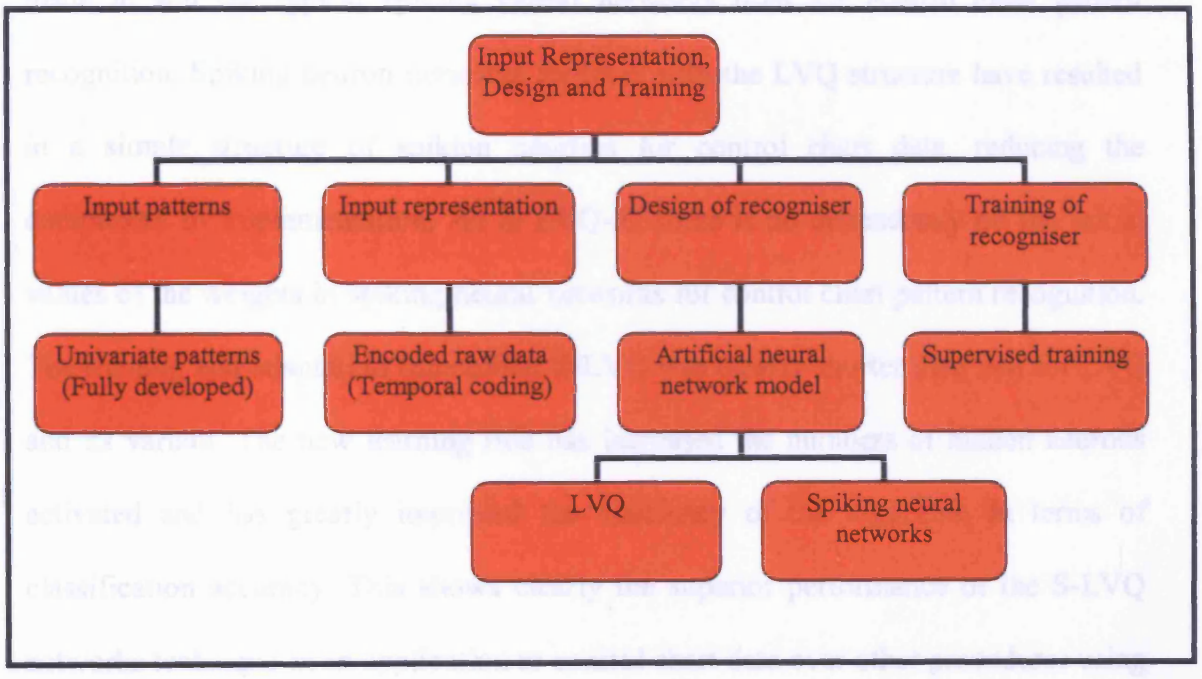

Figure 3.10: The implementation of the proposed S-LVQ algorithm for control chart pattern recognition.

3.16 Summary

This chapter has presented a new spiking learning vector quantisation (S-LVQ) algorithm for classification learning, particularly for control chart patterns which include the new architecture and new learning rule for S-LVQ. Modifications were made to suit the typical spiking neural networks used for control chart pattern recognition. Spiking neuron networks together with the LVQ structure have resulted in a simple structure of spiking neurons for control chart data, reducing the complexity of implementation. As in LVQ-X, there is no dependency on the initial values of the weights in spiking neural networks for control chart pattern recognition. The training and adaptation time of the S-LVQ was clearly shorter than that for LVQ and its variant. The new learning rule has increased the numbers of hidden neurons activated and has greatly improved the efficiency of the algorithm in terms of classification accuracy. This shows clearly the superior performance of the S-LVQ networks technique in an application to control chart data over other procedures using traditional neural networks.

The algorithm presented in this chapter is an implementation of spiking neurons to the standard LVQ which involved some modification of the architecture and learning rule. A new algorithm based on S-LVQ but with much simpler architecture than S-LVQ is considered in the next chapter.

CHAPTER 4

ENHANCED S-LVQ NETWORK (NS-LVQ)

4.1 Previous Work

Extensive recurrent connections between neurons exist in the brain. This has inspired researchers to propose recurrent neural network models with multiple feedback loops for many computations done in the brain, such as the Winner-Takes-All (WTA) computation [Hahnloser et al., 2000]. A WTA results when the dynamics of the network lead to sustained spiking of a single neuron or a group of neurons (the "winner"), although all neurons are driven by external inputs and are capable of spiking in the absence of couplings with other neurons. WTA behaviour in the brain's neural networks could be the basis of perceptual decision making [Salzman and Newsome, 1994] and control of visual attention [Niebur and Koch, 1996; Lee et al., 1999]. WTA can also be used for implementing universal computations [Maass, 2000] and as a hierarchical model of vision [Riesenhuber and Poggio, 1999]. However, recurrent networks are often assumed to be slow in converging to the computational results [Jin and Seung, 2002]. Jin and Seung in their work have shown that recurrent networks can perform fast computation if the detailed dynamics of individual spikes are considered. They specifically analyse a simple spiking recurrent network that performs WTA computation. In their work, they impose a structural symmetry on the network by using neurons with identical parameters, that is excitatory connections with the same strength. However they also neglect the time

course of the spikes and the time delay of spike transmissions. The external inputs are modelled as constant currents injected into the neurons. When inhibition and/or excitation are strong enough, the network performs a WTA computation for all possible external inputs and initial states of the network. The computation is done as soon as the winner spikes once. This is because the inhibition from the winner prevents other neurons from spiking. In general, the selection of the winner can be strongly influenced by the distribution of the external inputs and by the initial states of the network. In another case, if a group of neurons get external inputs close to the maximum input, the network is multistable, and any neuron in the group can be the winner. As to which neuron will be the actual winner, this depends largely on their initial membrane potentials.

Previous studies all assumed a particular initial state of the networks where all neurons are at the resting membrane potential, and they proposed that the winner will be the neuron with the maximum external input. In the brain, this assumption is too restrictive, since the membrane potential of the neurons often deviates from the resting membrane potential because of noise and the inputs from other brain areas.

Generally, their analysis shows that initial states of the network can strongly influence the selection of the winner, depending on the distribution of the external inputs. Their work also showed that WTA have the potential to carry out fast computation with suitable parameters and initial state of the membrane potential of the network.

Inspired by the fast computation of the Winner-Takes-All learning rule, together with the superior results of S-LVQ and the finding about SNNs with fewer neurons [Maass, Schnitger, and Sontag, 1991], this chapter presents an enhanced S-LVQ and so-called NS-LVQ

4.2 Motivation for Research

According to Ammar [Ammar et al., 2003], the first supervised training was suggested in Bohte [Bohte et al., 2000] where the classical back propagation, which is a gradient descent based algorithm, is adapted to temporal coding, and an approximation of the post-synaptic potential is assumed to allow derivation. A large set of weights have to be adjusted, since a connection between two neurons corresponds to sixteen sub-connections. The size of the network therefore increases drastically with the number of neurons. For this reason, more research into supervised learning for pulsed neural networks is essential. In addition, a more efficient supervised learning algorithm is needed for the better exploitation of the pulsed neural networks models.

To address this problem, a new approach for supervised training is proposed in Chapter 3 above with the so-called S-LVQ algorithm. The algorithm is applied to control chart pattern recognition. As mentioned in Chapter 3, S-LVQ is the combination of LVQ structure and spiking neuron models. The structure of LVQ was chosen because it is simple and easy to implement. These factors help to reduce the complexity of the spiking neuron network with multi-synapse. Results showed that

the algorithm gave better performance compared to traditional neural networks. However, previous experiments with spiking neuron networks have demonstrated the ablility to perform well with fewer neurons than a traditional neural network [Maass, Schnitger, and Sontag, 1991].

The proposed NS-LVQ presents an improvement to the network architecture of S-LVQ. The proposed architecture consists of a feedforward network with a simple structure of spiking neurons where each class of control chart patterns is represented by a single hidden neuron. The simple structure reduces the number of weights to be adjusted since it is fully connected between input layer and hidden layer only. It is partially connected between the hidden layer and output layer. The next section is organised as follows: first, section 4.3 will describe the proposed spiking neural networks architecture; next, section 4.4 gives the description of setting the weights, delays and the threshold; section 4.5 introduces the technique of pre-process weights; the next section describes the learning procedure applied here, which defines its characteristics, and its strengths and limitations are given in the context of classification problems; next comes an empirical evaluation of the proposed network and lastly a description of an experiment using a different learning rate technique on the proposed algorithm.

4.3 NS-LVQ Networks Architecture

This structure consists of a feedforward network fully connected between the input and hidden layers with multiple delayed synaptic terminals (m) and partially connected between the hidden and output layers, with each output neuron linked to different hidden neurons. An individual connection consists of a fixed number of

m synaptic terminals, where each terminal serves as a sub-connection that is associated with a different delay and weight between the input and hidden layers. The weights of the synaptic connections between the hidden and output neurons are fixed at 1. Experiments were carried out with a number of network structures with different parameters and learning procedures. The networks finally adopted had 60 input neurons in the input layer, which means the input patterns consisted of the 60 most recent mean values of the process variable to be controlled. One input neuron was therefore dedicated for each mean value. There were six output neurons, with one for each pattern category, and six hidden neurons where the number of hidden neurons here depends on the number of classes. Table 4.1 shows the details of the networks used.

At the beginning of training, the synaptic weights were set randomly between 0 and +1. The input vector components were scaled between 0 and 1. Using a temporal coding scheme, the input vector components were then coded by a pattern of firing times within a coding interval and each input neuron allowed firing once at most during this interval. In this work,, the coding intervals ΔT were set to [0-100] ms and the delays d^k to {1,..., 15} [ms] in 10 ms intervals. The available synaptic delays were therefore 1-16 ms. The PSP was defined by an α -function with a constant time τ =150 ms. Input vectors were presented sequentially to the network together with the corresponding output vectors identifying their categories as shown in table 4.2. Unlike the network structure used in S-LVQ, [Natschlager and Ruf, 1998] and [Bohte, Poutre and Kok, 2000], the proposed structure helps to reduce the complexity of the connections where the multiple synaptic delays only exist between the input and hidden neurons. There were six output neurons, one for each pattern category, and six

hidden neurons (the number of hidden neurons here depends on the number of classes).

Only single connections between the hidden and output neurons and the weights were fixed to 1. This reduced the number of weights that had to be adjusted since only the connections between the input and hidden neurons had multiple synaptic terminals. Generally, the NS-LVQ network adopted the spiking neuron models from S-LVQ. These two models, S-LVQ and NS-LVQ were based on the Spike Response Model [Gerstner and Kistler, 2002] with some tuning in the parameter used for NS-LVQ model in order for the network to be applied to control chart pattern recognition. The spike response function used in this architecture is exactly the same as in Chapter 3 in equation (25). The only difference is the value of the parameter for membrane time constant *tce,* which is 150 (ms). The synapse time constant *tci* used in this spike response function is 20 (ms). Here, st is equal to $(t - t_i - d^k)$ as in S-LVQ where t is the simulating time (0 to 300), t_i is the firing time of pre-synaptic neurons and d^k represents the delay with $k = 16$. Other parameters in this spike response function are the same as in S-LVQ in Chapter 3.

Number of inputs = 60	Number of outputs = 6
Number of hidden neuron for each output category = 1	Initial weights range = 0 to 1
Scaling range = 0 to 1	Coding interval = 0 to 100
Learning rate = 0.0075	Delay intervals = 15 (ms) in 10 (ms) interval
Synaptic delays = 1 to 16 (ms)	Time constant = 150 (ms)

Table 4.1: Details of the proposed NS-LVQ network used for control charts

Pattern	Outputs					
	1	2	3	4	5	6
Normal	1	0	0	0	0	0
Increasing trend	0	1	0	0	0	0
Decreasing trend	0	0	1	0	0	0
Upward shift	0	0	0	1	0	0
Downward shift	0	0	0	0	1	0
Cycle	0	0	0	0	0	1

Table 4.2: Representation of the output categories

Unlike the network structure used in the traditional neural network, the proposed structure has different features:

(i) Using spiking neurons instead of the common neurons;

(ii) Multi-synapse terminals instead of a single reference vector between input layer and hidden layer;

(iii) Each multi-synapse terminal has delay and weight instead of weight only for each reference vector;

(iv) The proposed network has a fewer neurons.

Compared to the structure of the S-LVQ network, the NS-LVQ structure has been modified to be simpler than the structure of the traditional LVQ, where only one hidden neuron is used to represent each category of patterns.

Figure 4.1 and Figure 4.2 show the architecture of the proposed network. Figure 6 demonstrates the multi-synapse terminals for the NS-LVQ network. The different layers are labelled as input, hidden, and output layer respectively as shown in Figure 4.1. It is assumed that any neuron can generate at most one spike during the simulation interval and discharges when the internal state variable reaches a threshold.

4.4 Setting the Weights, Delays and Threshold

One of the most important roles in a learning process is initialising the network. The important parameters in the networks are the weights, delays and also the threshold value. The connection weights are allowed in the range [0, 1] and the delays in the range of {1, ..., 15} [ms] in 10 ms intervals. This value was chosen after to give the

best performance after a few experiments. The available synaptic delays were therefore 1-16 (ms).

In the proposed network, in order to ensure that no neuron dominates all the other neurons for all or most of the input pattern, a pre-process weights technique is proposed. The range of the weights and delays are exactly the same as for the S-LVQ algorithm in Chapter 3.

The selection of the threshold value for a neuron needs more concentration as too low a value will force the neuron to fire prematurely without reflecting the entire input pattern. On the other hand, too high a value will prevent the neurons from firing and will block the learning process because a neuron can learn only when it is active. Therefore, the threshold value should be set to an appropriate value to capture the effect of all or most of the inputs while ensuring that it fires.

A suitable threshold value in this proposed algorithm is as the following equation:

$$Threshold = \frac{(Inputs)\ x\ (Num\ of\ sub\text{-}connection)\ x\ (Weights\ average)}{A\ constant\ value}$$

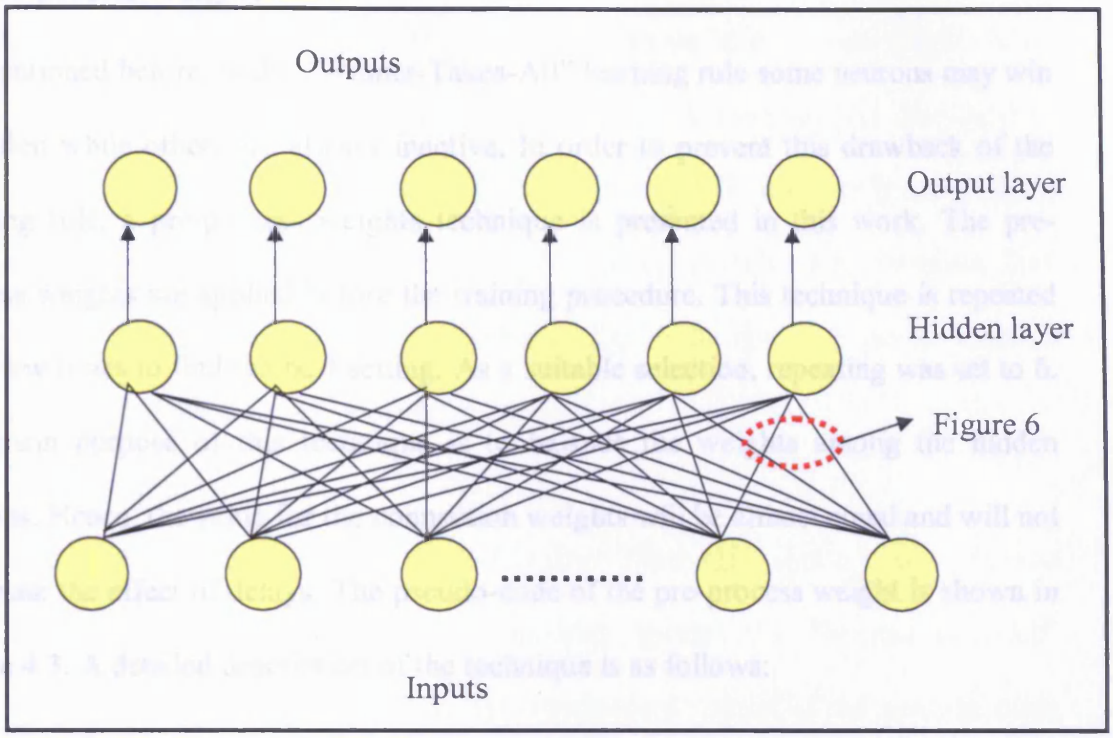

Figure 4.1: A structure proposed for the NS-LVQ network

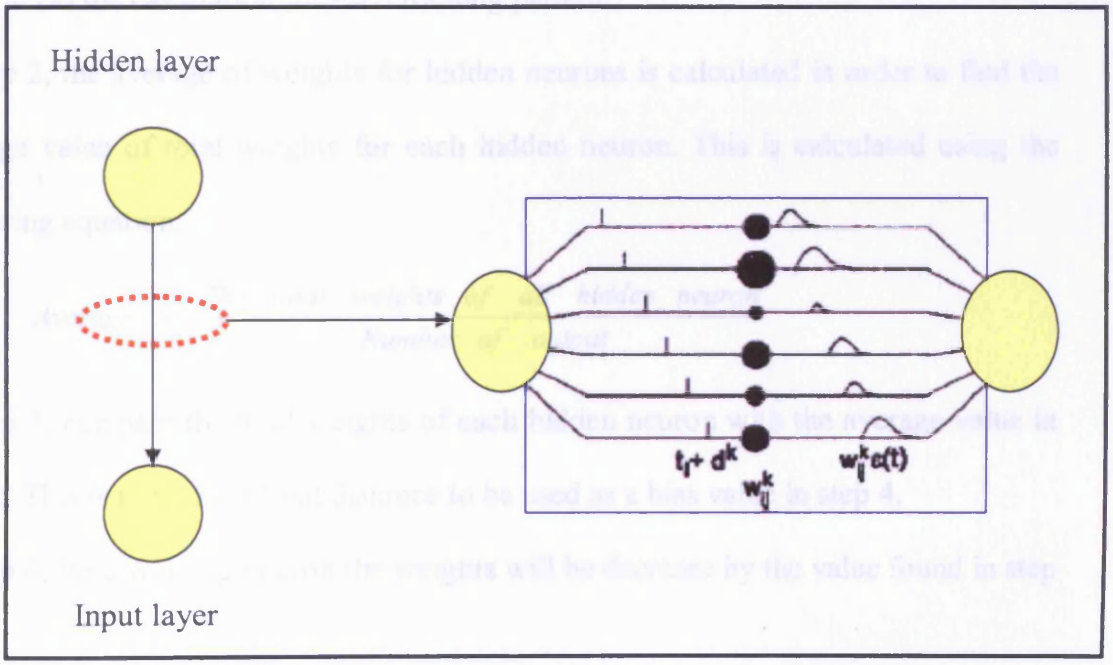

Figure 4.2: Multi-synapse terminals for the NS-LVQ network

4.5 Pre-process Weights

As mentioned before, in the "Winner-Takes-All" learning rule some neurons may win too often while others are always inactive. In order to prevent this drawback of the learning rule, a pre-process weights technique is presented in this work. The pre-process weights are applied before the training procedure. This technique is repeated for a few times to find the best setting. As a suitable selection, repeating was set to 6. The main purpose of this technique is to balance the weights among the hidden neurons. Hence, the value for the connection weights will be almost equal and will not dominate the effect of delays. The pseudo-code of the pre-process weight is shown in Figure 4.3. A detailed description of the technique is as follows:

In step 1, for each hidden neuron, do the summation of the weights for the sub-connections for that particular neuron. This will give the total weights for each hidden neuron. Do the calculation for each training pattern.

