BINDING SERVICES
Tel +44 (0)29 2087 4949
Fax.+44 (0)29 2037 1921
E-Mail Bindery@Cardiff.ac.uk

Supervised and unsupervised weight and delay adaptation learning in temporal coding spiking neural networks

A thesis submitted to the Cardiff University,
for the degree of
Doctor of Philosophy

by

Eugene Yougarajah Andrew Charles

Manufacturing Engineering Centre
Cardiff University

2006

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

Abstract

Artificial neural networks are learning paradigms which mimic the biological neural system. The temporal coding Spiking Neural Network, a relatively new artificial neural network paradigm, is considered to be computationally more powerful than the conventional neural network. Research on the network of spiking neurons is an emerging field and has potential for wider investigation. This research explores alternative learning models with temporal coding spiking neural networks for clustering and classification tasks.

Neurons are known to be operating in two modes namely, as integrators and coincidence detectors. Previous temporal coding spiking neural networks, realising spiking neurons as integrators, were utilised for analytical studies. Temporal coding spiking neural networks applied successfully for clustering and classification tasks realised spiking neurons as coincidence detectors and encoded input information in the connection delays through a weight adaptation technique. These learning models select suitably delayed connections by enhancing the weights of those connections while weakening the others. This research investigates the learning in temporal coding spiking neural networks with spiking neurons as integrators and coincidence detectors. Focus is given to both supervised and unsupervised learning through weight as well as through delay adaptation.

Three novel models for learning in temporal coding spiking neural networks are presented in this research. The first spiking neural network model, Self-Organising Weight Adaptation Spiking Neural Network (SOWA_SNN) realises the spiking neuron as integrator. This model adapts and encodes input information in its connection weights. The second learning model, Self-Organising Delay Adaptation Spiking Neural Network (SODA_SNN) and the third model, Supervised Delay Adaptation Spiking Neural Network (SDA_SNN) realise the spiking

neuron as coincidence detector. These two models adapt the connection delays in order to detect temporal patterns through coincidence detection. The first two models were developed for clustering applications and the third for classification tasks. All three models employ Hebbian-based learning rules to update the network connection parameters by utilising the difference between the input and output spike times.

The proposed temporal coding spiking neural network models were implemented as discrete models in software and their characteristics and capabilities were analysed through simulations on three bench mark data sets and a high dimensional data set. All three models were able to cluster or classify the analysed data sets efficiently with a high degree of accuracy. The performance of the proposed models was found to be better than the existing spiking neural network models as well as conventional neural networks. The proposed learning paradigms could be applied to a wide range of applications including manufacturing, business and biomedical domains.

to the loving memory of my father .....

Acknowledgements

I would like to express my deep and sincere gratitude to my supervisor, Dr. M. S. Packianather. His wide knowledge and logical way of thinking have been of great value for me. His understanding, encouraging and personal guidance have provided a good basis for the present thesis. I am deeply grateful to my supervisor, Professor D. T. Pham, for his detailed and constructive comments, and for his important support throughout this work. My sincere thanks also goes to Professor S. S. Dimov for his suggestions and comments throughout this work.

I wish to express my warm and sincere thanks to Dr. S. Mahesan for his tireless and generous effort which gave me the opportunity to do this research. My sincere thanks also belong to Dr. S. Kanaganathan. I wish to express my thanks to all the staff, colleagues and friends of University of Jaffna, Sri Lanka.

I would like to thank all my colleagues specially for the Data mining group members, for their guidance and company. My sincere thanks also to all the staff of the Manufacturing Engineering Centre. I wish to express my special thanks to all my friends in Cardiff, London and elsewhere.

My special gratitude is due to my mother, aunty, brother and my sister for their love and encouragement.

Contents

List of Figures

List of Tables

List of Symbols

δd_{ji}	The amount of change for delay to the connection from neuron i to j
ϵ	The spike response function
η	Learning rate
Γ_j	Set of neurons pre-synaptic to neuron j
$\mathbf{N}$	Set of all natural numbers
$\mathbf{R}^+$	Set of all positive real numbers (excluding zero)
$\mathbf{R}$	Set of all real numbers
$\mathcal{F}_i$	Set of all previous firing times of neuron i
ϕ	Activation function
τ	Time constant for the spike response function
τ_{stdp}	Synaptic time constant for potentiation and depression of the learning rule
ζ_j	Refractory function of neuron j
b_j	Bias value for neuron j
d_{ji}	Delay value of the connection from neuron i to j
dt	Time step between the adjacent states of simulation
dw_{ji}	Change in weight for the connection from neuron i to j
E	Set of synapses
N	A network of neurons
t	time
t_{input_window}	Input time window
t_{window}	Activation time window
$u_j(t)$	Potential or state variable of neuron j at time t

V	Set of neurons
v_j	Linearly combined output of the weighted input signals to neuron j
w_{ji}	Strength or weight of the connection from neuron i to j
x_i	Input value to neuron i
y_j	Output of neuron j
$\|\mathbf{w}\|$	Euclidean norm of $\mathbf{w}$

Chapter 1

Introduction

Artificial neural networks (ANNs) are one of the most powerful and flexible computing paradigms and a well studied area in artificial intelligence. The important property of an ANN is its ability to learn from the environment and to retain information. The development of ANN was inspired by the principles of biological neural networks which process information in an entirely different way from conventional methods. The modern era of ANNs was initiated with the pioneering work of McCulloch and Pitts [McCulloch and Pitts, 1943]. From then on, the research on ANNs grew in different directions and led to the creation of various artificial neurons, network architectures and learning algorithms. As a result, ANNs have been applied successfully for solving diverse problems in numerous domains [Haykin, 1999].

In recent years, temporal coding Spiking Neural Networks (SNN), networks of more biologically realistic artificial neurons, are receiving wider attention. Results from past research show that the learning in spiking neural networks can achieve the accuracy of conventional training algorithms with potential room for improvement [Bohte et al., 2002a]. This research explores alternative learning approaches for temporal coding SNNs for clustering and classification tasks. This chapter presents the preliminaries regarding the SNNs and defines the objectives

of this research. The structure of this research is also outlined here.