In step 2, the average of weights for hidden neurons is calculated in order to find the average value of total weights for each hidden neuron. This is calculated using the following equation:

$$Average = \frac{The\ \ total\ \ weights\ \ of\ \ all\ \ hidden\ \ neuron}{Number\ \ of\ \ output}$$

In step 3, compare the total weights of each hidden neuron with the average value in step 2. This is to find the final distance to be used as a bias value in step 4.

In step 4, for a winning neuron the weights will be decrease by the value found in step 3.

A step 5 is repeating the procedure until a certain time or until a stopping criterion is met. In this work, the procedure is repeated for 6 times.

4.6 NS-LVQ Learning Procedure

As mentioned in sub-section 3.9.2.1 in Chapter 3, there are, at most, two weights to be modified for the LVQ and its variant including LVQ-X methods. As discussed in sections 3.3 to 3.6, the main problem with the standard LVQ is that only the winning neuron is permitted to modify the connection weights in each learning iteration. This is so-called "Winner-Takes-All" competition. This will result in some neurons winning too often while others are always inactive.

Inspired by the fast computation of the "Winner-Take-All" learning rule [Jin and Seung, 2002], the proposed NS-LVQ algorithm applies the "Winner-Take-All" learning rule. Taking into account that the membrane potential of the neurons often deviates from the resting membrane potential because of noise and the inputs from other brain areas, the proposed network also applies the boosting technique as applied in the S-LVQ algorithm in Chapter 3. Moreover, this technique has been proved by S-LVQ to contribute to better classification.

Procedure for Pre-process Weights

1) Calculate the **total weights** for each hidden neuron;

2) Calculate the **Average** of weights for hidden neuron;

3) **For** each hidden neuron **Do**

    If the (total weight > Average) **Then**

    Distance = (total weights – Average)

    Else

    Distance = (Average - total weights)

    Final distance = Distance / (Number of input)*Number subconnection

   End For;

4) **If** a hidden neuron is the winner **Then**

    Weight = Weight – Final distance for that particular hidden neuron
    Else
    Weight = Weight + Final distance for that particular hidden neuron;

5) **Repeat** this procedure (1) to (4) until a certain time or until a stopping criterion is met.

Figure 4.3: A pseudo-code description for pre-process weight

The technique of boosting applied in this network is similar to that applied for the S-LVQ network. The Winner- Takes-All learning rule applied here will modify the weights between the input neurons and the neuron first to fire (winning neuron) in the hidden layer. The winner will be activated to 1 and the others to 0. In this learning procedure, only if the winning neuron is in the correct category and the start of the PSP at a synapse slightly precedes a spike in the target neuron, is the weight of this synapse increased, as it exerts a significant influence on the spike-time by virtue of a relatively large contribution to the membrane potential. A pseudo-code description for updating weight of S-LVQ is given in Figure 4.3. The details of the pseudo-code are as follows:

The network structure needs to be defined. This involves deciding on the number of inputs, outputs, and the number of sub-connections, the learning rate, and time constant parameter. The chosen values for theses parameters were mentioned in sub-section 4.3. SNN is a complex network model with a number of parameters which control its functionality. Tuning the network by assigning appropriate values for these parameters is essential for the smooth functioning of the network and for obtaining optimum performance. There is a lack of clear guidelines regarding the selection of these parameters. Hence, in this work, these parameters were found by analysing the preliminary results obtained with some initial trial values.

In step 2, initialising the network plays an important role in the learning process. The important parameters here are the weights and delays of the interconnections between the input neurons and the output neurons and also the threshold. The chosen values for these two parameters have been mentioned in section 4.4. In the proposed model,

the connection weights are assigned only with positive values and all the connections are realised as excitatory, with positive spike response function $\varepsilon(t) \in [0.1]$. This is to ensure that the effect of most of the input parameters contributes positively to the output.

In step 3, another important aspect of implementing the model is addressed, namely the temporal coding the continuous input values. Precision of the temporal code should be selected in such a way as to attain adequate accuracy with optimal computational efficiency. Experiments have revealed that an input time window with 100 units is adequate in the case of control chart data sets.

In step 4, a training input pattern will be presented to the network sequentially.

In step 5, the simulation time (t) in this work is selected at from 0 to 300 units. For each t, the network is updating the synapse potential for the training input pattern. Then followed the updating of the output. Here is where the potential neurons of the class belonging to that particular training input are raised.

Procedure Training For NS-LVQ Algorithm

1) Define the network structure ;
2) Initialise the weights and the delays;
3) Encode the input data into temporal coding;
4) Present a training input pattern to the network;
5) **For** each t (simulation time) **Do**
    i) **Update the synapse potential ;**
    ii) **Update the output (boosting);**

    While (t < time window)
6) Find the winner;
7) Update the weights as the following:
8) **Return** to (4).

Figure 4.4: A pseudo-code description for NS-LVQ Algorithm

In step 6, the system will determine the first neuron to fire. The first neuron to fire is determined as the winner.

In step 7, the weight of the winning neuron only will be modified or updated. If the winning neuron is in the correct category and $\Delta T > 0$, the neurons will be updated using equation (29) or $\Delta T < 0$, the equation (30) is used. If the winning neuron is in the incorrect category for $\Delta T > 0$ or $\Delta T < 0$, then equation (31) is used.

In step 8, the system will return to step 4 with a new training input pattern and repeat the procedure until all training patterns are correctly classified (or a stopping criterion is met).

In this research, the unsupervised learning equations in (24) were employed to create a supervised learning equation using the following update equations:

If the winner is in the correct category, then

$$w_{new} = w_{old} + dw \quad \text{where}$$

$$dw = \left(\eta \left(\frac{1}{\beta\lambda} \right) e^{-\left(\frac{\Delta T}{2\beta^2} \right)} \right) for \Delta T > 0 \quad \text{or} \qquad (29)$$

$$dw = \left(-\eta \left(\frac{1}{\beta\lambda} \right) e^{-\left(\frac{\Delta T}{2\beta^2} \right)} \right) for \Delta T < 0 \qquad (30)$$

If the winner is in the incorrect category, then

$w_{new} = w_{old} - dw$ where

$$dw = \left(\alpha\eta \left(\frac{1}{\beta\lambda} \right) e^{-\left(\frac{\Delta T}{2\beta^2} \right)} \right) \qquad \text{for} \quad \Delta T > 0 \ \ or \ \ \Delta T < 0 \qquad (31)$$

In the simulation, the parameter values for the learning function $L(\Delta T)$ were set to: η = 0.0075, β = 35, $\lambda = \sqrt{(2 * (22/7))}$, ΔT = [0-100], α = 0.8, w_{new} is the new value for the weight and w_{old} is the old weight value. The parameter η used here is a static learning rate. Parameter β sets the width of the positive part of the learning window and ΔT denotes the time difference between the onset of a PSP at a synaptic terminal and the time of the spike generated in the winning output neuron. Parameter α was used because in supervised learning there is prior information about the training sets.

4.7 Data Set

The same data set as used in Chapter 3 is used here in order to make a comparison of the network. There were 1500 data sets generated, using the process simulator mentioned above. From these data sets, 1002 were used as training patterns (167 patterns in each category) and 498 (83 patterns in each category) for the testing patterns. The patterns were sequentially applied to the network.

4.8 Empirical Evaluation of NS-LVQ

This section presents an empirical evaluation of the control chart pattern recognition performance with the NS-LVQ algorithm. Two criteria were used to evaluate the performance of the tested algorithm, namely, number of training epochs and the classification accuracy. The number of epochs determine the training time. The performance of the network is calculated based on the classification accuracy. The calculation of the accuracy was made using the following equation:

$$Accuracy\ (\%) = \frac{Number\ of\ patterns\ correctly\ classified}{Total\ number\ of\ patterns\ tested}\ X\ 100$$

The results obtained with the proposed architecture and the supervised learning procedure for control chart pattern recognition are presented in Table 4.3. A comparison was made with the results obtained with an LVQ network [Pham and Oztemel, 1994] and a back-propagation neural network.

Pattern recogniser	Num. of training epochs	Learning performance (%)	Test performance (%)
LVQ-X	20	100.00	97.70
Back-propagation	200	100.00	95.00
S-LVQ	40	99.85	98.28
NS-LVQ	15	99.93	97.85

Table 4.3: Results of different pattern recognisers applied to control chart data set.

4.9 Comparison with S-LVQ and Traditional Neural Networks

In Bohte et al's work, a large set of weights have to be adjusted since a connection between two neurons corresponds to sixteen sub-connections, so the size of the network increases drastically with the number of neurons. Furthermore, the network is fully connected. Modifications were made to suit the spiking learning vector quantisation (S-LVQ) used for a much simpler network. Compared to S-LVQ, the resulting network has a shorter training time although the classification accuracy of S-LVQ is slightly better. The resulting neural network has a simple structure of spiking neurons for control chart data, reducing the complexity of implementation. As mentioned in Chapter 3, the resulting network has no dependency on the initial values of the weights. The training and adaptation time of the resulting spiking network clearly was shorter than that for LVQ and back-propagation networks [Pham and Oztemel, 1992; Pham and Oztemel, 1994]. At the end of 10 training epochs, the network was able to classify correctly 98.73% of the training data set and 95.89% of the test set. After 15 training epochs, the overall recognition accuracy level was increased to 99.93% for the training set and 97.85% for the test set. This shows clearly the superior performance of the spiking neural networks technique in an application to control chart data over the other procedures using traditional neural networks.

4.10 Learning Parameter (η)

As mentioned in the above chapter, SNNS is a complex model which involves more parameters. Choosing the suitable parameters will affect the efficiency of the recogniser. Another important parameter to be identified is the learning parameter η . There is no way to decide it theoretically. The maximum allowable static value of the learning parameter can be obtained empirically. However, there are two techniques to determine the most suitable learning value. There are:

1) Static learning rate;
2) Adaptive learning rate.

4.10.1 Static Learning Rate

Static learning rate means that the value is fixed for all iterations. Experiments showed that the classification accuracy with a small learning parameter is better and the learning time is faster than with a larger learning rate. The smaller learning rate is in the range of [0.0025, 0.01] and the larger is in the range of [0.05-0.1]. In this thesis, the reasonable value of learning rate that give the best performance is 0.0075.

4.9.2 Adaptive Learning Rate

The adaptive learning rate monotonically decreases with time *t*. Different functions could be adopted to implement $\eta(t)$, including the exponential decay function as in Figure 4.4 and the linear decay function as in Figure 4.5. The allowed value for every step in both functions is 0.003. The learning rate value is updated at each learning iteration.

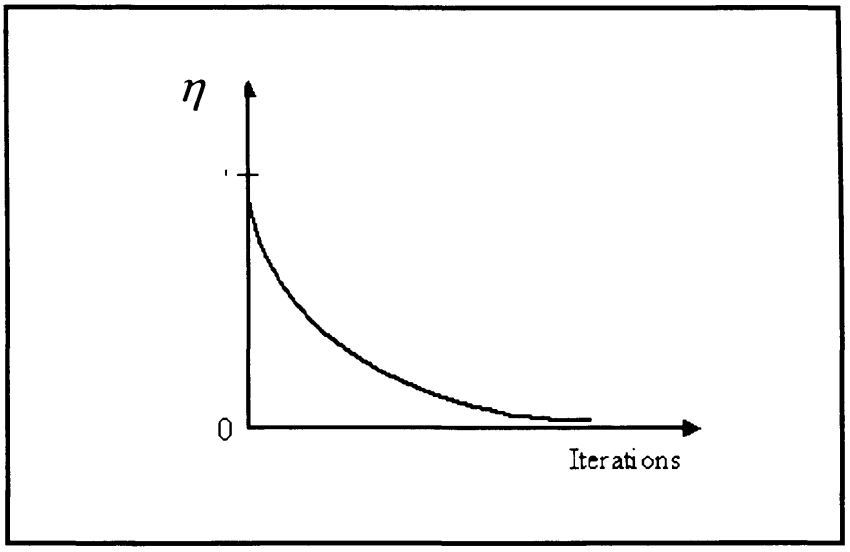

Figure 4.5: Exponential decay function

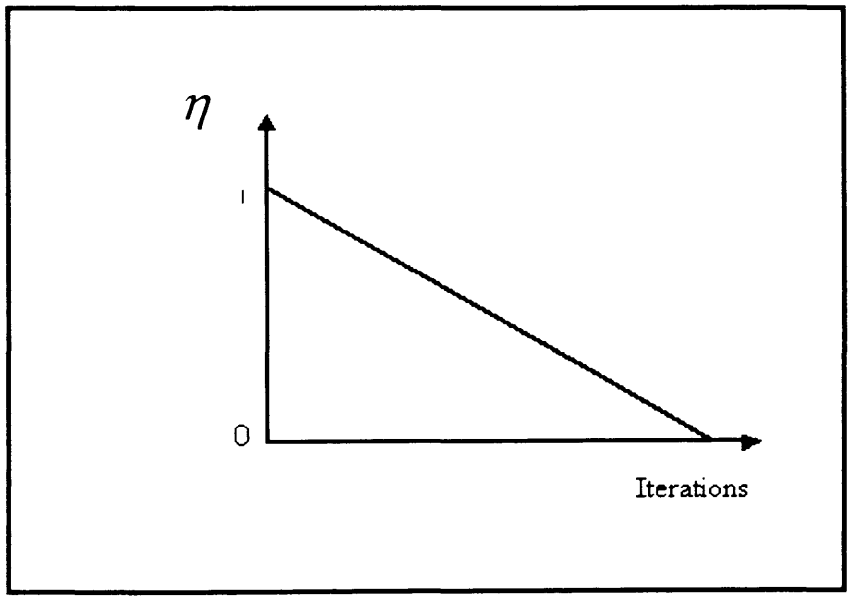

Figure 4.6: Linear decay function

Learning parameter η	Number of iterations			
	5	10	15	20
0.0025	91.83	97.87	98.21	98.72
0.0050	91.72	97.14	97.84	97.84
0.0075	93.06	97.23	97.85	97.85
0.01	93.77	97.13	97.74	95.42
0.05	93.63	96.46	97.39	94.68
0.1	90.33	94.72	95.25	96.36

Table 4.4: Testing accuracy for different values of static learning rate

Epochs	Testing accuracy with exponential decay [0.0025-0.01]	Testing accuracy with linear decay [0.0025-0.01]
5	94.26 %	92.35%
10	96.56%	95.67%
15	95.83%	95.67%
20	93.17%	94.33%

Table 4.5: Comparison of training accuracy based on two different types of adaptive learning rate.

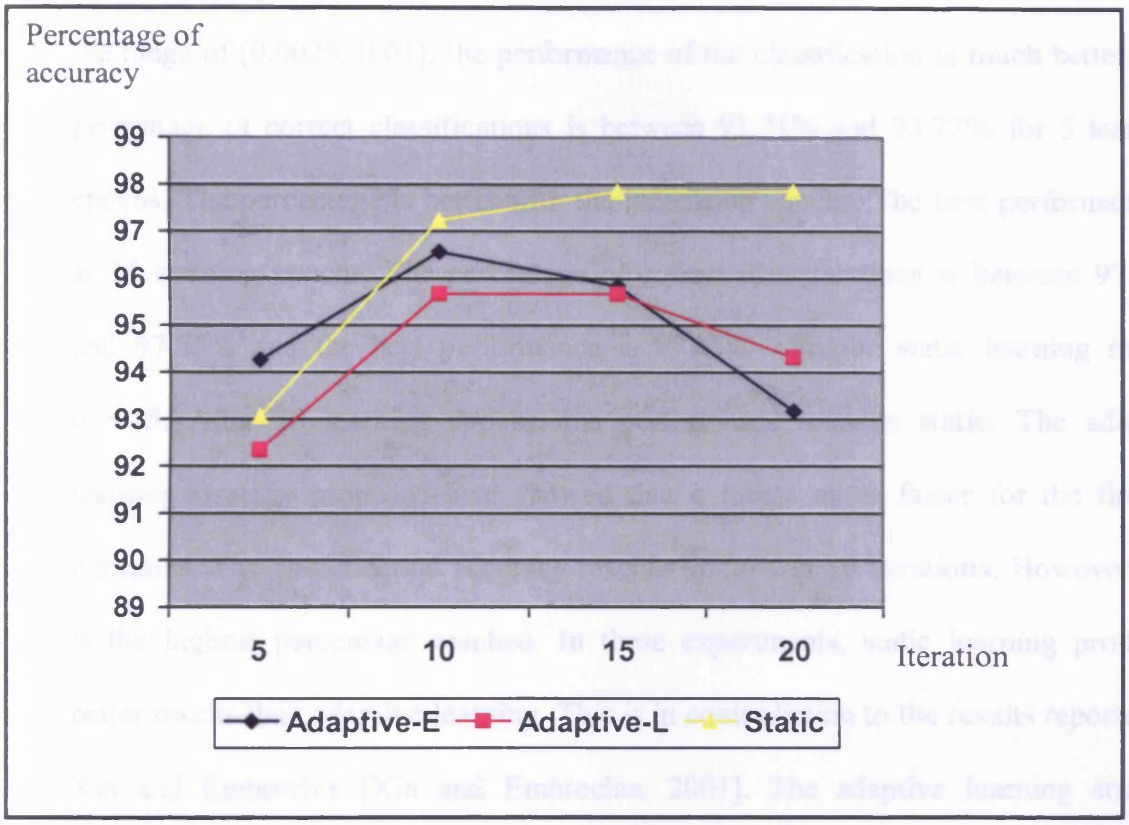

Figure 4.7: Adaptive vs. static learning for classification accuracy

4.10.3 Static Vs Adaptive Learning Rate

Tables 4.4 and 4.5 show the results for control chart data using static learning rate and adaptive learning rate respectively. Figure 4.7 presents the results clearly in a graphical way. Table 4.6 shows the results of different cases of static value. From the results shown in table 4.6, it is clear that with a small learning parameter, which is in the range of [0.0025, 0.01], the performance of the classification is much better. The percentage of correct classifications is between 91.71% and 93.77% for 5 learning epochs. The percentage is better with the increasing epochs. The best performance is at 15 learning epochs. The percentage of correct classifications is between 97.74% and 97.85% and the best performance is 97.85% with the static learning rate at 0.0075. After 20 learning epochs, the performance remains static. The adaptive learning strategy proposed here showed that it learns much faster for the first 10 iterations. The classification accuracy reaches 96.56% at 10 iterations. However, this is the highest percentage reached. In these experiments, static learning produced better results than adaptive learning. This is in contradiction to the results reported by Xin and Embrechts [Xin and Embrechts, 2001]. The adaptive learning strategy proposed here can be improved in future to get better results.

4.11 Summary

This chapter has presented an enhanced spiking learning vector quantisation (S-LVQ) algorithm, so-called NS-LVQ, particularly applied for control chart patterns. Here, a new architecture and a technique for pre-process weights were proposed. Previously, neural networks have proved capable of data smoothing and generalisation. This research has shown that spiking neural networks with a simpler architecture together

with an efficient learning rule can produce good capability in data smoothing and generalisation. This permits them to recognise noisy control chart patterns not identical to those they have been taught, as indicated by the good results presented in this work. Some improvement in the adaptive strategy proposed in this chapter will enable better classification accuracy.

CHAPTER 5

OPTIMISATION OF SPIKING NEURAL NETWORKS USING THE BEES ALGORITHM

5.1 Preliminaries

Generally, an optimisation algorithm is defined as a numerical method or algorithm for finding a value x such that $f(x)$ is as small (or as large) as possible, for a given function f, possibly with some constraints on x. Here, x can be a scalar or vector of continuous or discrete values. If x is continuous, then the study of the algorithm is part of numerical analysis. However, classical optimisation methods encounter great difficulty when faced with the challenge of solving hard problems with acceptable levels of time and precision. It is generally believed that NP-hard problems cannot be solved to optimality within polynomial bounded computation times, thus generating much interest in approximation algorithms that find near-optimal solutions within reasonable running times.

Over the past several years, researchers have been inspired by nature in many different ways. Swarm-based optimisation algorithms (SOAs) mimic nature's methods to drive a search towards the optimal solution. They often provide state-of-the-art solutions for hard optimization problems. They have been demonstrated to be efficient algorithms for tracking or finding the optimal solution in the dynamic

environment and have received increasing interest. Successful optimisation methods play a crucial role for finding optimal or near-optimal solutions for such problems.

5.2 Intelligent Swarm-based Optimisation Algorithms (SOAs)

Swarm intelligence is an innovative computational way to solve hard problems. Swarm intelligence (SI) is an artificial intelligence technique based around the study of collective behaviour in decentralised, self-organised systems. The expression "swarm intelligence" was introduced by Beni & Wang in 1989 [Beni and Wang, 1989], in the context of cellular robotics systems.

This discipline is mostly inspired by the behaviour of swarms of ants, termites, bees, wasps, fishes and other biological creatures. In general, there is some kind of mimicking of the behaviour of these swarms. SI systems are typically made up of a population of simple agents interacting locally with one another and with their environment. Although there is normally no centralized control structure dictating how individual agents should behave, local interactions between such agents often lead to the emergence of global behavior. Swarm Intelligence is a relatively novel discipline devoted to the study of self-organizing collective processes in Nature and Human artefacts as well as on their applications.

A key difference between SOAs and direct search algorithms such as hill climbing and random walk is that SOAs use a population of solutions for every iteration instead of producing a single solution. As a population of solutions is processed in iteration, the outcome of each iteration is also a population of solutions. If an optimisation problem has a single optimum, SOA population members can be expected to

converge to that optimum solution. However, if an optimisation problem has multiple optimal solutions, an SOA can be used to capture them in its final population.

Among the most popular SOAs are the Ant Colony Optimisation (ACO) algorithm [Dorigo and Stutzle, 2004], the Genetic Algorithm (GA) [Goldberg, 1989] and the Particle Swarm Optimisation (PSO) algorithm [Eberhart, Shi, and Kennedy, 2001].

An example of a particularly successful research direction in swarm intelligence is ant colony optimisation (ACO), which focuses on discrete optimisation problems, and has been applied successfully to a large number of hard discrete optimisation problems, including the travelling salesman, the quadratic assignment, scheduling, vehicle routing, etc., as well as to routing in telecommunication networks.

However, apart from these remarkably successful applications in optimisation as well as on their critical features as bio-inspired computational paradigms, a small number of research works have still been devoted to Pattern Recognition , Data Classification and Retrieval Systems, Clustering, Distributed Data-Mining, Web Mining and GRIDS, Collaborative Filtering, Image Analysis and Signal Processing, Pattern Formation, Perception, Memory and Generalisation.

In the real world one usually has to deal with the task of searching for an optimal solution in a dynamic environment. Because of the continual change of both the external environment and parameters, the optimum solution will also change with time. In contrast to the static case, the main goal in dynamic optimisation problems is no longer to acquire just the global extreme but to track its orbit through space as closely as possible, or to find a robust solution that operates optimally in the presence of uncertainties. Many algorithms fail when applied to the dynamic problem due to

their inability to adapt or self-adapt to the change of the environment. Currently, a swarm-based algorithm, the so called Bees Algorithm (BA), is claimed to be capable of locating good solutions efficiently [Pham et al, 2006].

This algorithm is inspired by the behaviour of honey bees [Pham et al, 2005]. There are other SOAs with names suggestive of possibly bee-inspired operations [Frisch, 1976; Seeley, 1996, Bonabeau, 1999, and Camazine et al, 2003]. However, as far as the author is aware, those algorithms do not closely follow the behaviour of bees. In particular, they do not seem to implement the techniques that bees employ when foraging for food.

5.3 The Basic Bees Algorithm

5.3.1 Honey Bees in Nature

A colony of honey bees can extend itself over long distances (more than 10 km) and in multiple directions simultaneously to exploit a large number of food sources [Frisch, 1976; Seeley, 1996]. A colony prospers by deploying its foragers to good fields. In principle, flower patches with plentiful amounts of nectar or pollen that can be collected with less effort should be visited by more bees, whereas patches with less nectar or pollen should receive fewer bees [Bonabeau, 1999, and Camazine et al, 2003].

The foraging process begins in a colony by scout bees being sent to search for promising flower patches. Scout bees move randomly from one patch to another. During the harvesting season, a colony continues its exploration, keeping a percentage of the population as scout bees [Seeley, 1996]. When they return to the hive, those scout bees that found a patch which is rated above a certain quality threshold (measured as a combination of some constituents, such as sugar content) deposit their nectar or pollen and go to the "dance floor" to perform a dance known as the "waggle dance" [Frisch, 1976].

This mysterious dance is essential for colony communication, and contains three pieces of information regarding a flower patch: the direction in which it will be found, its distance from the hive and its quality rating (or fitness) [Frisch, 1976; Camazine et al, 2003]. This information helps the colony to send its bees to flower patches precisely, without using guides or maps. Each individual's knowledge of the outside environment is gleaned solely from the waggle dance.

This dance enables the colony to evaluate the relative merit of different patches according to both the quality of the food they provide and the amount of energy needed to harvest it [Camazine et al, 2003]. After waggle dancing on the dance floor, the dancer (i.e. the scout bee) goes back to the flower patch with follower bees that were waiting inside the hive. More follower bees are sent to more promising patches. This allows the colony to gather food quickly and efficiently.

While harvesting from a patch, the bees monitor its food level. This is necessary to decide upon the next waggle dance when they return to the hive [Camazine et al,

2003]. If the patch is still good enough as a food source, then it will be advertised in the waggle dance and more bees will be recruited to that source.

5.3.2 Bees Algorithm

As mentioned above, the Bees Algorithm finds the optimal solution to a problem by copying the natural foraging behaviour of honey bees [Pham et al, 2005]. Figure 5.1 shows the flowchart for the basic Bees Algorithm in its simplest form [Pham et al, 2005]. Figure 5.2 illustrated the Bees Algorithm in a simple but attractive graphical representation [Pham et al, 2005]. The details of the Bees Algorithm will be described in the next section, with an addition to it.

5.3.3 Characteristics of Bees Algorithm

Population (n) is one of the key parameters in the Bees Algorithm. Pham et al., 2005 presented an experiment designed to measure the effect of changing population size on the mean number of iterations, number of evaluated points and reliability of successfulness of the algorithm. Experiments proved that increasing the population size will result in a reduced number of iterations. Elitism (e) is another important parameter. The number of elites does not have a major effect on the performance of the Bees Algorithm. Thus, the number of elites can be a small number more than zero.

Neighbourhood search is an essential concept for all evolutionary algorithms including the Bees Algorithm. In the Bees Algorithm, the searching process in the neighbourhood range is similar to the foraging field exploitation of natural bees.

In the Bees Algorithm, the natural behaviour of bees to find quality nectar has been used as a neighbourhood search. Only one bee is chosen from each neighbourhood site (foraging site). This bee has the best solution information about the field. Neighbourhood search is based on the random distribution of bees in a predefined neighbourhood range. For every selected site, bees are randomly distributed to find a better solution. The number of recruited bees around selected sites should be defined properly. Experiment has shown that increasing the number will result in increasing the probability of finding a good solution. This problem also depends on the neighbourhood range. Neighbourhood range is another variable which needs to be tuned for different types of problem spaces.

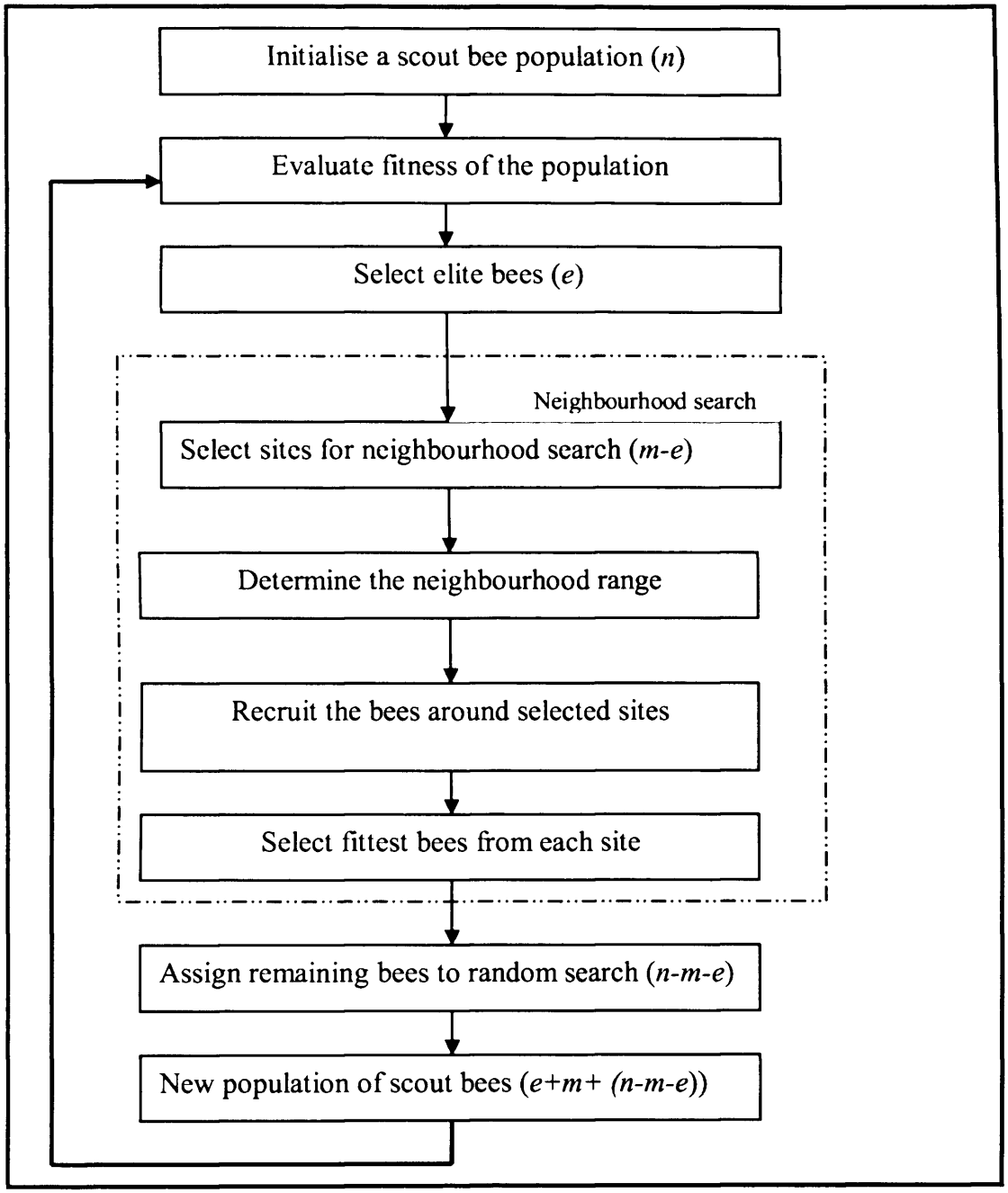

Figure 5.1: Flowchart of Bees Algorithm

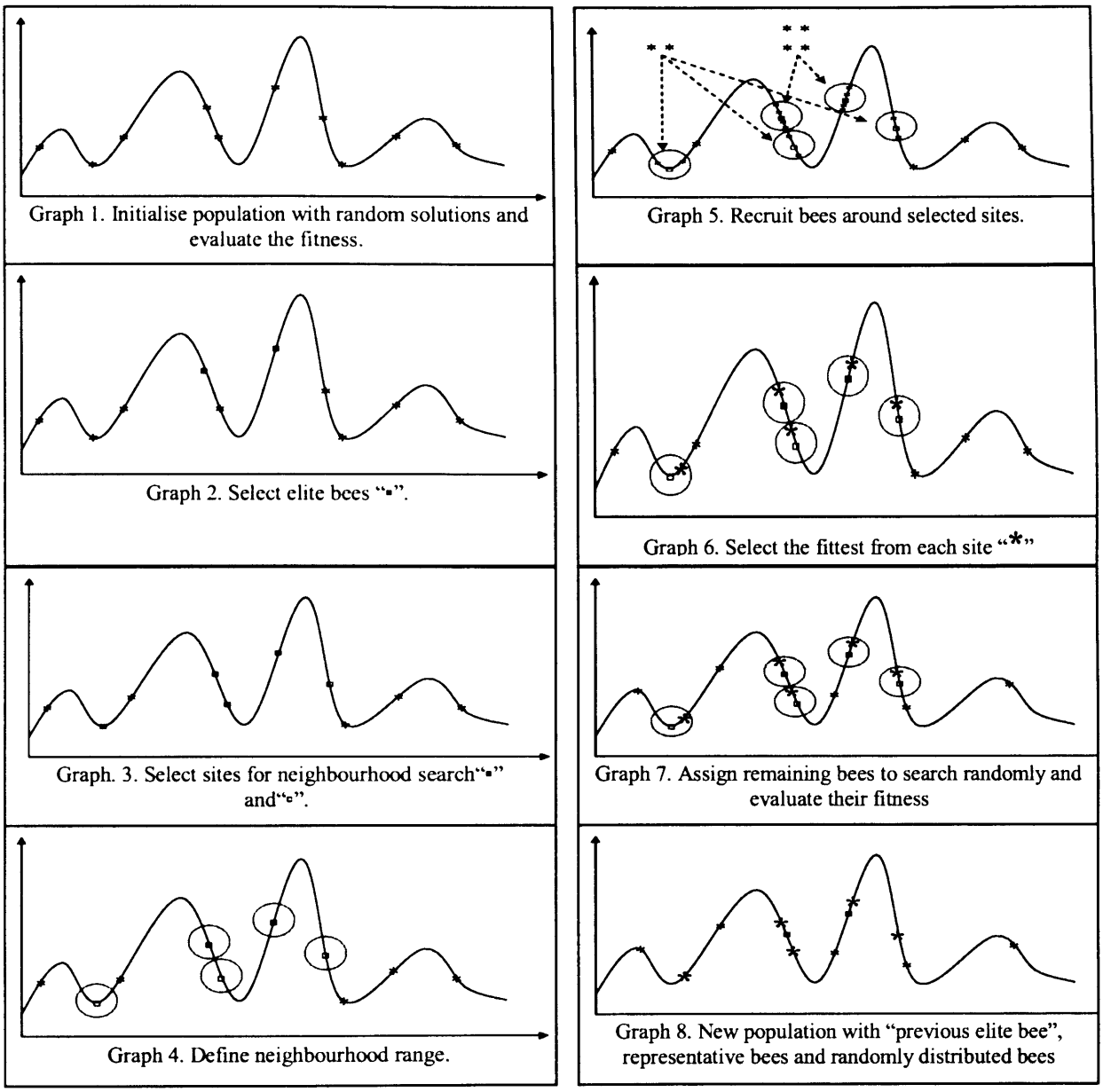

Figure.5.2: Graphical illustration of bee algorithm.

Therefore, it is necessary to discuss different strategies for increasing the robustness and quality of the algorithm. In this chapter, a shrinking method is applied to the Bees Algorithm in order to get better result and to speed the computation time. Details of the proposed method are discussed in following section.

5.4 Bees in Artificial Neural Networks

Optimising the topology of artificial neural networks is an important task, when one aims to get smaller and faster networks, as well as a better generalisation performance. Moreover, optimisation automatically avoids the time-consuming search for a suitable topology. The main criterion for optimising the network topology is the size of the network. The time needed for the optimisation and the classification accuracy are also important.

The Bees Algorithm has shown good performance in its application to neural networks for control chart pattern recognition. Architectures used are Learning Vector Quantisation (LVQ), Radial Basis Function (RBF), and Multi-Layer Peceptron (MLP) [Pham et al., 2006]. These papers have described the use of the Bees Algorithm to train the network for control chart pattern recognition

In terms of the Bees Algorithm, each bee represents an LVQ network with a particular set of reference vectors. The aim of the algorithm is to find the bee with the set of reference vectors producing the smallest value of the error function. The LVQ networks adopted had 60 input neurons in the input layer, 6 output neurons, and 36 hidden neurons. The algorithm was initialised with all weight values set randomly within the range 0 to 1. The classification accuracy levels achieved after 4000 iterations were 96.56% for the training data and 95.47% for the test data. The authors reported that despite the high dimensionality of the problem,(each bee represented 2160 (60X36) parameters that had to be determined) the algorithm still succeeded in training more accurate classifiers than those produced by the standard LVQ training algorithm.

[Pham et al., 2006] explained both the standard RBF training method and a training procedure based on the Bees Algorithm. In this algorithm, each bee represents an RBF network with a particular set of basis function centres, spreads and weight vectors. The aim of the algorithm is to find the bee producing the smallest value of the error function. The RBF network configuration used involves three layers: an input layer, a hidden layer and an output layer. The input layer has 60 neurons, one for each point in a pattern. The hidden layer consists of 35 neurons. The output layer comprises 6 neurons, one for each of the six classes. They reported that despite the high dimensionality of the problem (each bee represented 2345 (60*35+6*35+35) parameters that had to be determined), the algorithm still succeeded in training the network with 99.1% classification accuracy and their results are comparable with those given by conventional RBF networks.

For MLP networks, each bee represents an MLP network with a particular set of weight vectors. The aim of the algorithm is to find the bee with the set of weight vectors producing the smallest value of the error function. MLP networks are trained with the Bees Algorithm as well as with the standard back-propagation algorithm. The multi-layer perceptron (MLP) configuration used involves three layers: an input layer, a hidden layer and an output layer. The input layer has 60 neurons, one for each point in a pattern. The output layer comprises 6 neurons and the hidden layer consists of 35 neurons. The input neurons perform no processing roles, acting only as buffers for the input signals. Processing is carried out by the hidden and output neurons, the activation functions for which were chosen to be of the sigmoidal type. In addition, the algorithm was initialised with all weight values set randomly within the range -1 to 1. The authors reported that at the end of 1000 iterations, the MLP network was able correctly to classify 98.2% of the training set and 96.9% of the test set. The network was also here applied to a high dimensionality of the problem as each bee represented 2310 (60*35+35*6) parameters that had to be determined. Their results demonstrated that this algorithm outperformed the back-propagation algorithm.

Table 5.1 shows the values of the parameters adopted for the Bees Algorithm for the three recognisers. The values were decided empirically. The performance of the ANNs mentioned above is shown in Table 5.2.

Bees Algorithm parameter	Artificial neural networks architecture		
	LVQ	RBF	MLP
Population (*n*)	200	200	200
Number of selected sites (*m*)	20	10	20
Number of elite sites (*e*)	1	2	2
Initial patch size (*ngh*)	0.01	0.1	0.1
Number of bees around elite points (*nep*)	20	80	50
Number of bees around other selected points (*nsp*)	10	20	20

Table 5.1: The parameters of the Bees Algorithm for LVQ, RBF and MLP for control chart pattern recognition

Pattern recogniser with Bees	Learning accuracy	Test accuracy
LVQ	96.56%	95.47%
RBF	99.59%	99.10%
MLP	98.2%	96.9%

Table 5.2: Performance of different pattern recognisers with Bees

5.5 Evaluation Strategy (ES) in SNNs

In [Belatreche et al., 2003], the authors investigate the viability of evolutionary strategies (ES) regarding supervised learning in spiking neural networks. The use of the evolutionary strategy is motivated by the ability of ESs to work on real numbers without complex binary encoding schemes. ESs proved to be well suited for solving continuous problems [Spears et al., 1993]. The ESs are used to search for the optimum weights and delays that minimise the total error between actual and target output firing times. The objective function to be minimised is given by equation (32):

$$E = \sum_{t}^{T} \sum_{o \in 0} (t_o^a(t) - t_o^t(t))^2 \qquad (32)$$

where $t_o^a(t)$ and $t_o^t(t)$ are, respectively, actual and target output firing times of node i for pattern t, and T is the total number of patterns in the training set. In their work, a modified ES is used to train the spiking network in a supervised way, where a combination of Cauchy and Gaussian mutation is used.