1.1 Preliminaries

The human brain, the most interesting and important organ in the world, has revealed only a few of its secrets to humankind. Even though the exploration of the brain began 2500 years ago when Greek philosophers conceived the idea that the human beings have a mind and a soul, researchers managed to shed some light on the true nature of the brain only within the past hundred years. The study on biological neural systems is developing more rapidly and attracts people from various disciplines. The findings from these studies not only provide a better understanding of neural systems but also shape the research on artificial intelligence. Although the creation and development of ANNs were inspired by biological neural systems, ANNs are considered to be limited due to their simplistic structure and behaviour [Zador, 2000; Maass, 1997a].

The brain is composed of billions of simple computing elements called neurons which are connected in parallel with trillions of interconnections or synapses [Haykin, 1999]. Artificial neural networks are also structured similarly, with interconnected computational units. The interconnections between the neurons in a conventional artificial neural network are considered as passive entities. These synapses are regarded as simple linear entities whose essential role is in learning. But it is claimed by the neuro biologists that the biological synapses are not merely passive entities whose outputs are a linear function of their inputs. Instead, it has been recognised for a long time that the synapses are dynamic elements with complex non-linear behaviour [Zador, 2000].

Computation and communication within the biological networks are based on action potentials or spikes. The spikes are electrical pulses with a potential

of around 100 mV and last for one to two msec. The way the information is coded with these spikes is an ongoing debate. In general it is accepted that the coding could be either based on the rate of spikes or on the firing time of each spike. Artificial neural networks were developed based on the idea that the biological neurons communicate via the firing rate of spikes [Bohte, 2003]. Generally, the analogue values responsible for computation and communication in the conventional artificial neural networks are interpreted as the firing rates of biological neurons [Maass, 1997a]. In recent years, however, the notion of rate of spikes is claimed to be inadequate for the operation of fast neuronal events. On the other hand, spike time-based coding or temporal coding are found to be competent and widely accepted [Gerstner and Kistler, 2002].

The disputes over the structure and functionality of conventional artificial neural networks resulted in increased interest in temporally coding spiking neural networks. These networks incorporate the non-linear nature of the synapses and are capable of dealing with spike-time based coding. It has been proved that the networks of spiking neurons can simulate arbitrary feed-forward sigmoidal neural networks and thus approximate any continuous function [Maass, 1997a]. It was also proved that neurons that convey information by the timing of individual spikes are computationally more powerful than the classical neurons with sigmoidal activation function [Maass, 1997c]. Another feature of the spiking neurons is that even with a seemingly increased structural complexity they are relatively easier to implement in large neural networks [Bohte et al., 2002a]. Single spike-time based computing has also been suggested as a new paradigm for VLSI neural network implementation [Maass, 1996]. It was also observed that there is considerable opportunity for further research on SNN, with potential improvement in various directions [Maass, 2001b].

The connections of a spiking neural network are characterised by a weight

value and a delay mechanism which postpones the arrival of an input spike at the other end of the connection. It is understood that biological neurons operate in two modes, as integrators and as coincidence detectors [Konig et al., 1996]. Generally, artificial neurons are known to be integrators, but due to the connection delays, the spiking neurons exhibit the coincidence detection functionality [Maass, 2001a]. In this mode, a spiking neuron is activated only when it receives coinciding inputs. The identification of these two modes of operations of a spiking neuron is supported by several studies on biological neurons [Konig et al., 1996].

Temporal coding SNNs can be applied to the same types of tasks as conventional ANNs. Recently a number of researchers have carried out significant research on learning with spiking neural networks. In general, the learning models adapt the connection weights and attempt to encode the input information in the connection weights and delays. A significant number of learning models proposed in the past for spiking neural networks are based on the Hebbian rule [Natschläger and Ruf, 1998; Bohte et al., 2002b]. This is a preferred approach due to the nature of the spiking neurons' functionality and the spike-time based information coding strategy. Error gradient descent approaches have also been proposed and applied successfully [Bohte et al., 2002a]. A significant amount of research has targeted the pattern recognition, clustering and classification tasks as the main application for their proposed models [Ruf and Schmitt, 1997; Natschläger and Ruf, 1998; Ruf and Schmitt, 1998; Bohte et al., 2002b]. In addition, several examples can be found in the literature where the spiking neural networks have been applied to various other learning tasks such as function approximation [Iannella and Back, 2001], associative memory and speech recognition [Bohte and Kok, 2005].

1.2 Research objectives

The main objective of this research is to investigate the possible supervised and unsupervised learning strategies for temporal coding spiking neural networks and to develop alternative learning models. The focus of this research is on learning models which adapt network connection weights as well as delays for clustering and classification tasks. More details are given in section 2.7.

Most of the existing learning models for spiking neural networks are based on the learning models for conventional neural networks. But it is here claimed that for spiking neural networks, the learning strategies have to be developed distinctively in order to exploit the full potential of spiking neurons. However, this is relatively difficult to achieve without the knowledge of learning strategies of biological neural networks. Fortunately, there are plenty of research findings on learning in biological networks available which are yet to be incorporated in the development of artificial neural networks. Hence another objective of this research is to combine the knowledge gained from the valuable research on biological neural networks and the knowledge available from the existing ANN learning models in order to develop novel models for learning in spiking neural networks.

SNNs can be implemented in software as well as hardware. In this research, developing a suitable software platform is considered as an objective in which the spiking neural network learning models can be realised and analysed.

The research on learning in spiking neural networks is at a relatively early stage. The analysis on networks of spiking neurons is important for creating better network models and learning strategies. Providing analytical results on various aspects of the network models and learning strategies is also considered as an objective of this research.

1.3 Structure of the thesis

This study consists of six chapters and two appendices. Chapter 2 summarises the basics of temporal coding spiking neural network. This chapter starts with a description of artificial neural networks and methods through which they learn. A brief outline of the biological neural network is also included. Aspects of neuronal coding and various neuron models are discussed. The temporal coding spiking neural network is introduced and topics regarding computing with SNNs and realising the network are explained. Previous research on learning in spiking neural networks is summarised. Further, a description of clustering and classification, the main application of this study, is given along with the details of the data sets utilised. Finally, this chapter concludes with an outline of this research.