A feedforward fully connected spiking network is implemented in their work. Basically there are input, hidden and output layers (16X10X1) for IRIS benchmark data. The adopted spiking neurons are based on the Spike Response Model [Gerstner and Kistler, 2002] and are connected via synapse, which are characterised by a weight representing the synaptic strength and a delay for the time a spike takes to reach the post-synaptic neuron. The advantage here is its simple structure. The structure considers only one synaptic connection between two neurons, each of which is

characterised by a weight and a delay value. Moreover, both positive and negative weights are allowed. The approach has been reported as successful for learning nonlinearly separable problems without using any gradient information. The classification accuracy level achieved for the IRIS data set is 98.67% for the training set and 94.67% for the test set. However, a disadvantage of this approach is that it is time-consuming. These results demonstrate that an alternative approach such as optimisation for supervised learning with spiking neural network have a great potential.

5.6 Motivation for Research

Although significant progress has already been made in recognisng information codes that can be beneficial for computation in SNNs [Gerstner and Kistler, 2002a; Maass, 1999; 2003; Maass and Bishop, 1999], how to determine efficient neural learning mechanisms that facilitate the implementation of these particular time coding schemes is still an open problem. Numerical experiment has proved that the performance of the S-LVQ and the NS-LVQ algorithms is superior to that of the standard ANNs. However, experiment using a single synapse for each connection between two neurons showed that further work is needed to improve the performance of the classification achieved. These algorithms still suffer from problems that limit their efficiency and widespread use. One of the main limitations is the method used to set the values of weights and delays. In these algorithms, the value of weights, delays and of the delay interval must be set by trial and error to get the best result.

This section presents S-LVQ and NS-LVQ with the Bees Algorithm, an alternative algorithm which addresses the limitation of the spiking networks. In particular, it employs a new optimisation algorithm, which modifies the basic Bees Algorithm. Generally, the proposed Bees Algorithm applies the shrinking method to narrow the neighbourhood size in order to focus the searching process. This enhancement enables an improvement of performance for both networks.

The remainder of this chapter is organised as follows: section 5.7 gives a detailed description of the proposed Bees Algorithm and the spiking neural network structure; section 5.8 presents the pattern recognition results obtained using the S-LVQ and NS-LVQ networks with the Bees Algorithm.

5.7 Spiking Neural Networks with Proposed Bees Algorithm

5.7.1 Networks Structure

Optimisation using the Bees Algorithm will involve both of the proposed spiking networks presented above in Chapters 3 and 4, the so called SB-LVQ and NSB-LVQ networks respectively. In order to maintain comparability, the structure of the spiking neural networks (number of hidden layers and number of neurons) remained the same throughout. The networks and parameters adopted here are exactly the same. Details of the network structures are presented in Tables 5.1 and 5.2 respectively. Basically, those networks have a similar architecture to both the S-LVQ and the NS-LVQ networks, as explained above in Chapters 3 and 4 respectively. In this work, the adopted spiking neurons are based on the Spike Response Model [Gerstner and Kistler, 2002] with some modification to the spike response function in order for the

networks to be applicable to control chart pattern recognition. The spike response function used in that architecture is as explained in equation (21) in Chapter 3. Input vectors were also presented sequentially to the network together with the corresponding output vectors identifying their categories as shown in Table 3.4 above. Unlike the network structure used for S-LVQ and NS-LVQ in Chapters 3 and 4, this structure considers only one synaptic connection between two neurons, each of which is characterised by a weight and a delay value. Furthermore, this structure is much simpler compared to both previously proposed algorithms as only one synapse is considered. As previous work, only positive values of weights are tested here.

5.7.2 Optimising the Networks

Previously the method presented for refining such a spiking network was the use of the proposed S-LVQ and NS-LVQ algorithms. However, in both of the proposed algorithms, during the training phase, the new learning algorithms are used to adjust the weights. In this thesis, a modified version of Bees Algorithm is used to optimise the weights and the delays of the networks previously developed in Chapters 3 and 4 . The main motive for using the Bees Algorithm is to search for the optimum set of both synaptic weights and delays that allow the spiking network to learn temporal patterns.

Number of inputs = 60	Number of outputs = 6
Number of hidden neuron for each output category = 6	Initial weights range = 0 to 1
Scaling range = 0 to 1	Coding interval = 0 to 100
Learning rate = 0.0075	Time constant = 120 (ms)

Table 5.3: Details of the proposed S-LVQ network used for control charts with the Bees Algorithm.

Number of inputs = 60	Number of outputs = 6
Number of hidden neuron for each output category = 1	Initial weights range = 0 to 1
Scaling range = 0 to 1	Coding interval = 0 to 100
Learning rate = 0.0075	Time constant = 150 (ms)

Table 5.4: Details of the proposed NS-LVQ network used for control charts with the Bees Algorithm.

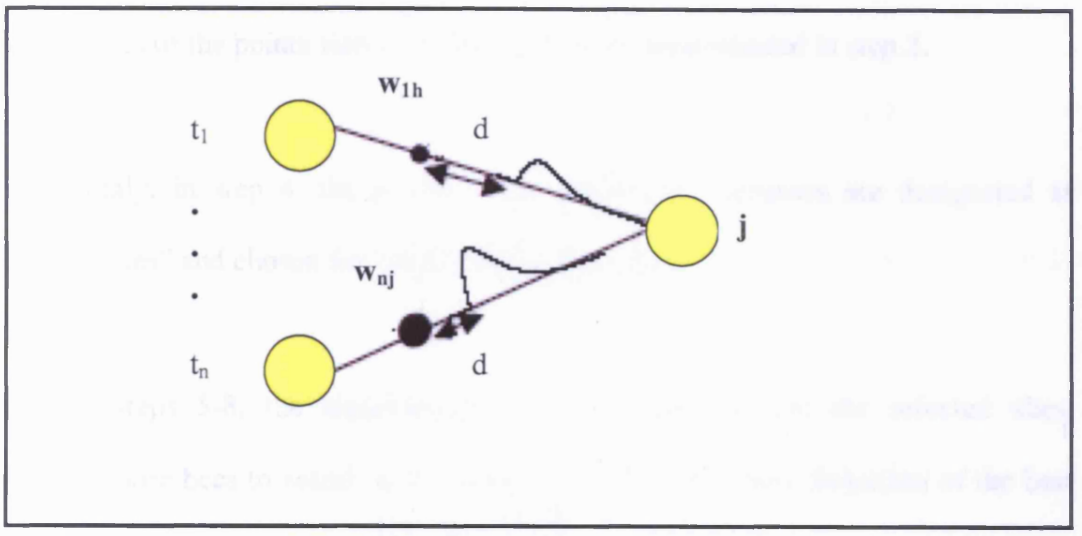

Figure 5.3: Single synapse connection between two neurons for the proposed spiking neural network with the Bees Algorithm.

5.7.3 Proposed Bees Algorithm

As mentioned above, figure 5.4 shows the pseudo-code for the proposed Bees Algorithm. The modification to the basic Bees Algorithm is the addition at step number 6 and 9. The detailed description of the pseudo-code is as follows:.

The algorithm starts with the n scout bees being placed randomly in the search space.

The fitnesses of the points visited by the scout bees are evaluated in step 2.

Subsequently, in step 4, the *m* sites with the highest fitnesses are designated as "selected sites" and chosen for neighbourhood search.

Then, in steps 5-8, the algorithm conducts searches around the selected sites, assigning more bees to search in the vicinity of the best *e* sites. Selection of the best sites can be made directly according to the fitnesses associated with them. Alternatively, the fitness values can be used to determine the probability of the sites being selected. Searches in the neighbourhood of the best *e* sites, those which represent the most promising solutions, are made more detailed. As already mentioned, this is done by recruiting more bees for the best *e* sites than for the other selected sites. Together with scouting, differential recruitment is a key operation of the Bees Algorithm. As explained previously, both scouting and differential recruitment are used in nature.

In step 6, the shrinking method is applied for neighbourhood size if the value of the fittest bee remains unchanged. This is determined after certain number of iterations.

Procedure for Proposed Bees Algorithm

1. Initialise population with random solutions.
2. Evaluate fitness of the population.
3. While (stopping criterion not met)

   // Forming new population.
4. Select elite bees and elite sites for neighbourhood search.
5. Select other sites for neighbourhood search.
6. Use initial patch size or shrink the patch size if the value of the fitness remains unchanged.
7. Recruit bees around selected sites (more bees for best elite sites) and evaluate fitnesses.
8. Select the fittest bee from each site.
9. Assign remaining bees to search randomly and evaluate their fitnesses.

   (After some iteration, reduce the population of the bees)
10. End While.

Figure 5.4: Pseudo-code of the proposed Bees Algorithm

The objective of using this method is to focus the searching in a smaller neighbourhood size. If the point is close to the optimum solution or the peak, a bigger neighbourhood range makes the search more difficult. Therefore, the patch size needs to be shrinking in order to speed up the search. After a few iterations of time shrinking, if there is no improvement to the value of the fittest bee then it means that the bee is at the peak, so the shrinking is stopped. This phenomenon is illustrated using a simple graphical example in figure 5.5. So far, this shrinking method has only been tested on an artificial neural network problem.

In step 8, for each patch, only the bee that has found the site with the highest fitness (the "fittest" bee in the patch) will be selected to form part of the next bee population. In nature, there is no such a restriction. This restriction is introduced here to reduce the number of points to be explored.

In step 9, the remaining bees in the population are assigned randomly around the search space to scout for new potential solutions.

At the end of each iteration, the colony will have two parts to its new population: representatives from the selected patches, and scout bees assigned to conduct random searches. After some iteration, reduce the population of the bees as this will speed up the algorithm.

Lastly, steps 4-9 are repeated until the stopping criterion is met. This usually means that either the best fitness value has stabilised over a number of iterations or the specified maximum number of iterations has been reached.

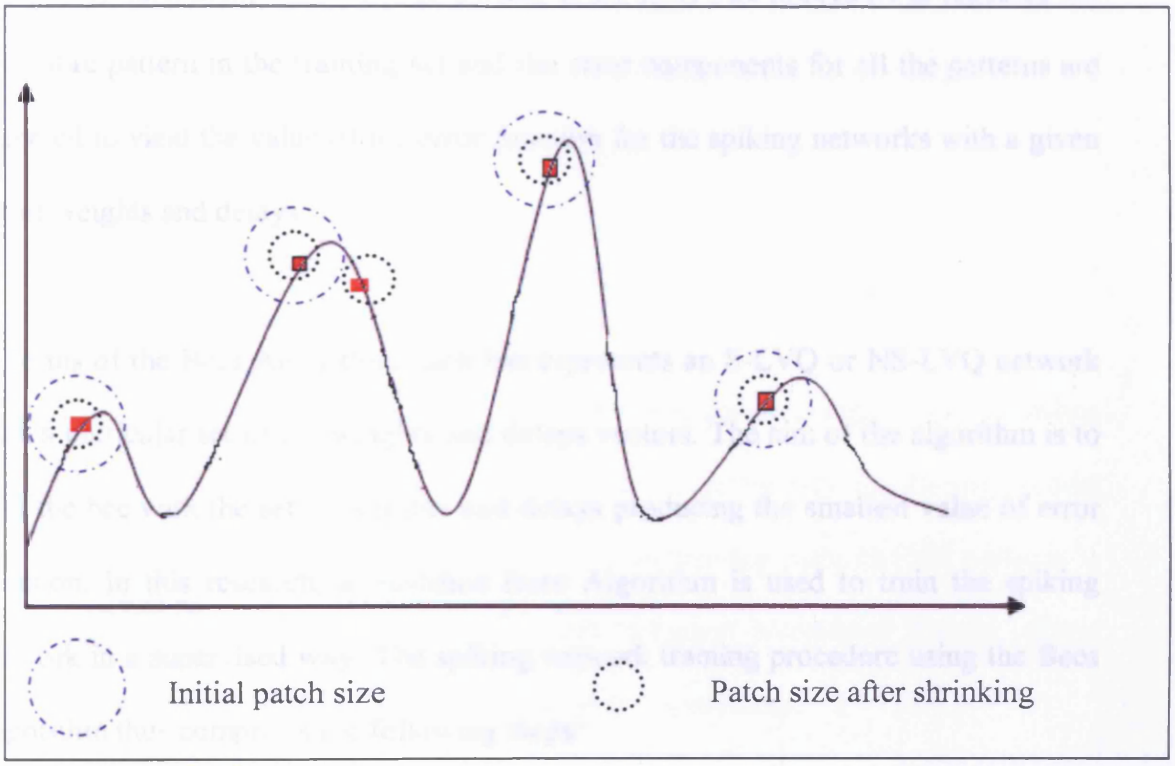

Figure 5.5: Graph illustrating the shrinking method for points that are near to the peak

5.7.4 Spiking Networks Training Procedure

The training of the artificial network can be regarded as the minimisation of an error function. The error function defines the total difference between the actual output and the desired output of the network over a set of training patterns [Pham and Oztemel, 1992]. Training proceeds by presenting to the network a pattern of known class taken randomly from the training set. If the class of the pattern is correctly identified by the network, the error component associated with that pattern is null. If the pattern is incorrectly identified, the error component is set to 1. The procedure is repeated for the entire pattern in the training set and the error components for all the patterns are summed to yield the value of the error function for the spiking networks with a given set of weights and delays.

In terms of the Bees Algorithm, each bee represents an S-LVQ or NS-LVQ network with a particular set of the weights and delays vectors. The aim of the algorithm is to find the bee with the set of weights and delays producing the smallest value of error function. In this research, a modified Bees Algorithm is used to train the spiking network in a supervised way. The spiking network training procedure using the Bees Algorithm thus comprises the following steps:

1. Generate an initial population of bee;
2. Apply the training data set to determine the value of the error function associated with each bee;
3. Based on the error value obtained in step 2, create a new population of bees comprising the best bees in the selected neighbourhoods and randomly placed scout bees;

4. Stop if the value of the error function has fallen below a predetermined threshold;

5. Else, return to step 2.

5.7.5 The Proposed Bees Algorithm Parameters

Table 5.5 shows the values of the parameters adopted for the Bees Algorithm for the spiking networks. The values were decided empirically. In addition, as mentioned above, only positive values for weights were tested. The algorithm was initialised with all weight values set randomly within the range 0 to 1. The range value of the delays was initially set empirically from 30 to 80. This is based on experience from the previous experiments in chapters 3 and 4 above.

5.8 Data Set

Spiking neural networks with the Bees Algorithm used the same data sets as described in Chapter 2. These were also generated using the previously mentioned process simulator. From these data sets, 1002 were used as training patterns (167 patterns in each category) and 498 (83 patterns in each category) were used for the testing patterns. The patterns were sequentially applied to the network.

Bees Algorithm parameters	Symbol	Value
Population	n	1000
Number of selected sites	m	20
Number of elite site	e	5
Initial patch size	ngh	0.5
Final patch size	ngh	0.007
Number of bees around elite points	nep	70
Number of bees around other selected points	nsp	50

Table 5.5: The parameters of the proposed Bees Algorithm

5.9 Empirical Evaluation of Spiking Networks with Proposed Bees Algorithm

This section presents an empirical evaluation of the control chart pattern recognition performance for spiking neural networks with a single connection using the Bees Algorithm. Two criteria were used to evaluate the performance of the tested optimisation algorithm, namely, the simple structure and classification accuracy. The accuracy level was calculated using the same equation as explained in chapters 3 and 4.

The results obtained with the proposed architecture and the supervised learning procedure for control chart pattern recognition using the Bees Algorithm are presented in Table 5.7. The results obtained for the S-LVQ and NS-LVQ networks with single synapse connection without optimisation are also presented. The results in Table 5.6 are presented graphically in Figure 5.6.

Pattern recogniser	Learning accuracy	Test accuracy
LVQ (standard)	95.18%	92.31%
S-LVQ (single synapse)	94.24%	86.65%
NS-LVQ (single synapse)	93.49%	85.10%
LVQ (Bees)	96.56%	95.47%
S-LVQ (single synapse with Bees)	96.44%	95.28%
NS-LVQ (single synapse with Bees)	94.24%	93.70%

Table 5.6: Results of different pattern recognisers

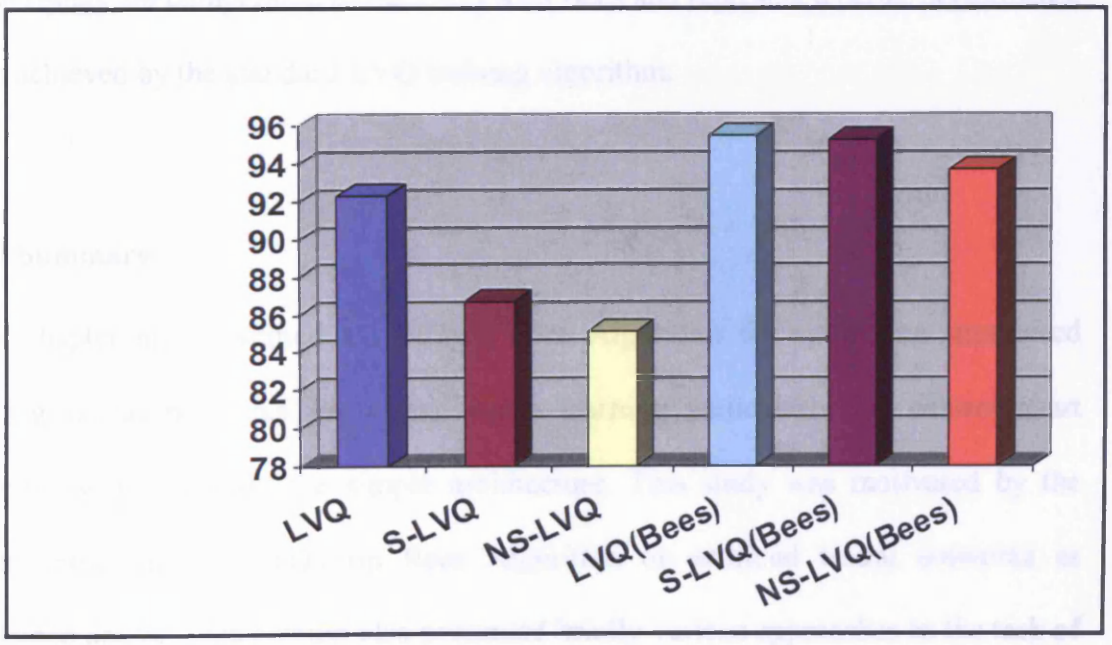

Figure 5.6: Classification accuracy of different pattern recognisers

5.9.1 Comparison with Spiking Network without Bees Algorithm

Spiking neural networks for single synapse connection for S-LVQ and NS-LVQ architecture was compared with S-LVQ and NS-LVQ with the Bees Algorithm. Experimental results have demonstrated that the proposed Bees Algorithm gives better performance compared to single synapse architecture without the algorithm. After 2000 iterations, S-LVQ can reach 95.28% of classification accuracy. NS-LQ achieved the classification accuracy 93.70% after 1000 iterations. The optimisation algorithm successfully improved the performance of the spiking network with single synapse connection. S-LVQ and NS-LVQ alone with single synapse only achieved 86.65% and 85.10% classification accuracy respectively. Compared to spiking networks without optimisation [Table 5.6], the performance demonstrated that the Bees Algorithm still succeeded in training more accurate classifiers despite the high dimensionality of the problem it faced as each bee represented 2160 (60X36) for S-LVQ and 360 (60X6) for NS-LVQ parameters need to be determined. Moreover, these results are comparable to the LVQ with Bees and represent a better performance than achieved by the standard LVQ training algorithm.

5.10 Summary

This chapter has presented a modified Bees Algorithm for optimising supervised spiking neural networks for classification learning particularly for control chart patterns, which include the simple architecture. This study was motivated by the recent experimental results on Bees Algorithm on artificial neural networks as explained above. This section also presented briefly various approaches to the task of optimising and discussed their potential in artificial neural networks. From the

literature review, only Evolution Strategy (ES) has been implemented for optimising spiking neural networks with single synapse. However, this method was only tested on XOR problems and on IRIS data. As far as the author is aware, this is the first application of the Bees Algorithm for optimising spiking neural networks for control chart pattern recognition.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

This chapter summarises the main contributions of this work and the conclusions reached. It also provides suggestions for future work.

6.1 Contributions

This research addressed the problem of pattern recognition in control chart data sets. The aim is to develop a good learning algorithm based on the spiking neuron model so that they can be successfully applied to control chart data sets. Its contributions include:

- *A thorough analysis of the issue of pattern recognition.* A critical overview has been conducted of recently available or applied learning techniques ranging from statistical to artificial intelligence methods specifically applied to control chart data sets. This also includes a discussion of their potential for the application of control chart pattern recognition. This led to the design of networks using the architecture of conventional ANNs.

- *A thorough analysis of spiking neural networks.* A critical overview has been performed of more plausible models of real biological neurons and also the existing learning algorithms, especially those suitable for pattern recognition. Their potential and successful application in pattern recognition were discussed. Finally, this led to the design of networks with spiking neurons instead of common neurons, that consider time as an important feature for information representation and processing.

- *A simpler architecture for the proposed spiking networks.* Four new architectures were developed, S-LVQ, NS-LVQ, SB-LVQ and NSB-LVQ. S-LVQ is a simple structure network similar to that of an LVQ network (S-LVQ). An enhanced network of S-LVQ (NS-LVQ) which is simpler than S-LVQ was developed. An optimisation technique was used to simplify both of the proposed spiking networks to a single synapse network instead of multi-synapses, as in SB-LVQ and NSB-LVQ. The implementation of these simple structures to the networks significantly reduced the complexity of the network and the learning time.

- *An efficient learning algorithm for better exploitation of this plausible model of real biological neurons.* The proposed algorithms employed appropriate strategies adopted from of nature and weight updating techniques that significantly increase the classification accuracy.

- *Nature-inspired techniques for improved spiking network learning.* The presented techniques for learning, which are boosting and motivation, were built inspired by the behaviour of human neuroscience for training purposes [Cooper, 2002; Tolman, 1948; Barch, 2005; Berridge, 2004]. These techniques overcame drawbacks in the spiking neural network applied to control chart pattern recognition. The Bees Algorithm is a new population-based search algorithm. The shrinking method proposed for the Bees Algorithm procedures is reliable for artificial neural network problems. This optimisation method resulted in a simpler architecture as well as better accuracy for control chart pattern recognition with spiking networks.

- *Supervised classification learning algorithms appropriate for control chart data.* Three new learning algorithms were developed, S-LVQ, NS-LVQ, and SB-LVQ and NSB-LVQ. The adopted spiking neurons in these networks are based on the Spike Response Model [Gerstner and Kistler, 2002] with some modification to the spike response function in order for the networks to be applied to control chart pattern recognition. The learning algorithm for S-LVQ is based on the integration of boosting and motivation techniques mentioned above. Such integration led to an efficient and effective learning rule for control chart pattern recognition. NS-LVQ is an enhanced network of S-LVQ. NS-LVQ is based on the integration of the boosting technique mentioned above and the Winner-Takes-All [Jin and Seung, 2002] updating weights. The main advantageous features of NS-LVQ over S-LVQ are simpler architecture and faster computation. Enriched with these new features, NS-LVQ should

produce a shorter learning time as well as better classification accuracy for control chart data sets. SB-LVQ and NSB-LVQ are improved versions of the two networks mentioned above respectively. Both networks used the Bees Algorithm to search for the optimum set of both synaptic weights and delays that allow the spiking network to learn temporal patterns. The main improvement of SB-LVQ and NSB-LVQ over NS-LVQ and S-LVQ is the simplified network structure as mentioned above. Such an optimisation method allows the network to implement a single synapse with better classification accuracy for control chart data sets.

- *An interesting effect for different numbers of hidden neurons.* The new learning algorithm proposed for S-LVQ took full advantage of the numbers of neurons in the hidden layer to improve its performance. It produced a precise learning rule as well as a learning time comparable to that achieved with standard LVQ and its variants and also with Back-propagation. This effect also resulted in better classification accuracy.

- *An effective method for pre-processing weights.* This method follows a simple mathematical formula. The weights pre-processing method which is applied before the training process overcame the drawback of using only a small range of weights in spiking networks. It is found that weight initialisation is a critical factor for good performance of the learning rule [Moore, 2002; Schrauwen and Van Campenhout, 2004; Tino and Mills, 2005; Xin and Embrechts, 2001] in certain algorithms such as SpikeProp.

6.2 Conclusions

To gain the edge in today's competitive environment, companies must employ effective tools to ensure that their products are of the highest quality. Moreover, continuous improvement of their production process is important in order continually to raise quality standards. Statistical Process Control (SPC) is a quality improvement tool widely adopted in industry and the most popular tool is control charts. It uses simple rules to determine if a process is out of control and needs corrective action. It is also possible to detect incipient problems and to prevent the process from going out of control by identifying the type of patterns displayed by the control charts [Pham and Oztemel, 1996]. Control chart pattern recognition has been successfully trained using artificial neural networks [Zorriassatine and Tannock, 1998]. For many years a common belief was that essential information in neurons is encoded in their firing rates. However, recent neurophysiological results suggest that efficient processing of information in neural systems can be founded also on the precise timing of action potentials (spikes) [Bohte, 2004; VanRullen et al., 2005; Thorpe et al., 2001]. Although the creation and development of ANNs were inspired by biological neural systems, ANNs are considered to be limited compared to their biological counterparts due to their simplistic structure and behaviour [Zador, 2000; Maass, 1999].

These considerations have led to increased interest in temporal-coding spiking neurons which are more biologically realistic artificial neurons and in Spiking Neural Networks (SNNs) which are made up of such neurons. SNNs have shown great promise as pattern recognisers [Hopfield, 1995]. Most of the existing learning algorithms are unsupervised, based on an adaptation of the famous Hebbian rule. Hebbian learning is biologically based and is a simple learning method. Therefore it is

a natural candidate for application to spiking neural networks [Kunkle and Merrigan, 2005]. However, the unsupervised approach is not suitable for learning tasks that require an explicit goal definition. Most of the supervised learning algorithms in previous research adopted the Multilayer Peceptron network with the classical back-propagation learning algorithm. However, the sizes of the network increases drastically with the number of neurons as a large set of weights have to be adjusted for multi-synapse connection.

Therefore, a new approach for supervised training is needed. Moreover, most existing algorithms were designed for low dimensions of data set. As far as the knowledge of the author goes, this thesis presents the first application of the spiking network to control chart pattern recognition [Pham and Shahnorbanun, 2006]. Producing a good supervised learning algorithm for such high dimension data sets is a formidable challenge. Therefore, simple networks architecture together with efficient and effective learning algorithm is needed for better exploitation of these new applications.

This research presented a new supervised learning algorithm that can efficiently extract accurate and comprehensible models from high-dimensional data sets such as control charts. These algorithms were tested in several experiments and the results proved that they produced better performance as regards classification accuracy.

A simulator have been developed to create and to perform a detailed analysis of spiking neuron networks on CCPR. The code for the simulator can be found in Appendix A and B.

Chapter 3 presented a supervised learning algorithm that is suitable specifically for application to control chart pattern recognition. The proposed algorithm employs a new learning rule, which embodies several techniques of natural behaviour and conventional architecture. Such integration produced a simple network which reduced the network complexity as well as overcaming the drawbacks of most LVQ learning algorithms. This algorithm also demonstrated the interesting effect of using different numbers of hidden neurons in the network. This effect resulted in precise classification accuracy.

Chapter 4 concentrated not only on enhancement of the proposed network mentioned above, but also on the procedure before the training process. The main objective of chapter 4 is to produce a simpler architecture provided with an efficient learning algorithm. The procedure before training, so called pre-process weights, is a reliable way to balance the weights so that they will not dominate the neuron potential. Adaptive learning rate experiments were also carried out, aiming to take into account the effect of the learning parameter on the learning time of the network. Experiments indicated that the proposed network with the integration of several procedures substantially improved performance in terms of learning time.

The application of spiking neural networks to control chart pattern recognition is a new research area. Therefore, it is formidable challenge to develop powerful learning mechanisms with a non-complex architecture. Setting the values of several parameters, such as the threshold, the range of the weights and the delays, are very important as they play an important role in the network. Optimisation methods are well suited for solving this task. Chapter 5 introduced a new optimisation method, the

Bees Algorithm (BA), to search for these parameters. This thesis presents the first application of the Bees Algorithm to the optimisation of parameters of spiking neural networks for control charts. The Bees Algorithm is employed to search for the optimum set of both synaptic weights and delays. The threshold is based on the previous experiments in chapters 3 and 4. The BA allows the networks to consider only a single synapse instead of multi-synapse as in the networks proposed in chapters 3 and 4. Moreover, it produces a simpler architecture for both networks. In this work, software called CONDOR is used to speed up the optimisation process. CONDOR is a software system that creates a High-Throughput Computing (HTC) environment. Condor effectively utilises the computing power of workstation that communicate over a network. Condor's power comes from the ability to effectively harness resources under distributed ownership. The new optimisation technique demonstrated competitive results in terms of classification accuracy compared to a spiking neural network with a single synapse for the networks proposed in chapters 3 and 4.

6.3 Future Work

This section suggests some of the ways in which the method and algorithms developed in this thesis could be enhanced.

- In the proposed network, only the first spike produced by a neuron is relevant and the rest of the time course of the neuron is ignored. Whenever a neuron fires a single spike, it is not allowed to fire again and this is so called 'time-to-first-spike' coding scheme [Kasinski and Ponulak, 2006]. The reason for considering this coding scheme is to ease the implementation of the network.

However, further work should consider investigating the second, the third and the rest of the spike as well as the first spike.

- The proposed networks were only tested positive weight values. The weights were initialised within the range of 0 and 1. Further work should allow negative weights which still lead to successful convergence. Examples of research in this area can be found in [Moore, 2002; Xin and Embrechts, 2001].

- In [Schrauwen and Van Campenhout, 2004], the authors adapted the gradient descent method derived in *SpikeProp* to adjust not only synaptic weights, but also synaptic delays, time constants and neurons' thresholds. They claimed that this resulted in faster algorithm convergence and in smaller network topologies required for the given learning task. A further research task is to optimise these parameters using the Bees Algorithm, as in BA each bee represent an individual bee.

- It is also possible to apply other competitive optimisation algorithms such as the Ant Colony (ACO) algorithm [Pham et al, 2006; Dorigo et al., 2004], Genetic Algorithms [Pham et al, 2006; Goldberg 1989], and the Particle Swarm Optimisation (PSO) algorithm [Xu and Eberhart, 2002a and 2002b; Carlisle and Dozier 2002; Eberhart et al., 2001] to spiking neural networks to make a realistic comparison in terms of speed of optimisation and accuracy.

- In the Bees Algorithm, the values of the tuneable parameters used are set by conducting a number of trials. Further work should address a method to solve

this drawback and to achieve a reduction of parameters. Since this algorithm is new, it is still an open problem to determine efficient learning mechanisms.

- The experiments carried out for the proposed shrinking method in the Bees Algorithm demonstrated that this method is limited for artificial neural network problems only. Further work should address a method suitable for general purposes.

Appendix A

C++ simulator for Spiking Learning Vector Quantisation (S-LVQ) and Enhanced- Spiking Learning Vector Quantisation (NS-LVQ)

MAIN

```
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <iostream>
#include <stdlib.h>

#include "spkmodified.h"
//#include "cancer.h"
//include "irish.h"
//#include "xor.h"

int main(int argc, char *argv[])
{
 spkmodified_dataset snnRBF;
 snnRBF.start();
 return EXIT_SUCCESS;
}
```

MATRIX

```
#include "Vector.h"
using namespace std;

class matrix{
private:
 int rows; int cols; double *melt; int *meltct;

public:
 matrix(int,int); int ctype; int index(int i,int j){return (i-1)*cols+j-1;};
 void initmatrix(double type); void fill_random(int seed);
 void print_matrix(); int nrows(){return rows;}; int ncols(){return cols;};
 double getelt(int,int); void setelt(int, int , double);
 int get_row_Vector(int rowno, int scol, int ecol, Vector v);
 int set_row_Vector(int rowno, int scol, int ecol, Vector v);
 int get_col_Vector(int colno, int srow, int erow, Vector v);
 int set_col_Vector(int colno, int srow, int erow, Vector v);
```

```
 int addmatrix(matrix);  int vmproduct(Vector, Vector);
 int mmproduct(matrix, matrix);
 int copymatrix(int srow, int scol,int erow, int ecol, matrix);
 void transpose();  void v_normalize();  void h_normalize();
 int read_matrix_from_file(const char*);  int write_matrix_to_file(const char*, int);
 void multiply(double);  void swap_rows(int,int);  void swap_cols(int,int);
 void shuffle(int);
};

matrix::matrix(int m, int n)
{
 rows=m;  cols=n;  melt=new double[rows*cols*sizeof(double)];
}

double matrix::getelt(int i, int j)
{
 return(melt[(i-1)*cols+j-1]);
}

void matrix::setelt(int i, int j, double e)
{
 melt[(i-1)*cols+j-1]=e;
}

void matrix::initmatrix(double type)
{
 for(int i=0;i<rows;i++)
   for(int j=0;j<cols;j++)
     melt[i*cols+j]=type;
}

void matrix::multiply(double val)
{
 double tmp;

 for(int i=0;i<rows;i++)
   for(int j=0;j<cols;j++){
     tmp=melt[i*cols+j]*val;
     melt[i*cols+j]=tmp;
 }
}

void matrix::fill_random(int seed)
{
 int tmp1;
 double tmp2;
 srand(seed);
 for(int i=0;i<rows;i++)
   for(int j=0;j<cols;j++){
    do{
```

```
    tmp1=rand()%1000;
    tmp2=tmp1/1000.0;
   }while((tmp2<0.3)||(tmp2>0.8));
    melt[i*cols+j]=tmp2;
  }
}

void matrix::print_matrix()
{
 for(int i=0;i<rows;i++)
   {
    for(int j=0;j<cols;j++){
     cout.width(wdth);
     cout<< melt[i*cols+j]<<" ";
    }
    cout<<endl;
   }
}

int matrix::addmatrix(matrix am)
{
 if ((rows!=am.nrows())||(cols!=am.ncols()))
   cout<<"Dimensions do not match(addmatrix)";

  else
{
 int i,j;
 for(i=1;i<=rows;i++)
   for(j=1;j<=cols;j++)
     setelt(i,j,getelt(i,j)+am.getelt(i,j));
 return 1;
}
return 0;
}

int matrix::mmproduct(matrix pm, matrix opm)
{
 if (cols!=pm.nrows())
   cout<<"Dimensions do not match(mmproduct)";

  else
{
 double sum;
 int i,j,k;
 for(i=1;i<=rows;i++)
   for(j=1;j<=pm.cols;j++)
     {
         sum=0;
         for(k=1; k<=pm.rows;k++)
           sum=sum+getelt(i,k)*pm.getelt(k,j);
```

```
            opm.setelt(i,j,sum);
        }
  return 1;
}
return 0;
}

int matrix::vmproduct(Vector pv, Vector opv)
{
  if (rows!=pv.getcnt()){
    cout<<"Dimensions do not match(vmproduct)";
      }
   else
{
  double sum;
  int j,k;

     for(j=1;j<=cols;j++)
       {
            sum=0;
            for(k=1; k<=rows;k++)
              sum=sum+pv.getelt(k)*getelt(k,j);
            opv.setelt(j,sum);
       }
  return 1;
}
return 0;
}

int matrix::copymatrix(int srow, int scol, int erow, int ecol, matrix cpym)
{

  int i,j,nrow,ncol;

  nrow=erow-srow+1;
  ncol=ecol-scol+1;

  for(i=1;i<=nrow;i++)
    for(j=1;j<=ncol;j++)
            cpym.setelt(i,j,getelt(i+srow-1,j+scol-1));

  return 1;
}

int matrix::get_row_Vector(int rowno, int scol, int ecol, Vector rowv)
{

  int cnt;
  cnt=ecol-scol+1;
  double tmp;
```

```
 int tmpcl;

 if(cnt>rowv.getcnt())
  cout<<"Destination dimension do not match(getrowvect)";

 else
{
 for(int i=1;i<=cnt;i++){
        if(i==1){
                tmpcl=getelt(rowno, i+scol-1);
                rowv.setelt(i,tmpcl);
                ctype=tmpcl;
        }
        else
        {
        tmp=getelt(rowno,i+scol-1);
  rowv.setelt(i,tmp);
        }
 }

 return 1;
 }
 return 0;
}
int matrix::set_row_Vector(int rowno, int scol, int ecol, Vector rowv)
{

 int cnt;
 cnt=ecol-scol+1;
 if(cnt>rowv.getcnt())
  cout<<"Destination dimension do not match(getrowvect)";

 else
{
 for(int i=1;i<=cnt;i++)
        if(i==1){
                setelt(rowno,i,rowv.getelt(i+scol-1));
        }
        else
  setelt(rowno,i,rowv.getelt(i+scol-1));
 return 1;
 }
 return 0;
}

int matrix::get_col_Vector(int colno, int srow, int erow, Vector colv)
{
 int cnt;
  cnt=erow-srow+1;
 if(cnt>colv.getcnt())
```

```
   cout<<"Destination dimension do not match(getcolvect)";

  else
{
    for(int i=1;i<=cnt;i++)
     colv.setelt(i,getelt(i+srow-1,colno));
  return 1;
}
  return 0;
}

int matrix::set_col_Vector(int colno, int srow, int erow, Vector colv)
{
 int cnt;
    cnt=erow-srow+1;
   if(cnt>colv.getcnt())
    cout<<"Destination dimension do not match(setcolvect)";

  else
{
    for(int i=1;i<=cnt;i++)
     setelt(i+srow-1,colno,colv.getelt(i));
  return 1;
}
  return 0;
}

int matrix::read_matrix_from_file(const char *file_name)
{
  int i,j;  char ch;  double ele;

  ifstream infile(file_name);
  for(i=1;i<=rows;i++){
   for(j=1;j<cols;j++){
            if (j==1){
           infile>>ctype;
           infile>>ch;
    setelt(i,j,ctype);
            }
            else {
           infile>>ele;
    infile>>ch;
           setelt(i,j,ele);
            }
   }
    infile>>ele;
     setelt(i,j,ele);
      }

  infile.close();
```

```
 return 1;
}

int matrix::write_matrix_to_file(const char *file_name,int type)
{

 char ch;  int i,j;  ch=',';
 if (type==1){
  ofstream outfile(file_name);
  for(i=1;i<=rows;i++){
   for(j=1;j<=cols-1;j++)
     outfile<<getelt(i,j)<<ch;
   outfile<<getelt(i,j)<<endl;
  }
   outfile<<endl;
   outfile.close();
 }
  else{
    ofstream outfile(file_name,ios::app);
      for(i=1;i<=rows;i++)
                for(j=1;j<=cols;j++){
                if (j<cols) outfile<<getelt(i,j)<<ch;
        else outfile<<getelt(i,j)<<endl;
      }
   outfile<<endl;
   outfile.close();
  }

  return 1;

}
```