This research focuses on learning in spiking neural networks through adapting the connection weights as well as delays by supervised and unsupervised learning methods. Chapter 3 focuses on learning through weight adaptation in an unsupervised manner. This chapter briefly describes the Kohonen's self-organising map (SOM), which is a popular unsupervised learning model for artificial neural networks. Previous research on SNNs regarding the unsupervised and Hebbian-based learning is summarised along with the key findings on learning in biological neural networks. The Self-Organising Weight Adaptation Spiking Neural Network (SOWA_SNN) is introduced and the details regarding its implementation are given. Finally, the results of the analytical studies conducted on the proposed model are summarised and discussed.

Most of the previous research on learning with spiking neural networks adapts the connection weights. Chapter 4 proposes an unsupervised learning model for SNNs, in which the connection delays are adapted. Here the spiking neurons are realised as coincidence detectors, and delays and coincidence detection are

explained in this chapter. Previous work on delay adaptation learning is summarised. The details about the proposed Self-Organising Delay Adaptation Spiking Neural Network (SODA_SNN) are presented along with the implementation details. Finally, the description of the analytical studies performed on the proposed model is explained and the results obtained are summarised and discussed.

Chapter 5 deals with learning in spiking neural networks in a supervised manner, where the spiking neurons are realised as coincidence detectors, as in chapter 4. This chapter briefly describes supervised learning models in artificial neural networks and summarises previous supervised learning models for spiking neural networks. The novel Supervised Delay Adaptation Spiking Neural Network (SDA_SNN) is introduced and the details regarding its implementation are given. The details of the analytical studies conducted are presented and the chapter concludes with the summary of the results obtained along with the discussion.

Each chapter ends with a brief conclusion covering the topics discussed in that chapter. Chapter 6 gives the conclusions drawn form this whole research and presents some suggestions for future work.

Appendix A presents the source code of the software developed to implement the proposed learning models.

Appendix B provides the data sets used for training and testing the proposed models.

Chapter 2

The basics of Spiking Neural Networks

Spiking neurons are claimed to be the third generation of artificial neurons with *McCulloch-Pitts* [Haykin, 1999] neurons as the first generation and neurons with continuous activation functions (sigmoidal neurons) as the second generation [Maass, 1997b]. This chapter lays the foundation for this study by introducing the basics of SNNs. In order to get a better understanding of SNNs, topics regarding the neural network models of both artificial and biological systems and other related themes are explained in this chapter.

This chapter is structured as follows: section 2.1 introduces the concept of artificial neural networks and related topics; section 2.2 explains the biological neural system and how it forms the basis for artificial neural networks. Neuronal coding, i.e., the way in which the information is coded in biological neural systems is summarised in section 2.3, and neuron models are described in section 2.4. The spiking neural network is defined in section 2.5. This section includes details of computing with SNNs and the realisation of the network. The previous research on learning in SNNs is also summarised in this section. Section 2.6 introduces the application area of this research, namely, clustering and classification. The details of the data sets used for analytical studies are also described in this section.

Finally, section 2.7 defines the outline of this research.

2.1 Artificial Neural Networks

An artificial neural network (ANN) is an information processing paradigm made up of simple processing units analogous to a massively parallel distributed processor. The simple computational units are called neurons after biological neurons, which were the inspiration for this method of information processing [Haykin, 1999]. This relatively new computational tool has found its way into solving many complex problems successfully and efficiently. The important characteristics of ANNs are non-linearity, high parallelism, fault and noise tolerance, learning and generalisation capabilities [Jain et al., 1996].

2.1.1 Structure of the artificial neural network

The structure or topology of an ANN defines the way the neurons are placed in the network and the way in which they are connected to one another. The structure of the network plays an important role in information processing since it is closely linked to the learning algorithm used to train the network. Generally, in an ANN, neurons are placed in layers. A network can have more than one layer of neurons in addition to the input layer. The input layer is simply a set of non-processing nodes from where the inputs are fed to the network. Neurons in each layer are connected to neurons in other layers through a set of links. A typical structure of an ANN is shown in Figure 2.1. Although it is not shown in this figure, neurons within a layer can also be connected one to another through lateral connections. The output from a neuron is passed to the other neurons through these links. Depending on the direction of flow of information, the network is referred to as a *feed-forward network* or a *recurrent network*. In the former, the flow is in

the forward direction only and in the latter the flow can be both in forward and as well as backward direction. Each link or connection is characterised by a weight value, which is known as the *connection strength.* These connections are the pathways for the information within the network [Haykin, 1999]. The processing of information is performed by a neuron which is described in the next sub-section.

2.1.2 The artificial neuron

A neuron is the fundamental processing unit of an ANN as in a biological neural network. Figure 2.2 shows a formal base model of a neuron. The output of a neuron is the linear weighted sum of its inputs subjected to an activation function. Equations 2.1 and 2.2 define the output of an artificial neuron [Pham and Liu, 1999].

$$v_j = \sum_{i=1}^{n} w_{ji}\, x_i \; + \; b_j \tag{2.1}$$

$$y_j = \phi(v_j) \tag{2.2}$$

where x_i, $i = 1..n$ are the input values and b_j is the bias value. w_{ji}, $i = 1..n$ are the connection weights for neuron j from neuron i. Here n is the number of input neurons. v_j is the intermediate output which is passed to the activation function ϕ to produce the final output y_j.

There are several forms of activation functions from which a suitable one can be selected according to the target application. Common activation functions are *threshold function, piecewise linear function* and *sigmoidal function,* which are shown in figure 2.3 [Haykin, 1999]. The important feature of the artificial neural networks made up of these simple processing units is their ability to learn and retain information from their environment. Learning in ANNs is discussed in the next sub-section.

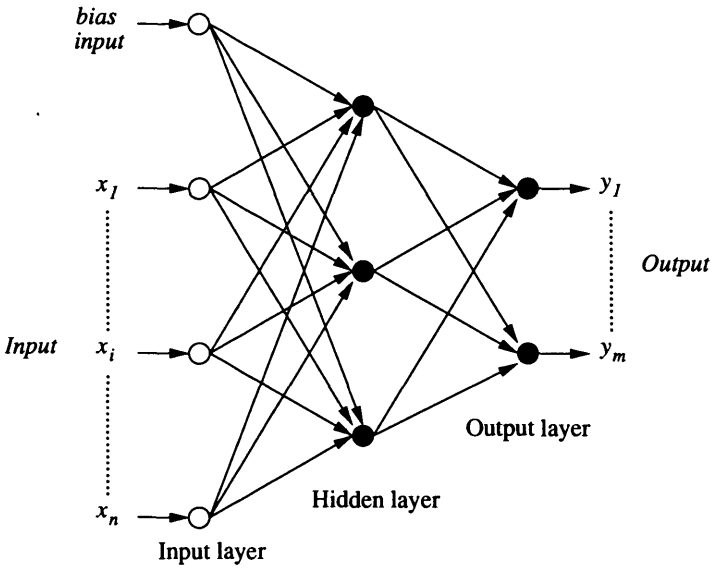


Figure 2.1. General structure of an artificial neural network.

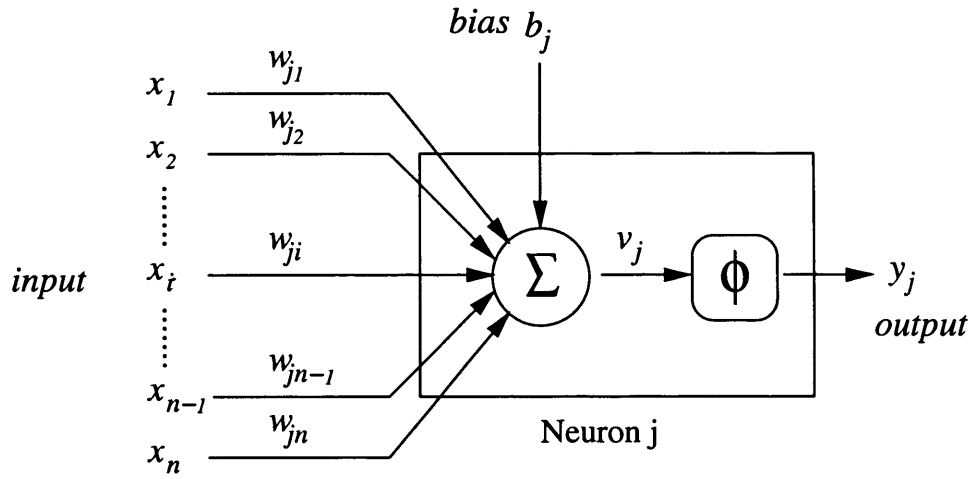
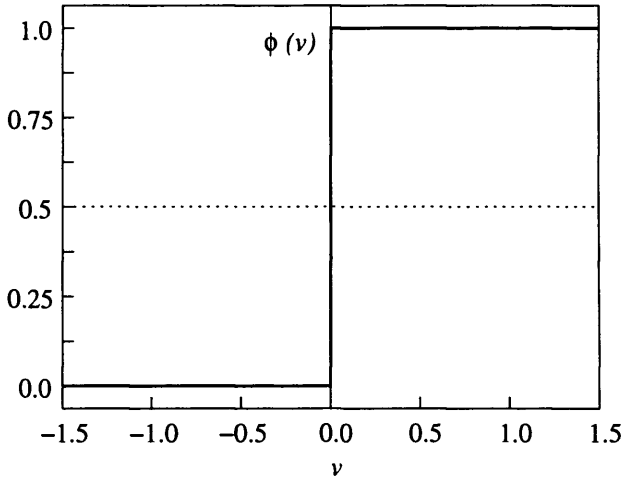
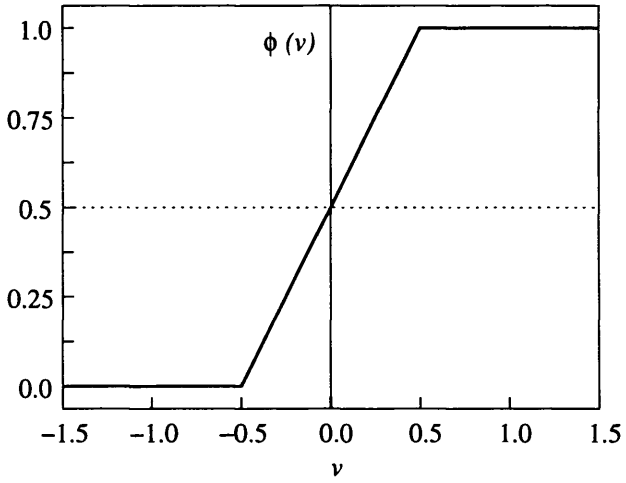


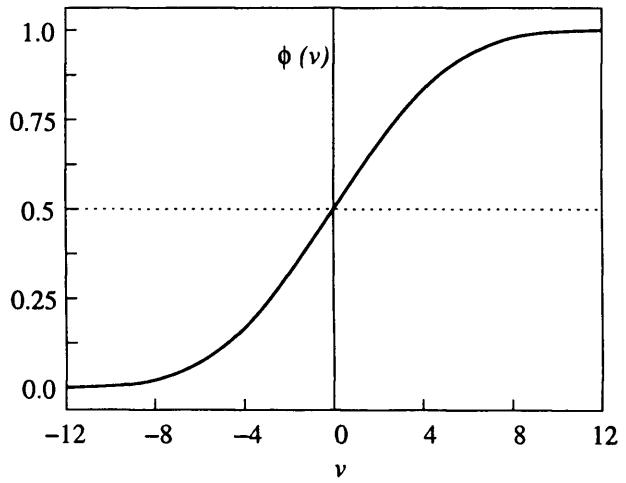
Figure 2.2. Schematic diagram of an artificial neuron. Redrawn from [Pham and Liu, 1999].