MATRIX 3D

```
//#include "Vector.h"

class matrix3D{
private:
 int rows;  int cols;  int planes;  int P_ele;    //No of elements in a plane
 int N_ele;    //Total no of elements  double *melt;
public:
 matrix3D(int,int,int);  void initmatrix(double type);  void fillrandom(int seed);
 void fill_inc(int seed,int, int);  void print_matrix();  int nrows(){return rows;};
 int ncols(){return cols;};  int nplanes(){return planes;};  double getelt(int,int,int);
 void setelt(int, int , int, double);  void setelt1(int, int , int, double);
 int read_matrix_from_file(const char*);  int write_matrix_to_file(const char*, int);
 void multiply(double);
};

matrix3D::matrix3D(int l, int m, int n)
```

```
{
 rows=m; cols=n; planes=l; P_ele=rows*cols; N_ele=l*P_ele;
 melt=new double[N_ele*sizeof(double)];
}

double matrix3D::getelt(int i, int j, int k)
{
 return(melt[(k-1)*P_ele+(i-2)*cols+j-1]); }
void matrix3D::setelt(int i, int j, int k, double e)
{
 melt[(k-1)*P_ele+(i-2)*cols+j-1]=e;
}

void matrix3D::setelt1(int i, int j, int k, double e)
{
 melt[(k-1)*P_ele+(i-2)*cols+j-1] = e+0.4;

 if (melt[(k-1)*P_ele+(i-2)*cols+j-1] > 1) {
        melt[(k-1)*P_ele+(i-2)*cols+j-1]=1;
 }
}

void matrix3D::initmatrix(double type)
{
 for(int k=0;k<N_ele;k++)
   melt[k]=type;
}

void matrix3D::multiply(double val)
{
 double tmp;

 for(int k=0;k<N_ele;k++){
   tmp=melt[k]*val;
   melt[k]=tmp;
 }
}

void matrix3D::fill_inc(int seed, int min, int max)
{
 int rnd; double val,s,p; srand(seed); p=planes; s=(max-min)/p;

  for(int i=1;i<=rows;i++)
    for(int j=1;j<=cols;j++) {
      val=min;
      for(int k=1;k<=planes;k++){
        rnd=rand()%1000;
        val=val+(rnd/1000.0)*s;
        setelt(i,j,k,val);
      }
```

```
    }
}

void matrix3D::fillrandom(int seed)
{
 int rnd, tot_ele;
 double val;

 srand(seed);
 tot_ele=rows*cols*planes;
 for(int k=0;k<tot_ele;k++){
        // do{ testing 4/10/05
    rnd=rand()%1000;
    val=rnd/1000.0;
    melt[k]=val;
 }
 }

void matrix3D::print_matrix()
{
 for(int i=0;i<rows;i++){
    for(int j=0;j<cols;j++){
      for(int k=0;k<planes;k++){
        cout.width(wdth);
        cout<< melt[k*P_ele+i*cols+j]<<" ";
      }
    cout<<endl;
   }
 cout<<endl;
 }
}

int matrix3D::read_matrix_from_file(const char *file_name)
{
 int i,j,k;  char ch;  double ele;

 ifstream infile(file_name);
 for(k=1;k<planes;k++){
   for(i=1;i<=rows;i++){
    for(j=1;j<cols;j++){
      infile>>ele;
      infile>>ch;
      setelt(k,i,j,ele);
    }
    infile>>ele;
    setelt(i,j,k,ele);
   }
 }
 return 1;
}
```

```
int matrix3D::write_matrix_to_file(const char *file_name,int type)
{

 char ch;  int i,j,h;  ch=' ';
 if (type==1){
  ofstream outfile(file_name);
  for(i=2;i<=rows+1;i++)
   for(j=1;j<=cols;j++){
     for(h=1;h<planes;h++)
       outfile<<getelt(i,j,h)<<ch;
   outfile<<getelt(i,j,h)<<endl;
   }
   outfile<<endl;
  }
   else{
    ofstream outfile(file_name,ios::app);
    for(i=2;i<=rows+1;i++)
            for(j=1;j<=cols;j++){
      for(h=1;h<planes;h++)
               outfile<<getelt(i,j,h)<<ch;
      outfile<<getelt(i,j,h)<<endl;
      }
   outfile<<endl;
   }
 return 1;

}
```

MULTISYNAPSE

```
#include "matrix.h"
#include "matrix3d.h"
#include <iostream>
#define PI 22.0/7.0
using namespace std;

inline double mod(double val){return  (val<0) ? -val : val;}

void write_string_to_file(const char *file_name ,const char *text)
{
 ofstream outfile(file_name,ios::app);
 outfile<<text<<endl;
}

void write_number_to_file(const char *file_name ,double number)
{
 ofstream outfile(file_name,ios::app);
 outfile<<number<<endl;
}
```

```
double round(double num, double preci)
{

double multiplier=pow(10,preci);
int val=(int)(num*multiplier+0.5);
return val/multiplier;

}

double normdist(double e)
{
 double sigma=0.2;  double mu=0.5;  double val;

 val=2*sigma*sqrt(2*PI);  val=1/val;
 val=val*exp(-(e-mu)*(e-mu)/(2*sigma*sigma));

 return val;
}

class spikeNN_multi{
protected:

 int N_records,N_attrib,N_target_attrib,N_train,N_test,N_class,N_sample,N_bias;
 int max_epocs, cont_learn;
  int N_input,N_output,N_subc,N_popNeuron,N_rows,N_cols;
 const char *trainset,*train_targetset,*testset,*test_targetset,*sampleset;
 const char *outputfile,*outputfile_2,*outputfile_sample,*outputfile2;
 matrix *train_matrix,*tv_train,*tv_test,*test_matrix;

 double maxw, minw;  Vector *threshold;  Vector *pspmax;

 int t,dt,twindow_input,twindow,timestep,early_fire,late_fire;//time,
 double winning_time;  int winner,winner_found;

 double min_learning_rate, max_learning_rate, learning_rate;
 float min_nehbr_width, max_nehbr_width, nehbr_width_time,nehbr_width_dist;

 double total_error, min_error,tolerance;
 int precision;
 Vector *inpv;

 double winner_bias_chg, looser_bias_chg;

 matrix3D *wght;    //Synaptic weights - N_input X N_output X N_subc
 matrix3D *dwght;   //weight change
 double tce,tci;        //syanptic time constant
 matrix3D *delay;    //Synaptc delays
 Vector *iln_type;  Vector *oln_type;

 matrix3D *sp;     //synapse potential - N_input X N_output X N_subc
```

```
  Vector *np;      //Neuron(soma) potential - N_output
  Vector *op;      //Output - N_output
  Vector *exact_op; //Exact Output - N_output
  matrix *test_output;  //to store test output N_test X N_output
  matrix *test_winner;  matrix *train_winner;

public:
  spikeNN_multi();  void createNN();  void init_spikeNN();
  void init_parameters();  void set_neuron_type();
  double correct_class();  double patterns_evaluated();  float dist(int, int);
  int check_convergence();  void set_threshold_min(int seed);
  double adj_learning_rate(double delta_step);
  void preprocess_weight(const char *opt_train);

  void update_synapse_potential();  void update_output();
  void update_output_test();  void update_output_pw();


  void find_winner(int output_label);   void prefind_winner

  void train_spikeNN();  void test_spikeNN(const char *opt_train);

  double error_train_set();

  double calculate_error();  double error(int ,int);  int set_error_vect();

  void update_weight(int classt);  void update_weight_new();
  void update_weight_modified();

  double spike_response(double t);    double spike_response_Char(double st);
  double spike_response_Shah(double st);  double spike_response_abc(double st);
  double spike_response_new1(double st);  double spike_response_123(double st);

  double spike_time_total;  double spike_time_new[6];  double dif_spike_time[6];
  double test_win_t[6];  double arr_op[6];

  double etamin;  float train_accuracy,test_accuracy;

  int ctype;  int classtype;  int patterns,total;  int sum_total,sum_patterns;
  };
spikeNN_multi::spikeNN_multi()
{
}

void spikeNN_multi::createNN()
{

  train_matrix=new  matrix(N_train,N_attrib*N_popNeuron); //Matrix  contains  the
training records;
```

```
 test_matrix=new matrix(N_test,N_attrib*N_popNeuron); //Matrix contains the test
records;

 inpv =new Vector(N_attrib);
 wght =new matrix3D(N_subc,N_input,N_output);
 dwght=new matrix3D(N_subc,N_input,N_output);
 iln_type=new Vector(N_attrib);  oln_type=new Vector(N_output);

 delay=new matrix3D(N_subc,N_input,N_output);
 sp=new matrix3D(N_subc,N_input,N_output);
 threshold=new Vector(N_output);  pspmax=new Vector(N_output);
 np=new Vector(N_output);
 op=new Vector(N_output);  exact_op=new Vector(N_output);
 test_winner=new matrix(N_test,4);  train_winner=new matrix(N_train,4);

}

void spikeNN_multi::init_parameters()
{
//to be defined in the parent_class}
double spikeNN_multi::adj_learning_rate(double delta_step)
{
 learning_rate=(learning_rate - delta_step);
 if(learning_rate < etamin)
        learning_rate = etamin;
 return learning_rate;
}


void spikeNN_multi::init_spikeNN()
{
 sp->initmatrix(0.0);
 np->initialise(0.0);
 op->initialise(twindow);
 exact_op->initialise(twindow);
 dwght->initmatrix(0.0);
}

void spikeNN_multi::set_neuron_type()
{
 int i;
 for(i=+2;i<=N_input+1;i++)
  iln_type->setelt(i,1);

 for(i=1;i<=N_output;i++)
  oln_type->setelt(i,1);
}
void spikeNN_multi::set_threshold_min(int seed)
{
 int rownum,active,i;  double max_threshold;
```

```
 srand(seed);
 max_threshold=(N_input*N_subc*0.5)/10;
 inpv->initialise(40.0);

  do{
   init_spikeNN();
   rownum=rand()%N_train;
        train_matrix->get_row_Vector(rownum,1,N_attrib,*inpv);
   for(i=1;i<=N_output;i++){
    threshold->setelt(i,max_threshold);
        }

        t=0;
   active=true;
   init_spikeNN();
   do{
    t=t+dt;
    update_synapse_potential();
    update_output();
   }while(t<twindow);

   for(i=1;i<=N_output;i++)
    if (active) active=(op->getelt(i)<twindow);

   max_threshold-=5.5;
 }while(!(active)&&(max_threshold>0));

 threshold->initialise(max_threshold-5.0);   //testing
 cout<<"Threshold:"<<threshold->getelt(1)<<endl;
 if (!(active)) cout<<"Not all neurons can fire";
       }
double spikeNN_multi::spike_response_123(double st)
{
 double val=0.0;

 if (st>0) val=(1/(1-(tce/tci)))*(exp((-(1+st)/tci))-exp((-(1+st)/tce)));
 return val;
}

void spikeNN_multi::update_synapse_potential()
{
 double inpT,dlay,spikeT,spoten,type;   int i,j,k,inpTclass;

  for(i=1;i<=N_attrib;i++){
        if(i==1){
              inpTclass=inpv->getelt(i);}
        else {
   inpT=inpv->getelt(i);
   type=iln_type->getelt(i);
   if ((inpT<0)||(inpT>twindow_input)) continue;
```

```
    for(j=1;j<=N_output;j++){
      if (np->getelt(j)<0.0) continue;
       for(k=1;k<=N_subc;k++){
        dlay=delay->getelt(i,j,k);
        spikeT=t-inpT-dlay;

        if (spikeT<0) continue;
                    /* if (inpTclass==1){
                              spoten= spike_response_123(0.1+spikeT)*wght-
>getelt(i,j,k)*type;
                              sp->setelt(i,j,k,spoten); }

                    else if (inpTclass==2){

                              spoten= spike_response_123(0.3+spikeT)*wght-
>getelt(i,j,k)*type;
                              sp->setelt(i,j,k,spoten); }


                    else if (inpTclass==3){

                              spoten= spike_response_123(0.5+spikeT)*wght-
>getelt(i,j,k)*type;
                              sp->setelt(i,j,k,spoten); }

                    else if (inpTclass==4){

                              spoten= spike_response_123(0.7+spikeT)*wght-
>getelt(i,j,k)*type;
                              sp->setelt(i,j,k,spoten); }

                    else if (inpTclass==5){

                              spoten= spike_response_123(0.9+spikeT)*wght-
>getelt(i,j,k)*type;
                              sp->setelt(i,j,k,spoten); }

                    else {

                         spoten=     spike_response_123(0.95+spikeT)*wght-
>getelt(i,j,k)*type;
                              sp->setelt(i,j,k,spoten); }


          spoten=spike_response_123(spikeT)*wght->getelt(i,j,k)*type;

          sp->setelt(i,j,k,spoten);

       } //end for k
    }//end for j
```

```
  }//end for i
  }
//        outfile.close();
}

void spikeNN_multi::update_output_pw()
{
  double sv,pnp,spike_time;   int i,j,k,outTclass;

  for(i=1;i<=N_output;i++){
   if(np->getelt(i)<0) continue;
    sv=0.0;
    for(j=1;j<=N_attrib;j++)
                if(j==1) {
               outTclass=inpv->getelt(j);}
                else {
                for(k=1;k<=N_subc;k++)

          sv=sv+sp->getelt(j,i,k);
                        }

                if(i==outTclass) {
                pnp=np->getelt(i);
                np->setelt(i,sv=sv+0.0);
          }
          else
          {
          pnp=np->getelt(i);
    np->setelt(i,sv);
    }
    pnp=np->getelt(i);
    np->setelt(i,sv);

    if(sv>=threshold->getelt(i)) {

     spike_time=(threshold->getelt(i)-pnp)/(sv-pnp);

                op->setelt(i,t);
     exact_op->setelt(i,t-1+spike_time);

     np->setelt(i,-1000);
     for(j=+2;j<=N_attrib;j++)
      for(k=1;k<=N_subc;k++)
       sp->setelt(j,i,k,0);
    }
                }
}

void spikeNN_multi::update_output()
{
```

```
  double sv,pnp,spike_time;   int i,j,k,outTclass;

  for(i=1;i<=N_output;i++){
   if(np->getelt(i)<0) continue;
    sv=0.0;
    for(j=1;j<=N_attrib;j++)
                  if(j==1) {
                outTclass=inpv->getelt(j);}
                  else {
                  for(k=1;k<=N_subc;k++)

          sv=sv+sp->getelt(j,i,k);
                                  }

          if(i==outTclass) {
                  pnp=np->getelt(i);
                  np->setelt(i,sv=sv+0.9);
          }
          else
          {
          pnp=np->getelt(i);
    np->setelt(i,sv);
    }
    pnp=np->getelt(i);
    np->setelt(i,sv);

    if(sv>=threshold->getelt(i)) {
     spike_time=(threshold->getelt(i)-pnp)/(sv-pnp);

                  op->setelt(i,t);
     exact_op->setelt(i,t-1+spike_time);

     np->setelt(i,-1000);
     for(j=+2;j<=N_attrib;j++)
      for(k=1;k<=N_subc;k++)
        sp->setelt(j,i,k,0);
    }
                  }
 }
void spikeNN_multi::update_output_test()
{
  double sv,pnp,spike_time;   int i,j,k,outTclass;

  for(i=1;i<=N_output;i++){
   if(np->getelt(i)<0) continue;
    sv=0.0;
    for(j=1;j<=N_attrib;j++)
                  if(j==1) {
                outTclass=inpv->getelt(j);}
                  else {
```

```
                for(k=1;k<=N_subc;k++)

        sv=sv+sp->getelt(j,i,k);
                        }

        if(i==outTclass) {
                pnp=np->getelt(i);
                np->setelt(i,sv=sv);}
        else
        {
        pnp=np->getelt(i);
    np->setelt(i,sv);
    }
    pnp=np->getelt(i);
    np->setelt(i,sv);

    if(sv>=threshold->getelt(i)) {

     spike_time=(threshold->getelt(i)-pnp)/(sv-pnp);

                op->setelt(i,t);
     exact_op->setelt(i,t-1+spike_time);

     np->setelt(i,-1000);
     for(j=+2;j<=N_attrib;j++)
      for(k=1;k<=N_subc;k++)
        sp->setelt(j,i,k,0);
    }
                }
}
void spikeNN_multi::prefind_winner()
{
 double win_t=twindow;

 for(int i=1;i<=N_output;i++)
  if (win_t>=exact_op->getelt(i)) {
    winner=i;
    win_t=op->getelt(i);

 }
 winner_found=(win_t<twindow);
 winning_time=win_t;
}

void spikeNN_multi::find_winner(int output_label)
{
 double win_t=twindow;
 int winning_node, winner_category;
```

```
 for(int i=1;i<=N_output;i++)
  if (win_t>=exact_op->getelt(i)) {
     winner=i;
     win_t=op->getelt(i);

  }
  winner_found=(win_t<twindow);
  winning_time=win_t;

  winning_node = winner;
          if(winning_node==1)
                  {winner_category = 1;}
          else if(winning_node==2)
                  {winner_category = 2;}
          else if(winning_node==3)
                  {winner_category = 3;}
          else if(winning_node==4)
                  {winner_category = 4;}
          else if(winning_node==5)
                  {winner_category = 5;}
          else
                  {winner_category = 6;}

   //*****Update Number Of Correct Classifications*****

  if(winner_category == output_label) {
            correct_class();
  }

  //*****Update Number of Patterns Processed******

            patterns_evaluated();
}

double spikeNN_multi::correct_class()
{
          total = total + 1;
          sum_total = total;
          return (sum_total-1);
}

double spikeNN_multi::patterns_evaluated()
{
          patterns = patterns + 1;
          sum_patterns = patterns;
          return (sum_patterns-1);
}
void  spikeNN_multi::train_spikeNN()
{
  int epoc;  int max_epocs=15;  double eta=0.0003;  etamin=0.0025;
```

```
 double learning_rate=0.0075;  double delta_eta;
 wght->write_matrix_to_file(outputfile,1);  epoc=0;
 min_error=1000;  total_error=0.0;  cont_learn=1;  total=0;  patterns=0;
 train_accuracy=0.0;

 do{
  epoc++;
       delta_eta=(learning_rate-eta);


       for(int r=1;r<=N_train;r++){
        spike_time_total=0;
    init_spikeNN();

        train_matrix->get_row_Vector(r,1,N_attrib,*inpv);

    classtype=train_matrix->getelt(r,1);  //add on 5/8/05 to classify

    inpv->write_Vector_to_file(outputfile,2);
    t=0;
    winner_found=0;

    do{
     t=t+dt;
          update_synapse_potential();
      update_output();
       }while(t<twindow);

    find_winner(classtype);
    if (winner_found) update_weight(classtype); //add on 5/8/05 to classify
    cont_learn=check_convergence();
       }

   cout<<"epoc :"<<epoc<<endl;
       adj_learning_rate(delta_eta);

 }while((epoc<max_epocs) & cont_learn);
 train_accuracy=((float)correct_class()/(float)patterns_evaluated())*100.00;
 cout<<"train accuracy :"<<train_accuracy<<"\n";
}


int spikeNN_multi::check_convergence()
{
int converged;double acc_weight_change;

 for(int j=1;j<=N_output;j++)
  for(int i=2;i<=N_input+1;i++)
   for(int k=1;k<=N_subc;k++)
    acc_weight_change+=mod(dwght->getelt(i,j,k));
```

```
 converged=(acc_weight_change<tolerance);
 converged=1;
 return converged;
}

void spikeNN_multi::update_weight(int classt)
{
 double input,weight,newweight,dw,deltaT,dlay;
 double wf,output;  int type,winning_node,winner_category;
 double beta=35;  double spi=sqrt(2*22.0/7.0);  cont_learn=1;

        winning_node = winner;
        if(winning_node==1)
                {winner_category = 1;}
        else if(winning_node==2)
                {winner_category = 2;}
        else if(winning_node==3)
                {winner_category = 3;}
        else if(winning_node==4)
                {winner_category = 4;}
        else if(winning_node==5)
                {winner_category = 5;}
        else
                {winner_category = 6;}
        output=op->getelt(winner);
    for(int i=2;i<=N_attrib;i++){
    input=inpv->getelt(i);
    type=(int)iln_type->getelt(i);

         if (input<0) continue;

      for(int k=1;k<=N_subc;k++){
                dlay=delay->getelt(i,winner,k);   //testing on 12/9/05
       deltaT=output-dlay-input;
       weight=wght->getelt(i,winner,k);  //testing on 12/9/05
       wf=0.0;  dw=0.0;    newweight=0.0;

                        if(winner_category==classt) {

                if (deltaT>=0)
          dw=learning_rate*(1/(beta*spi))*exp(-
(sqrt(deltaT*deltaT)/(2.0*beta*beta)));//add on 5/8/05 to classify


                else
          dw=-learning_rate*(1/(beta*spi))*exp(-
(sqrt(deltaT*deltaT)/(2.0*beta*beta)));

                newweight=weight+dw;
```

```
        if (newweight<0.0) newweight=0.0;
        if (newweight>1.0) newweight=1.0;

                }
                // *** Weight Adjustments For Incorrect Output Label *** //

                else {

                 if (deltaT>=0)

                        dw=0.8*learning_rate*(1/(beta*spi))*exp(-
(sqrt(deltaT*deltaT)/(2.0*beta*beta)));

                 else

        dw=-0.8*(-learning_rate)*(1/(beta*spi))*exp(-
(sqrt(deltaT*deltaT)/(2.0*beta*beta)));

                newweight=weight-dw;
         if (newweight<0.0) newweight=0.0;
         if (newweight>1.0) newweight=1.0;

                        dwght->setelt(i,winner,k,dw);
                        wght->setelt(i,winner,k,newweight);
                                }
     }
    }
}
/*
void spikeNN_multi::update_weight(int classt)
{
 double input,weight,newweight,dw,deltaT,dlay;
 double wf,output;  int type,winning_node,winner_category;

 double beta=35;  double spi=sqrt(2*22.0/7.0);  cont_learn=1;
        winning_node = winner;
        if(winning_node==1)
                {winner_category = 1;}
        else if(winning_node==2)
                {winner_category = 2;}
        else if(winning_node==3)
                {winner_category = 3;}
        else if(winning_node==4)
                {winner_category = 4;}
        else if(winning_node==5)
                {winner_category = 5;}
        else
                {winner_category = 6;}

        output=op->getelt(winner);  //testing on 12/9/05
```

```
    for(int i=2;i<=N_attrib;i++){
    input=inpv->getelt(i);
    type=(int)iln_type->getelt(i);
          if (input<0) continue;
     for(int k=1;k<=N_subc;k++){
          dlay=delay->getelt(i,winner,k);   //testing on 12/9/05
       deltaT=output-dlay-input;
           weight=wght->getelt(i,winner,k);  //testing on 12/9/05
      wf=0.0;   dw=0.0;     newweight=0.0;

        if (deltaT>=0)

                dw=learning_rate*(1/(beta*spi))*exp(-
(sqrt(deltaT*deltaT)/(2.0*beta*beta)));

        else

     dw=-learning_rate*(1/(beta*spi))*exp(-(sqrt(deltaT*deltaT)/(2.0*beta*beta)));


                // *** Weight Adjustments For Correct Output Label *** //

                if(winner_category==classt) {

                                newweight=weight+dw;
        if (newweight<0.0) newweight=0.0;
        if (newweight>1.0) newweight=1.0;

                  dwght->setelt(i,winner,k,dw);
                  wght->setelt(i,winner,k,newweight);

                }
                        // *** Weight Adjustments For Incorrect Output Label *** //

                else {

                   newweight=weight-dw;
        if (newweight<0.0) newweight=0.0;
        if (newweight>1.0) newweight=1.0;

                  dwght->setelt(i,winner,k,dw);
                  wght->setelt(i,winner,k,newweight);
                                        }
     }
    }
}
*/
void spikeNN_multi::test_spikeNN(const char *opt_test)
{
```

```
 test_accuracy=0.0;

 test_matrix->write_matrix_to_file(opt_test,1);
 wght->write_matrix_to_file(opt_test,2);
 for(int r=1;r<=N_test;r++){
  init_spikeNN();

       test_matrix->get_row_Vector(r,1,N_attrib,*inpv);
  t=0;
  do{
   t=t+dt;
   update_synapse_potential();
   update_output_test();
  }while(t<twindow);

  op->write_Vector_to_file(opt_test,2);
  find_winner(classtype);

 }
 test_accuracy=((float)correct_class()/(float)patterns_evaluated())*100.00;
 cout<<"test accuracy :"<<test_accuracy<<"\n"; }
```