(a) Threshold function



(b) Piecewise linear function



(c) Sigmoidal function

Figure 2.3. Common activation functions. Redrawn from [Haykin, 1999].

2.1.3 Learning in artificial neural networks

The most significant property of the neural system is its ability to learn from its environment and so improve its performance. In the context of learning in artificial neural networks, learning can be defined as follows: *"Learning is a process by which the free parameters of a neural network are adapted through a process of simulation by the environment in which the network is embedded"* [Mendel and McLaren, 1970]. Likewise, an ANN also posses this important feature. With the aid of a learning procedure, the ANNs can extract and store information from the data made available to the network. The extracted information is stored in the network through the connection weights, and can be retrieved for future use. The learning in the ANNs can either be with a teacher (*supervised learning*) or without a teacher (*unsupervised learning*). In supervised learning the network is presented with a set of input-output examples. Based on the output for a given example, the teacher will specify a desired output which the network is expected to produce. The difference between the actual output and the desired output is called an *error signal.* The objective of the training procedure is to modify the network parameters in such a way that the network produces an output which is as close as possible to the desired output, thus reducing the error. The modification is based on both the input signal and the error signal [Haykin, 1999].

For unsupervised learning, the examples presented are not labelled and the learning is performed without any external supervision. There are two modes in this form of learning, namely, *self-organising learning* and *reinforcement learning.* In self-organising learning the network undergoes change of its parameters according to its learning rules without any supervision. The modifications to the network parameters are performed in such a way that the network automatically

discovers for itself any possibly existing patterns, regularities, separating properties etc. in the presented data [Zurada, 1999]. Even though the reinforcement learning comes under the category of unsupervised learning, it can be considered as a special case of supervised learning because of the use of a *critic* to control the learning. Unlike in the case of supervised learning, here the critic evaluates the quality of the output produced by the network for a given input signal. Modification to the network parameters are based on this criticism and the input signal [Pham and Liu, 1999].

Supervised and unsupervised learning are known as learning paradigms. In both models the actual modifications to the network parameters are performed through learning rules. There are five basic learning rules mentioned in the literature. They are *error-correction learning, memory-based learning, Hebbian learning, competitive learning* and *Boltzman learning* [Haykin, 1999]. Error-correction learning, as the name suggests, tries to correct an error estimation. For a particular training sample, the difference between the actual output of the network and a desired output is considered as the error. The learning is performed in such a way that the network is enabled to give an output as close as possible to the desired output, thus reducing the error.

Memory-based learning functions by storing all the past experience or training samples $(\mathbf{x}_i, d_i)$ explicitly in a large memory. Here $(\mathbf{x}_i, d_i)$ are correctly classified input-output samples. Classification of an unseen sample is done by retrieving and analysing a training sample from the stored memory which falls within the logical neighbourhood of this new sample [Haykin, 1999].

The Hebbian learning rule was proposed in a neuro-biological context. This rule, which was named in honour of Hebb, is the oldest and most popular among the five learning rules. The Hebbian learning rule was introduced to explain

learning in biological neural networks, which suggests that a particular connection will be strengthened if neurons on both ends are active simultaneously and persistently. In mathematical terms, Hebb's hypothesis can be described by equation 2.3.:

$$dw_{ji}(n) = \eta\, y_j(n)\, x_i(n) \tag{2.3}$$

where $dw_{ji}(n)$ is the change in strength for the connection from neuron i to j; $y_j(n)$ is the output of neuron j and $x_i(n)$ is an input. η is a learning rate parameter and n specifies some stage in the learning process.

In competitive learning the output neurons compete with each other to become active. *Winner-Take-All* is a popular example of this type of learning. Generally this learning rule is used for learning the statistical properties of the inputs [Haykin, 1999; Zurada, 1999].

Boltzmann learning is a stochastic process based on statistical mechanics. A neural network with Boltzmann learning is often called as a *Boltzmann machine.* Generally, this is a recurrent network and the neurons operate as binary nodes by being either in an *on* or *off* state. An energy function is accompanied with the machine which can measure the energy contained by the network. A neuron is selected at random and its state is flipped during the learning process. This is continued until some equilibrium state is reached [Haykin, 1999].

2.2 Biological background

The creation of the ANNs was inspired by the way biological neural systems process information [Jain et al., 1996]. For a better understanding, it is necessary to explore briefly the biophysical aspects of biological neural networks at a high level of abstraction regarding the processing of information. Over the past

hundred years, detailed knowledge about the structure and functionality of the neural system has been acquired due to the valuable research conducted in various disciplines. It is known in neuroscience that the brain, the centre of a nervous system, is mainly made up of nerve cells called *neurons* and *neuroglia* or simply *glia*, a glue like substance. Neurons are the elementary processing units and are connected to each other in a complex pattern. Glia is mainly responsible for giving the structural stabilisation and providing energy for neurons [Shepherd and Koch, 1990]. It has been estimated that human brain consists of approximately 10 billion neurons and 60 trillion connections [Haykin, 1999].

2.2.1 The neuron

Neurons, the structural and functional components of the neural systems, came to light in human history relatively recently. In 1836 Jan Purkinje, a Czech physiologist, published his observations of cells in the *cerebellum* (a region of brain responsible for the integration of sensory perception and motor output). However, his work showed little more than the nucleus and surrounding jelly-like material called *cytoplasm* that fills the cell. Otto Deiters of Bonn observed that two kinds of fibres arise from the nerve cell body in 1865. Later, in 1885, Camillo Golgi of Paria found a method to stain a nerve cell in its entirety to enable it to be visible. Based on the work by Golgi, in 1889 Santiago Raymon y Cajal, a Spanish histologist, visualised an entire nerve cell and proved that each nerve cell is an individual entity. In 1891, Wilhelm Waldeyer, a Professor of anatomy and pathology, applied the cell theory to the nerve cells and suggested the name *neuron* for the nerve cell. A more detailed description of the history of neurons can be found in [Shepherd and Koch, 1990].

Neurons differ in size and shape but in general can be described as shown in figure 2.4, which is a schematic diagram of the *pyramidal cell*, one of the most

common cortical neurons [Haykin, 1999]. The main functional components of a typical neuron are the *cell body* or *nucleus*, *dendrite* and *axon*. Dendrites are the receptors of a neuron. This tree like structure receives signals from other neurons and passes them to the nucleus. Neuronal signals are in the form of short electrical pulses called *action potentials* or *spikes*. The nucleus is the functional component which performs non-linear processing on its inputs and generates the output. The output signals, which are in the form of pulses, will propagate through the axon, which is the effector of the neuron. This tube-like component contains many branches and each branch terminates in a special component named a *synapse*. The signals carried along the axon will be replicated at each branch and will be passed onto the dendrites of other neurons through the synapses.

2.2.2 The synapse

The synapse is the connector of the terminal point of an axon branch and a dendrite. A signal from a *pre-synaptic neuron* will be passed onto a *post-synaptic neuron* through a synapse. It is common to refer to a sending neuron as the pre-synaptic neuron and the receiving neuron as the post-synaptic neuron. Figure 2.5 shows a chemical synapse, the most common synapse in the vertebrate brain. The terminal point of an axon branch and a dendrite is separated by only a small gap, called the *synaptic cleft*. When a spike arrives at this point, a series of biochemical steps will be triggered which will eventually pass the signal to the other side [Gerstner and Kistler, 2002]. Depending on the type of the synapse, the effect on the receiving side can be positive or negative. A synapse is not merely a passive device whose output is a linear function of its input, but is a dynamic element with complex non-linear behaviour [Zador, 2000].

Synapses play an important role in the learning and memory of a nervous system. The strength of a synapse determines the amount of excitation induced

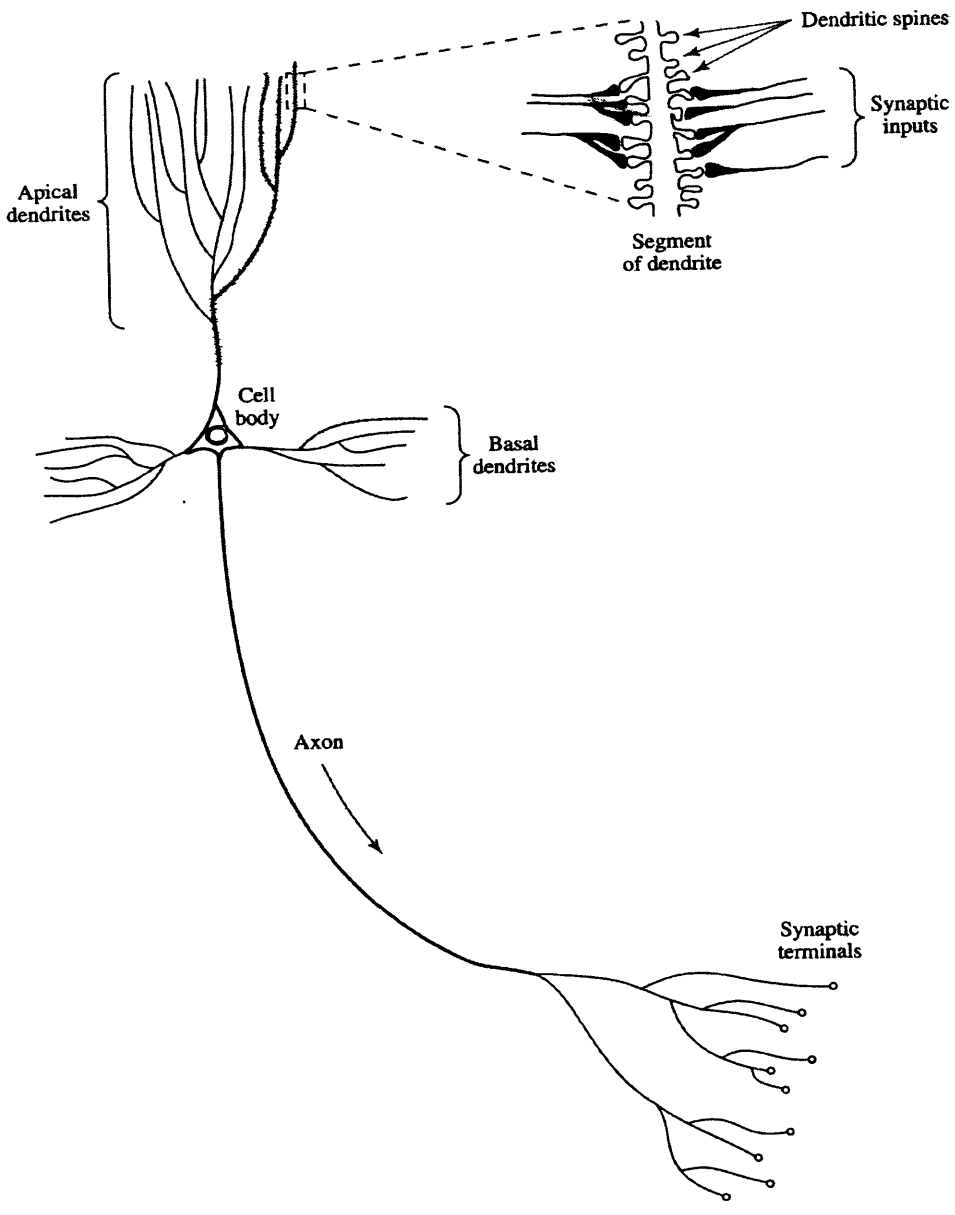


Figure 2.4. Schematic diagram of a neuron. From [Haykin, 1999].

by the synapse on a dendrite of a post-synaptic neuron for stimulation from a pre-synaptic neuron. The important aspect is that the strength of a synapse is modifiable through some molecular mechanisms. This phenomenon is known as *synaptic plasticity* [Gaiarsa et al., 2002]. Learning in neural systems is achieved through some mechanism which modifies the synaptic strength. These modifications will be retained by the synapses for either short term or long term depending on the molecular mechanism involved. This plasticity feature enables the learning and memory storage capability for a neural system [Shi et al., 1999]. Neuronal activity is a complex mechanism which is based on the flow of ions. The next section describes the electrical properties of the neurons and gives brief detail about neuronal activity.