SPKMODIFIED

```
#include "multisynapse.h"

class spkmodified_dataset: public spikeNN_multi
{
private:

public:
spkmodified_dataset();voidselect_data();voidpreprocess_data();
void shuffle_trainset();void init_parameters();void set_neuron_type();
void start();
};

spkmodified_dataset::spkmodified_dataset()
{
 max_epocs=5; N_attrib=61; N_target_attrib=1; N_train=1002; N_test=498;
 N_input=N_attrib-1; N_rows=1; N_cols=6; N_output=N_rows*N_cols;
 N_subc=16; N_popNeuron=1; N_class=6;  learning_rate=0.001;
 dt=1; timestep=1; twindow=300; early_fire=15; late_fire=25;
 twindow_input=100; tolerance=1.0; maxw=1.0; minw=0.0;

 outputfile="cchart_test_output.txt"; outputfile_2="cchart_test.txt";
 t=0;
}

void spkmodified_dataset::preprocess_data()
{
```

```
 int i,j,category;  double target;

for(i=1;i<=train_matrix->nrows();i++)
        for(j=1;j<=train_matrix->ncols();j++){

                if(j==1){
                 category=train_matrix->getelt(i,j);
           train_matrix->setelt(i,j,category);
                 }
         else {
                 target=train_matrix->getelt(i,j);
      //target=100-(target*100);
                 target=1*target;
      train_matrix->setelt(i,j,target);
                 }
 }
for(i=1;i<=test_matrix->nrows();i++)
 for(j=1;j<=test_matrix->ncols();j++){

         if(j==1) {
  category=test_matrix->getelt(i,j);
  test_matrix->setelt(i,j,category);
         }
         else {
  target=test_matrix->getelt(i,j);
 //  target=100-(target*100);
       target=1*target;
  test_matrix->setelt(i,j,target);
 }
}
}

void spkmodified_dataset::init_parameters()
{
int seedgen;int seed_array[35000]; int seed_cnt=0;int i,j,k;

 seedgen=time(NULL)%1000;
 for(i=0;i<seedgen;i++)
  rand();
 for(i=0;i<35000;i++)
  seed_array[i]=rand();

 for(i=2;i<=N_input+1;i++)
         for(j=1;j<=N_output;j++)
    for(k=1;k<=N_subc;k++){
     delay->setelt(i,j,k,k*10);
    }
 wght->fillrandom(seed_array[seed_cnt++]);
 for(i=2;i<=N_input+1;i++)
         for(j=1;j<=N_output;j++)
```

```
    for(k=1;k<=N_subc;k++){
                wght->setelt(i,j,k,double(wght->getelt(i,j,k)));
                 }
  wght->write_matrix_to_file(outputfile,1);

}
void spkmodified_dataset::set_neuron_type()
{

for(int i=2;i<=N_input+1;i++)
        iln_type->setelt(i,1);
}

void spkmodified_dataset::start()
{
  createNN(); set_neuron_type(); select_data(); preprocess_data();
  init_parameters();
  tce=150; tci= 20;
  set_threshold_min(time(NULL));
  for(int i=1;i<=5;i++)
//   preprocess_weight("preprocess_wght.txt");
  set_threshold_min(time(NULL));
  train_spikeNN();
  test_spikeNN("test_set_output.txt");
}

void spkmodified_dataset::select_data()
{
   int i;
        matrix sourcetr_matrix(N_train,N_attrib);
        matrix sourcete_matrix(N_test,N_attrib);
  sourcetr_matrix.read_matrix_from_file("R:\Traindata.txt");
  sourcete_matrix.read_matrix_from_file("R:\Testdata.txt");
  Vector svect(N_attrib);
for(i=1;i<=N_train;i++) {
sourcetr_matrix.get_row_Vector(i,1,N_attrib,svect);
 train_matrix->set_row_Vector(i,1,N_attrib,svect);
}
for(i=1;i<=N_test;i++) {
 sourcete_matrix.get_row_Vector(i,1,N_attrib,svect);
 test_matrix->set_row_Vector(i,1,N_attrib,svect);
}
 }
```

VECTOR

```
# include <iostream>
# include <math.h>
# include <stdlib.h>
# include <time.h>
```

```
# include <fstream>
using namespace std;

#define wdth 15

typedef double eltype;

class Vector{
 private:
  int nelt;
  double *velt;
public:
  Vector(int);  void initialise(double val);  int getcnt(){return nelt;};
  double getct(int i) {return velt[i-1];};  double getelt(int i){return velt[i-1];};
  void setelt(int i, double e){velt[i-1]=e;};  void setct(int i,int e) {velt[i-1]=e;}
  double min();  double max();  void print_Vector();  void normalize();
  int write_Vector_to_file(const char*,int);//File name, open type(write/append)
  int read_Vector_from_file(const char*);  void fill_inc(int seed, int min, int max);
  void fill_random(int seed);  void multiply(double);  void code_temporal();
  void swap_elts(int, int);  void shuffle(int);

};

Vector::Vector(int n)
{
  nelt=n;
  velt=new eltype[nelt*sizeof(eltype)];
}

eltype Vector::min()
{
  eltype mn=10;
  for(int i=0;i<nelt;i++)
    if (velt[i]<mn) mn=velt[i];
  return mn;
}
eltype Vector::max()
{
  eltype mx=0;
  for(int i=0;i<nelt;i++)
    if (velt[i]>mx) mx=velt[i];
  return mx;
}

void Vector::initialise(double val)
{
  for(int i=0;i<nelt;i++)
    velt[i]=val;
}
```

```
void Vector::print_Vector()
{
 for (int i=0;i<nelt;i++){
  cout.width(wdth);
  cout<<velt[i]<<" ";
 }
 cout<<endl;
}

void Vector::normalize()
{

 eltype mn,mx,diff,tmp;

 mn=min()-0.5;  mx=max()+0.5;  diff=mx-mn;
 for(int j=0;j<nelt;j++) {
    tmp=1-((velt[j]-mn)/diff);
    velt[j]=tmp;
 }

}

int Vector::write_Vector_to_file(const char *file_name,int type)
{

 if (type==1){
  ofstream outfile(file_name);
  for(int j=0;j<nelt;j++)
   outfile<<velt[j]<<" ";
  outfile<<endl<<endl;
 }
 else {
  ofstream outfile(file_name,ios::app);
  for(int j=0;j<nelt;j++)
   outfile<<velt[j]<<" ";
  outfile<<endl;
 }
 return 1;
}

int Vector::read_Vector_from_file(const char *file_name)
{
 int k;  char ch;  double ele;

 ifstream infile(file_name);
 for(k=1;k<nelt;k++){
  infile>>ele;
  infile>>ch;
  setelt(k,ele);
```

```
  }
  infile>>ele;
  setelt(k,ele);
  return 1;
}

void Vector::fill_random(int seed)
{
  double rnd1,rnd2;

  srand(seed);
  for(int k=1;k<=nelt;k++){
     do{
     rnd1=rand()%1000;
     rnd2=rnd1/1000.0;
    }while((rnd2<0.3)||(rnd2>0.8));
   setelt(k,rnd2);
  }
}

void Vector::multiply(double val)
{
  double e;

  for(int k=1;k<=nelt;k++){
    e=getelt(k)*val;
    setelt(k,e);
  }
}

void Vector::fill_inc(int seed, int min, int max)
{
  int rnd;
  double val,s;
  srand(seed);
  s=(max-min)/double(nelt);

  val=min;
  for(int k=1;k<=nelt;k++){
    rnd=rand()%1000;
    val=val+s*rnd/1000.0;
    setelt(k,val);
  }
}

void Vector::code_temporal()
{
double val;

 for(int k=1;k<=nelt;k++){
```

```
    val=getelt(k);
    val=10-val;
    setelt(k,val);
  }
}

void Vector::swap_elts(int i, int j)
{
double tmp;

 tmp=getelt(j);
 setelt(j,getelt(i));
 setelt(i,tmp);
}

void Vector::shuffle(int seed)
{

srand(seed);int nshuffle=rand()%10;int e1,e2;

 for(int cnt=0;cnt<=nshuffle;cnt++){
   e1=1+rand()%nelt;
   e2=1+rand()%nelt;
   swap_elts(e1,e2);
  }
}
```

Appendix B

C++ for Pre-process Weight

```
void spikeNN_multi::preprocess_weight(const char *opt_train)
{
 double weight;
 double twon[16]; //array of 3 variables twon: total weight output neuron
 double totalow; // total output weight
 double ttow_gr[6]; double Sum_totalw; double Average; double dif_ttow;
 double dist_diftw[6];

 for(int r=1;r<=N_train;r++){
  init_spikeNN();
       train_matrix->get_row_Vector(r,1,N_attrib,*inpv);

  t=0;
  do{
   t=t+dt;
   update_synapse_potential();
   update_output_pw();
  }while(t<twindow);

  prefind_winner();  // add on 5/8/05 for classifying
  Sum_totalw=0;
       for(int j=1;j<=N_output;j++){
       totalow=0;
    for(int i=2;i<=N_attrib;i++ ){

                for(int k=1;k<=N_subc;k++)
      {
                        weight=wght->getelt(i,j,k);
                        wght->setelt(i,j,k,weight);
                   twon[k] = weight;
                        totalow += twon[k];
                                        }
                                                }
                ttow_gr[j] = totalow;
                Sum_totalw += ttow_gr[j];
                                }
                Average = Sum_totalw / N_output;

                for(j=1;j<=N_output;j++){

                        if (ttow_gr[j] > Average)
                                dif_ttow = ttow_gr[j] - Average;
                        else
                                dif_ttow = Average - ttow_gr[j];

                        dist_diftw[j] = dif_ttow / ((N_attrib-1)*N_subc);
                }

  for(int i=2;i<=N_attrib;i++)
    for(int j=1;j<=N_output;j++){
```

```
    if (j==winner)

     for(int k=1;k<=N_subc;k++)
     {
                    weight=wght->getelt(i,j,k);
                    weight=weight - dist_diftw[j];
               wght->setelt(i,j,k,weight);
     }
          else
               for(int k=1;k<=N_subc;k++)
     {
      weight=wght->getelt(i,j,k);
                    weight=weight + dist_diftw[j];
      wght->setelt(i,j,k,weight);}
}
}
}
```

Appendix C

Procedure for Basic Bees Algorithm

1. Initialise population with random solutions.
2. Evaluate fitness of the population.
3. While (stopping criterion not met)

   // Forming new population.
4. Select elite bees and elite sites for neighbourhood search.
5. Select other sites for neighbourhood search.
6. Recruit bees around selected sites (more bees for best elite sites) and evaluate fitnesses.
7. Select the fittest bee from each site.
8. Assign remaining bees to search randomly and evaluate their fitnesses.
9. End While.

REFERENCES

Adrian, E. D. (1926). "The impulse produced by sensory nerve endings." Journal of Physiology, London, vol. 61, pp.49-72.

Abeles, M. (2002) "Synfire chain." The handbook of Brain theory and neural networks, MIT Press, New York, USA, pp. 1143-1146.

Hassan, A. (2002) "On-line recognition of developing control chart patterns". PhD Thesis, Universiti Teknologi Malaysia.

Abeles, M., Domany, E., Schulten, K. and van Hemmen, J.L., (1994) "Firing rates and well-timed events" Models of Neural Networks 2, Springer, New York, chapter 3, pp. 121-140.

Ammar, N., Nelis, E., Merlini, L., Barisic, N., Amouri, R., Ceuterick, C., Martin, J.J., Timmerman, V., Hentati, F., and De Jonghe, P. (2003) "Identification of novel GDAP1 mutations causing autosomal recessive Charcot-Marie-Tooth disease". Neuromuscul Disord 13, pp 720-728.

Baig, S., and Liechti, P.A. (2001). "Ozone treatment for bio refractory COD removal" Water Science and Technology, vol. 43(3), pp. 197-204

Barch, D.M., (2005). "The relationships among cognition, motivation, and emotion in schizophrenia: How much and how little we know". Schizophrenia Bulletin, vol. 31(4), pp. 875-881.

Belatreche, A., Maguire, L.P., McGinnity, M., and Wu, Q.X., (2003). "An evolutionary strategy for supervised training of biologically plausible neural networks". The Sixth International Conference of Computational Intelligence and Natural Computing, Cary, North Carolina,USA, pp. 1524-1527.

Berridge, K.C., (2004). "Motivation concepts in behavioural neuroscience". Physiology and Behaviour, vol. 81, pp.179-209

Bialek, W., Rieke, F., de Ruyter, RR, and Warland, D. (1991)."Reading a neural code". Science, vol. 252(5014), pp. 1854-1857.

Bonabeau, E., Dorigo, M., and Theraulaz, G., (1999). "Swarm intelligence: from Natural to artificial systems". New York: Oxford University Press.

Buonomano, D.V., and Merzenich, M. (1999). "A neural network model of temporal code generation and position invariant pattern recognition". Neural Computation, vol. 11, pp.103-116.

Bohte, S.M., Kok, J.N., and Poutre, H.L., (2000). "Unsupervised classification in a layered network of spiking neurons" Proceedings of IJCNN, vol. iv, pp.279-285.

Bohte, S.M., (2004). "The evidence for neural information processing with precise spike-times: A survey". Natural Computing, vol. 3(4), pp. 195-206.

Camazine, S., Deneubourg, J.L., Franks, N.R., Sneyd, J., Theraula, G., and Bonabeau, E. (2003). "Self-organisation in biological systems". Princeton:Princeton University Press.

Carlisle, A. and Dozier, G. (2002). "Tracking changing extreme with adaptive particle swarm optimiser". Proceedings of the Fifth Biannual World Automation Congress, Orlando, Florida, USA, pp. 265-270.

Cheng, C. S. (1995). "A multi-layer neural network model for detecting changes in the process means". Computers and Industrial Engineering, vol. 28, pp. 51-61.

Cheng, C. S and Hubele, N. F. (1992) "Design of knowledge-based expert systems for statistical process control". Computers and Industrial Engineering, vol. 22(4), pp. 501-517.

Cheng, C. S. (1997) "A neural network approach for the analysis of control chart patterns". International Journal of Production Research, vol. 35(3), pp. 667-697.

Cheng, C., (1989). "Group technology and expert system concepts applied to statistical process control in small batch manufacturing" PhD Dissertation, Graduate College, Arizona State University, Tempe, AZ.

Cheng, C. S. and Hubele, N. F. (1996). "A pattern recognition algorithm for an X-Bar Control Chart." IIE transactions, vol. 28, pp. 215-224.

Chang, S. I. and Aw, S. A. (1996). "A neural fuzzy control chart for detecting and classifying process means shifts" International Journal of Production Research, vol.34 (8), pp.2265-2278.

Cullen, J. and Bryman, A. (1988). "The knowledge acquisition bottleneck: Time foe reassessment" Expert Systems, vol. 5(3), pp.216-225.