2.2.3 Electrical properties of neurons

Neurons or nerve cells and their surrounding contain a huge number of ions and molecules in different varieties similar to other biological cells. Ions such as K^+, Na^+, Ca^{2+}, Mg^{2+}, Cl^- and organic anions (A^-) are the more commonly found components. The nerve cells are covered by a bilayer membrane which is almost impermeable to ions, hence blocking the free movement of ions and anions across the membrane. Generally, due to the difference in the concentration of ions, there is a net negative charge inside the cell and a net positive charge outside the cell. The ions repel each other and accumulate closer to the inside surface of the cell membrane. Due to electrostatic forces, positive ions will be attracted to the outer surface, as in a capacitor. Since the membrane is impermeable for ions, a potential difference will be maintained across it. This potential difference, called the *membrane potential*, is necessary to keep the nerve cell functioning. The neuronal pulses or spikes are the outcome due to the changes in the membrane potential. The entire surface of the cell membrane is not strictly impermeable

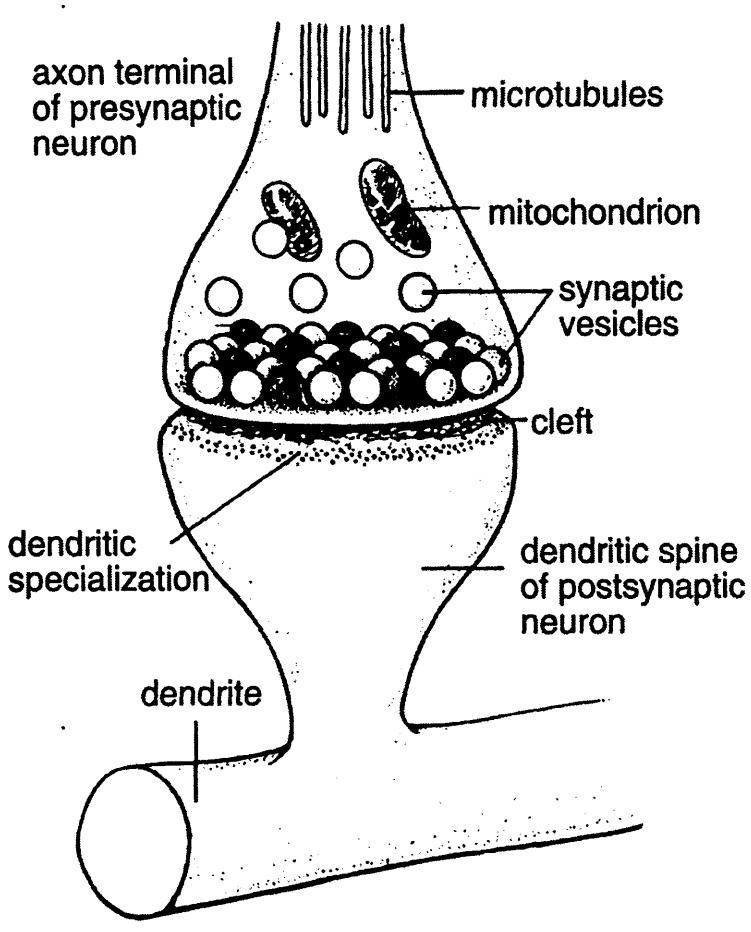


Figure 2.5. A chemical synapse. From [Dayan and Abbott, 2001].

to ions. On the surface there are embedded numerous passages. One type of passage, called *ion-channels*, will allow specific types of ions to pass through them. Another type of ion-channel embedded on the membrane is an ion pump. When activated, this type of ion channel can pump ions through the membrane into or out of the cell. The capacity of the channels for conducting ions can be modified by factors such as membrane potential, intracellular messengers and extracellular neuro-transmitters. When there are no external excitations, a neuron will be in an equilibrium state. The membrane potential at this state is called *reversal potential*, and under normal conditions this is about -70 mV. If the membrane potential drops below or becomes higher than reversal potential, ions will flow into or out of the cell, to reverse the potential back to the equilibrium state [Dayan and Abbott, 2001].

2.2.4 Neuronal firing

Neuronal activity is marked by the firing of neurons. Generally, in a neuron the arrival of inputs from other neurons will result in an excess net inward ion flow. Because of this, a positive charge will build up inside the cell, increasing the neuron potential. Depending on the previous and current inputs, this charge build-up may increase or may dissipate over time. This charge build-up will eventually open more ion channels, which will in turn increase the inward ion flow in a cyclic manner. At the point when the membrane potential reaches a *threshold* value, an action potential will occur causing the neuron to fire due to the huge amount of positive charge inside the membrane. An action potential is roughly a 100 mV fluctuation in the electrical potential across the cell membrane with a time duration of about 1 to 2 msec. Although the action potentials can vary in duration, amplitude and shape, they are generally treated as identical stereotyped events in neural encoding studies. After the firing event, the membrane potential

will reach a very low value. Due to this low potential and open ion channels, a neuron cannot fire again within a period following a previous firing activity. This time period is known as the *refractory period.* Eventually, the neuron will return back to its resting state and will be ready again for another episode of firing activity [Shepherd and Koch, 1990; Dayan and Abbott, 2001].

2.3 Neuronal coding

Neuronal coding is one of the fundamental issues in neuroscience. It is said that in every small volume of the cortex of a mammalian brain, thousands of spikes are being generated in each millisecond. Each neuron emits spikes continuously as a spike train. Generally it is accepted that the information is coded by means of these spikes [Dayan and Abbott, 2001]. How the information is coded is still an ongoing debate. What information is transmitted and is it possible for an external observer to decode that information, are the other two fundamental issues of interest [Gerstner, 2001]. Over the recent years, several coding schemes have been proposed and analysed. However, three coding schemes, namely, *rate coding*, *temporal coding* and *population coding* are most widely used in practice. Another important issue here is to understand whether the individual action potentials and individual neurons encode independently or the correlations between different spikes from the same or several neurons carry significant information. In a simpler form, the rate coding and temporal coding schemes deal with a single neuron. However, the population coding is based on the spikes from a population of neurons. This scheme considers the correlations among spikes from a single neuron as well as from other neurons [Gerstner and Kistler, 2002].