Cheng, C.S., Hubele, N.F. (1989), "A framework for rule-based deviation recognition systems in statistical process control." Proceedings of 1989 International Industrial Engineering Conference, Toronto, Canada, pp.617-682.

Cooper, D.C., (2002). "The significance of action potential bursting in the brain reward circuit". Neurochemistry International, vol. 41, pp.333-340.

Diesmann, M., Gewaltig, M. O. and Aertsen, A. (1999) "Stable propagation of synchronous spiking in cortical neural networks" .Nature, vol. 402, pp. 529-533.

DeSieno, D., (1988). "Adding a conscience to competitive learning". Proceeding of the International Joint Conference on Neural Networks, San Diego, California, vol. 1, pp. I-117-I-124.

Dorigo, M., and Stutzle, T., (2004). "Ant colony optimisation". Cambridge: MIT Press.

Eberhart, R., Shi, Y. and Kennedy, J. (2001). "Swarm intelligence". San Francisco: Morgan Kaufmann.

Fine, G.A. (1983). "Shared Fantasy". Chicago: The University of Chicago Press, pp. 186.

Frisch, K.V., (1976). "Bees: Their vision, chemical senses and language. Revised Edition". Ithaca, New York: Cornell University Press.

Garvin, D. A. (1987) "Competing in the eight dimensions of quality". Harvard Business Review, Sep. – Oct.

Goldberg, D.E., (1989). "Genetic Algorithms in search, optimisation and machine learning". Reading: Addison-Wesley Longman.

Guo, Y and Dooley, K. J. (1992). "Identification of change structures in statistical process control". *International Journal of Production Research*, vol. 30, 1655-1669.

Guh, R. S, Tannock, J. D. T and O'Brien, C. (1999a). "IntelliSPC: A hybrid intelligent tool for on-line economical statistical process control". Expert Systems with Application, vol. 17, pp. 195-212.

Guh, R. S and Tannock, J. D. T. (1999). "Recognition of control chart concurrent patterns using a neural network approach". International Journal of Production Research, vol. 37 (8), pp. 1743-1765.

Gerstner, W. (1995). "Time structure of the activity in neural network models". Physical Review, vol. 51, pp. 738-758.

Gerald M. Edelman (1993). "Neural Darwinism: Selection and reentrant signaling in higher brain function". Neuron, vol.10, pp. 115-125.

Gerstner, W. and Van Hemmen, J.L. (1994). "How to describe neuronal activity: spikes, rates, or assemblies?" Advances in Neural Information processing Systems, vol. 6, pp. 463-470.

Gerstner, W., Van Hemmen, J.L., and Cowan, J.D., (1996). "What matters in neuronal locking" Neural Computation, vol. 8(8), pp. 1653-1676.

Gerstner, W., and Kistler, W.M. (2002). "Spiking neuron models". Cambridge University Press, New York.

Hwang, H. B. and Hubele, N. F. (1993). "Back-propagation pattern recognisers for X-bar control charts: methodology and performance". Computers and Industrial Engineering, vol. 24(2), pp. 219-235.

Hwang, H. B. (1992). "Pattern recognition on Shewhart control charts using a neural network approach. Arizona State University: PhD Dissertation.

Haykin, S. (1999). "Neural networks: A comprehensive foundation." 2nd ed. Upper Saddle River, N. J. Prentice Hall.

Hopfield, J. (1995). "Pattern recognition computation using action potential timing for stimulus representation." Nature, vol. 376, pp. 33-36.

Hubel, D.H (1988). "Eye, Brain, and Vision." New York, WH Freeman.

Hubel, D.H., and Wiesel, T.N. (1962). "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex." Journal of Physiology (London), vol. 160, pp. 106-154.

Hwang, H.B. and Hubele, N.F. (1993a). "Back-propagation pattern recognizers for X-Bar control charts; Methodology and performance." Computers Industrial. Engineering, vol. 24(2), pp 219-235.

Hwang, H.B. and Hubele, N.F. (1993b). "X-bar control chart pattern identification through efficient off-line neural network training." IIE Transaction, vol. 25(3), pp. 27-40.

Hwang, H.B., Chong, C.W. (1995). "Detecting process non-randomness through a fast and cumulative learning ART-based pattern recogniser." International Journal of Production Research, vol 33(7), pp. 1817-1833.

Hodgkin, A.L and Huxley, A.F. (1952). "Hodgkin and Huxley model" Journal of Physiology, vol. 116, pp. 449-566.

Hubel, D. H., and Wiesel, T. N. (1959). "Receptive fields of single neurons in the cat's striate cortex" Journal of Physiology, vol. 148, pp.574-591.

Hopfield, J.J. (1995) "Pattern recognition computation using action potential timing for stimulus representation" Nature vol. 376, pp.33-36.

Hahnloser, R., Sarpeshkar, R., Mahowald, M.A., Douglas, R.J., and Seung, H.S., (2000). "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit". Nature, vol. 405, pp. 947-951.

Ikegaya, Y., Aaron, G. Cossart, R., Aronov, D., Lampl, I., Ferster, D., and Yuste, R. (2004). "Synfire chains and cortical songs: Temporal modules of cortical activity." Science, vol. 304(5670), pp. 559-564.

Jacob, D.A. and Luke, S.R. (1993). "Training artificial neural networks for statistical process control." The Tenth Biennial University Government Industry Microelectronics Symposium IEEE, Piscataway, NJ, USA, pp. 235-239.

Jianguo, X., and Embrechts, M.J. (2001). "Supervised learning with spiking neural networks" Proceedings of Neural Networks. IEEE International Joint Conference, vol. 3, pp. 1772-1777.

Jin, D.Z. and Seung, H.S. (2002). "Fast computation with spikes in a recurrent neural network". Physical Review E, vol. 65(5), pp.051922-1-051922-4.

Kahraman, C., Tolga, E. and Ulukan, Z. (1995). "Using triangular fuzzy numbers in the tests of control charts for unnatural patterns". Proceedings of INRIA/IEEE Symposium on Emerging Technologies and Factory Automation, vol. 3, pp. 291-298.

Kasinski, A., and Ponulak, F., (2006). "Comparison of supervised learning methods for spike time coding in spiking neural networks". International Journal of Applied Math and Computing Science, vol. 16(1), pp. 101-113.

Koch, C. (1999). "Biophysics of Computation." Oxford University Press, New York.
Kohonen, T. (1984). "Self-Organisation and associative memory." Berlin etc.: springer-Verlag

Kohonen, T. (1988). "Learning vector quantisation". Neural Networks, vol. 1(303).

Kohonen, T. (1990) "Improved versions of learning vector quantisation". Neural Networks, IJCNN International Joint Conference, San Diego, CA, USA.

Kunkle, D.R., and Merrigan, C., (2005). "Pulsed neural networks and their application". Neurobot: A Computation Neuroscience Weblog, Rochester Institute of Technology, Rochester, New York.

Lippmann, R. P., (1989). "Pattern classification using neural networks". IEEE Communications magazine, pp. 47-64.

Litvak, V., Sompolinsky, H., Segev, I., and Abeles, M. (2003). "On the transmission of rate code in long feed forward networks with excitatory-inhibitory balance." Journal of Neuroscience, vol. 23(7), pp. 3006.

Lippmann, R.P. (1991) "A critical overview of neural network pattern classifiers" Neural Networks for Signal Processing, Proceedings of the 1991 IEEE Workshop, pp. 266-275.

Lee, B.B, Dacey, D. M., Smith, V. C., and Pokorny, J. (1999). "Horizontal cells reveal cone type-specific adaptation in primate retina". Proc Natl Acad Sci U S A, vol. 96(25) pp.14611-14616.

Lai, T L. (1995). "Sequential change point detection in quality control and dynamical systems (with discussion)". Journal of the Royal Statistical Society, Ser. B, vol. 57, pp. 613-658.

Love, P.L. and Simaan, M. (1989) "A knowledge –based system for the detection and diagnisos of out-of-control events in manufacturing processes". American Control Conference, Pittsburgh, PA, vol. 3, pp. 2394-2399.

Mass, W. (1997a). "Networks of spiking neurons: the third generation of neural network models. Neural Networks, vol. 10, pp. 1659-1671.

Mass, W., and Bishop, C. M (2001)." Pulsed neural networks". MIT Press, MA, USA.

Mel, B.W. (1993). "Synaptic integration in an excitable dendrite tree." Journal of Neurophysiology, vol. 70, pp. 1086-1101.

Maass, W. (1997a). "Fast sigmoidal networks via spiking neurons" Neural Computation, vol. 9(2), pp. 279-304.

Milner, P.M. (1974). "A model for visual shape recognition." Psychological Review, vol. 81, pp. 521-535

Mao, B., Wu, W., Li, Y., Hoppe, D., Stannek, P., Glinka, A., Niehrs, C. (2001). "LDL-receptor-related protein 6 is a receptor for Dickkopf proteins" Nation, vol. 41(6835), pp. 321-325.

Mazurek, M.E., and Shadlen, M.N., (2002). "Limits to the temporal fidelity of cortical spike rate signals" Natural Neuroscience, vol. 5(5), pp. 463-471.

Maass, W. (1997b). "Networks of spiking neurons: the third generation of neural network model" Neural Networks, vol. 10(9), pp. 1659-1671.

Maass W., (1996). "On the computational power of noisy spiking neurons" Advances in Neural Information Processing Systems 8. Proceedings of the 1995 Conference, MIT Press, Cambridge, MA.

Maass, W., and Zador, M.A., (1999). "Dynamic stochastic synapses as computational units" Neural Computation, vol. 11(4), pp. 903-917.

Maass, W. (2000). "On the computational power of winner-take-all". Electronic Colloquium on Computational Complexity, report no. 32.

Maass, W., Schnitger, G. and Sontag, E.D. (1991). "On computational power of sigmoid versus Boolean threshold circuits". Foundation of Computer Science. Proceedings, 32nd Annual Symposium, pp.767-776. San Juan, Puerto Rico.

Montgomery, D. C. (1997). "Introduction to statistical quality control (3rd ed.)". New York: Wiley.

Moore, S.C. (2002). "Back-propagation in spiking neural networks". M.Sc. thesis, University of Bath, available at: http://www.simonchristianmoore.co.uk.

Mountcastle, V.B. (1957). "Modality and topographic properties of single neurons of cat's somatosensory cortex". Journal of Neurophysiology, vol. 20, pp.408-434.

Natschlager, T., Maass, W., and Zador, M.A., (2001). "Efficient temporal processing with biologically realistic dynamic synapses" Network Computation in Neural systems, vol. 12(1), pp. 75-87.

Nager, C., Storck, J., and Deco, G. (2002) "Speech recognition with spiking neurons and dynamic synapses" Neurocomputing, vol. 44-46, pp. 937-942.

Niebur, E. and Koch, C. (1996). "Control of elective visual attention: Modelling the 'where' pathway". Advances in Neural Information Processing Systems vol. 8. pp. 802-808. MIT Press, Cambridge, MA.

Natschlager, T., and Ruf, B. (1998). "Spatial and temporal pattern analysis via spiking neurons" Network Computation in Neural Systems, vol. 9(3), pp. 319-332.

O'Keefe, J., and Recce, M. (1993) "Phase relationship between hippocampal place units and the hippocampul theta rhythm" Hippocampus, vol. 3, pp. 17-330.

Pandya, A. S and Macy, R. B. (1995). "Pattern recognition with neural networks in C++". IEEE Press

Perry, M B; Spoerre, J K; and Velasco, T. Control chart pattern recognition using artificial neural networks. International Journal of Production Research, vol. 39(15), pp. 3399-3418, 2001

Pham, D.T, and Oztemel, E. (1992). "Control chart pattern recognition using neural networks" Journal of Systems Engineering.

Pham, D. T and Oztemel, E. (1992a)."XPC: An on-line expert systems for statistical process control". International Journal of Production Research, vol. 30 (12), pp. 2857-2872.

Pham, D. T and Oztemel, E.(1992b). "Tempex: An expert systems for temperature control in an injection moulding process". Quality and Reliability Engineering International, vol. 8, pp. 9-15, 1992b.

Pham and Oztemel, (1993a). "Combining multi-layer perceptrons with heuristics for reliable control chart pattern classification". Application of Artificial Intelligence in Engineering Conference Proceedings, pp. 801-810.

Pham and Oztemel, (1993b). "Control chart pattern recognition using combinations of multi-layer perceptrons and learning vector quantisation networks". Journal of Systems and Control Engineering, Proceeding Institute Mech. Engrs.,pp. 113-118.

Pham, D.T and Oztemel, E. (1994). "Control chart pattern recognition using learning vector quantization networks". International Journal of Production Research,vol. 32, pp. 721-729, 1994.

Pham, D.T and Oztemel, E. (1996). "Intelligent quality systems". Springer-Verlag, London, UK, ISBN 3-540-76045-8, 201pp.

Pham, D.T., and Wani, M.A., (1997). "Feature-based control chart pattern recognition" International Journal of Production Research, vol. 35(7), pp. 1875-1890.

Pham , D.T., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S., and Zaidi, M. (2006). "The Bees Algorithm, a Novel tool for complex optimisation problems". Proceeding 2nd International Virtual Conference on Intelligent Production Machines and Systems (IPROMS'06). Oxford, Elsevier, pp. 454-459.

Pham, D.T., and Sahran, S., (2006) "Control chart pattern recognition using spiking neural network". Proceeding 2nd International Virtual Conference on Intelligent Prouction Machines and Systems (*IPROMS 2006*), Oxford: Elsevier, pp. 319-325.

Pham, D.T., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S., and Zaidi, M. (2005). "Technical note: Bees Algorithm". Manufacturing Engineering Centre, Cardiff University: Cardiff.

Ryan, T P. Statistical methods for quality improvement (2nd edition.) New York: Wiley, 2000.

Reinagel, P., and Reid, R.C. (2002). "Precise Firing Events Are Conserved across Neurons" Journal of Neuroscience, vol. 22(16), pp. 6837–6841.

Riehle, A., Grun, S., Diesmann, M., and Aertsen, A. (1997). "Spike synchronization and rate modulation differentially involved in motor cortical function" Science, vol. 278(5345), pp. 1950-1953.

Rieke, F., Warland, D., de Ruyter van Steveninck, R., and Bialek, W. (1996). "Spikes-exploring the neural code". MIT Press, Cambridge, MA.

Riesenhuber, M. and Poggio, T. (1999). "Hierarchical models of object recognition in cortex". Nature Neuroscience, vol. 2(11), pp. 1019-1025.

Rowlands, H. and Wang, L.R. (2000). "An approach of fuzzy logic evaluation and control in SPC". International Journal of Quality and Reliability Engineering, vol. 16, pp. 91-98.

Ruf, B. and Schmitt, M. (1998). "Self-organisation of spiking neurons using action potential timing." Neural Networks, IEEE Transactions, vol. 9(3), pp. 575-578.

Salzman, C.D. and Newsome, W.T. (1994). "Neural mechanisms for forming a perceptual decision". Science, vol. 264(5156), pp. 231-237.

Schalkoff, R.J. (1997). "Artificial neural networks". Computer Science Series" McGraw-Hill Co. Inc. New York.

Seeley, T.D., (1996). "The wisdom of the hive: The social physiology of honey bee colonies". Cambridge, Massachusetts: Harvard University Press.

Shewhart, W. A. (1931). "Economic Control of Quality of Manufactured Products". New York: Van Nostrand, 1931

Shadlen, M.N. and Newsome, W.T. (1998). "The variable discharge of cortical neurons: implications for connectivity, computation and information coding". Journal of Neuroscience, vol.18, pp. 3870-3896.

Simaan, M. and Love, P.L. (1990). "Knowledge-based detection of out-of-control outputs in process control". Proceeding of the 29th IEEE Conference on Decision and Control, Honolulu, Hawaii, pp.128-129.

Schrauwen, B., and Van Campenhout, J., (2004). "Extending spikeprop" Proceedings of Neural Networks. IEEE International Joint Conference, vol. 1, pp. 475.

Shadlen, M. N., and Newsome, W. T. (1994). "Noise, neural codes and cortical organization" Current Opinion in Neurobiology, vol. 4, pp. 569-579.

Softky, W. R. (1995). "Simple codes versus efficient codes" Current Opinion in Neurobiology, vol. 5, pp.239-247.

Spears, W.M., Jong, K.A.D., Baeck, T., Fogel, D.B., and de Garis H., (1993). "An overview of evolutionary computation". Proceeding European Conference Machine Learning, Vienna, Austria, vol. 667, pp. 442-459.

Swift, J.A., and Mize, J. H., (1995). "Out-of-control pattern recognition and analysis for quality control charts using LISP-based systems". Computers and Industrial Engineering, vol. 28(1), pp. 81-91.

Swift, J.A. (1987). "Development of a knowledge-based expert system for control chart pattern recognition analysis". PhD dissertation, Graduate College, Oklahoma State University, Stillwater, Oklahoma.

Thorpe, S., Fize, D., and Marlot, C., (1996). "Speed of processing in the human visual system". Nature, vol. 381, pp.520-522.

Thorpe, S.J., Delorme A. and VanRullen R., (2001). "Spike-based strategies for rapid processing" Neural Network, vol.14 (6-7), pp. 715-726.

Tino, P. and Mills, A.J. (2005)."Learning beyond finite memory in recurrent networks of spiking neurons". Advances in Natural Computation-ICNC 2005, (L. Wang K.

Chen and Y. Ong, Eds.), Lecture Notes in Computer Science, Berlin, Springer, pp. 666-675.

Tolman, E.C., (1948). "Cognitive maps in rats and men [1]". The Psychological Review, vol. 55(4), pp. 189-208. (Classics in the history of psychology: An internet resource developed by Christopher D. Green, York University, Toronto, Ontario.

Turban, E. (1995). "Decision support and expert systems: management support systems". Prentice-Hall, Inc. Upper Saddle River, NJ, USA.

Van Rullen R., Guyonneau R. and Thorpe S.J., (2005). "Spike times make sense" TRENDS in Neuroscience, vol. 28(1), pp. 1-4.

Villa et al.(1999). "Spatial-temporal activity patterns of rat cortical neurons predict responses in a conditional task". Proc Natl Acad Sci USA. vol. 96, pp. 1106-1111.

Wolf, E., Zhao, F.Y., and Roberts, A. (1998). "Non-linear summation of excitatory synaptic inputs to small neurones: a case study in spinal motoneurones of the young *Xenopus* tadpole" The Journal of Physiology, vol. 511(3), pp. 871-886.

Wei, C.H., and Fahn, C.S. (2002). "The multisynapse neural network and its application to fuzzy clustering" IEEE Transaction on Neural Networks, vol. 13(3), pp. 600-618.

Xin, J. and Embrechts, M.J. (2001). "Supervised learning with spiking neuron networks". Proceeding IEEE Int. Joint Conf. Neural Networks, IJCNN'01, Washington D.C., pp.1772-1777.

Xu, X. and Eberhart, R.C. (2002a). "Adaptive particle swarm optimisation: Detection and response to dynamic Systems". Proceedings of the IEEE Congress on Evolutionary Computation, Honolulu, Hawaii USA, pp. 1666-1670.

Xu, X. and Eberhart, R.C. (2002b). "Multi-objective optimisation using dynamic neighbourhood particle swarm optimisation". Proceedings of the IEEE Congress on Evolutionary Computation, Honolulu, Hawaii USA, pp. 1677-1681.

Wang, L.R. and Rowlands, H. (1999) "A fuzzy logic application in SPC evaluation and control". Proceeding of 7th IEEE International Conference on Emerging Technologies and Factory Automation, vol. 1, pp. 679-684.

Zahedi, F. (1990) "A method of quantitative evaluation of expert systems" European Journal of Operational Research, vol. 48, pp. 136-147.

Zador A.M, (2000). "The basic unit of computation" Nature Neuroscience, vol. 3, pp. 1167.