2.3.1 Rate coding

The most commonly used coding scheme is the rate coding method, where the information is transferred by using the mean firing rate of a neuron. However, there is no clear definition for mean firing rate in the literature. In fact, there are at least three modes of rate coding methods used in practice. Firing rate as a temporal average is the most general notion, which is more applicable when the stimulus is constant or slowly varying. The second rate coding concept is the average as a spike density. In this mode, the average is found over several observations of an event. This method is more suitable for the evaluation of neuronal activity, particularly where the stimulus varies over time. Although it is acceptable as an experimental procedure, applicability of this notion is questionable in practical situations. The third form is the rate as population activity, where the average is taken over a population of neurons instead of using a single neuron. Throughout the past several decades there was a wide belief that the information is coded in the neural systems through the mean firing rate of a neuron. Since the measurement of spike rate is relatively easy and straight forward, this coding scheme was used as a standard tool for describing the properties of biological neurons [Gerstner and Kistler, 2002]. Furthermore, the development of the artificial neural networks was also based on this idea [Zador, 2000], where the analog value generated by an artificial neuron as its output represents the firing rate of a neuron [Maass, 1997a].

2.3.2 Temporal coding (Spike coding)

Unlike rate coding, the temporal coding scheme, also known as spike coding, utilises the timing of individual spikes. Here, information is considered to be coded through the timing of each spike generated by a neuron. This scheme

utilises the exact firing times of neurons, which has the potential to convey a huge amount of information [Thorpe et al., 2001; Panchev and Wermter, 2001]. There are several variations of this coding notion. A straight forward scheme is *time-to-first-spike*, where the information is coded through the timing of the first spike for a particular stimulation. In this case, the spikes following the first one are considered as irrelevant and ignored. The idea behind this scheme is that the time difference between the first spike and a reference signal is used for coding the information. The reference signal could be local to the neuron or group of neurons concerned which specifies the commencement of the stimulation. A highly stimulated neuron will tend to fire rapidly while a less stimulated neuron will generate spikes more slowly. Hence a neuron which fires closer to the reference signal would indicate a strong stimulation while a later one could represent a weak stimulation [Gerstner and Kistler, 2002]. This idea of the timing of the first spike contains much of the information conveyed through a spike train, is reinforced by a constraint faced in neural events. It is considered that, due to time constraints, it is unlikely for the brain to be able to evaluate all the spikes but only the first from each neuron per processing step [Thorpe et al., 1996]. Due to the simplicity of this coding scheme, this is deployed in several analytical studies [Maass and Schmitt, 1999; Maass, 2001a] as well as in this research.

The reference signal considered in the coding model discussed here is local to the neuron concerned. Alternatively, global periodic signals, which are common in some areas (Hippocampus, Olfactory) of the brain can be deployed for reference. *Phase coding* is a variant of the temporal coding scheme which makes use of the above mentioned global reference signal. Spikes from some other neurons can also be used as the reference signal for spike codes, which take into consideration the correlation and synchrony of neurons. This can be also viewed as a population coding scheme, which is discussed in the next section.

2.3.3 Population coding

Population coding views the problem of neuronal coding in a different way as compared with the other two schemes mentioned above. The fundamental issue here is whether the individual spike events and individual neurons encode the information independently of each other or does the correlation between different spikes and different neurons carry significant amounts of information? Population coding takes these correlations into account. Here it is not important whether the information is carried through spike rates or spike times [Dayan and Abbott, 2001]. For example, a form of population coding views the coding mechanism as a rate of population activity. Here, instead of the average spike count from a single neuron, the average is found from spikes generated by a group of neurons. The population coding scheme can also utilise the timing of individual spikes. For example, a reference spike from a neuron can be used to code the information from some other neurons based on the temporal property of their spikes [Gerstner and Kistler, 2002].

2.3.4 On the coding of neural information

The nature of the neuronal code is a topic of intense debate within the neuro science community. The rate coding scheme has been successfully applied in various models and studied over the past years due to the relative ease of measuring the spike rate experimentally. However, in recent years the validity of rate coding has been severely questioned [Gerstner and Kistler, 2002]. This coding scheme is thought to be much too simplistic to perform complex tasks. Not only does the temporal averaging of the spike train lose valuable information on the individual spike, but also the process of finding the average firing rate is relatively slow compared to the speed of neural events. It was found in several studies that the

time window available for a neuron during a neuronal event is too small to find the rate of spikes [Thorpe and Imbert, 1989]. In other words, in neural systems where very rapid processing is required, the neurons participating at each level of the processing hierarchy will not have enough time to emit or wait for more than one spike [Thorpe et al., 2001]. In reality, information is likely to be coded both by the rate of spike and the spike timing of single neurons. But the question here is whether the rate coding can be used for fast and complex computation. Experimental evidence is mounting in favour of temporal coding to establish it as the most plausible one in most cases [Thorpe, 1990].

In neural systems, information is likely to be carried by both individual spikes and through correlation. It is evident that the correlation can carry additional information but it was found that this information is rarely larger than 10% of the information carried by independent spikes. Since independent spike codes are much simpler to implement and analyse than population codes, most work on neural coding favour spike independence [Dayan and Abbott, 2001]. For these reasons, temporal coding has been widely deployed in recent research studies on spike-based neural networks.

2.4 Neuron models

Neuron models are well detailed mathematical models used to describe the behaviour of neurons. Based on the knowledge of biophysical mechanisms responsible for generating neuronal activity, several neuron models have been proposed. These models vary in their level of abstraction, descriptive detail and complexity. Two types of models, namely, *conductance-based* models and *spiking neuron* models are described here. The conductance based models are described here to give a better understanding of a neuron's physical behaviour. The spiking neuron

models are the basis for the spiking neural networks which describe the state of a neuron at a higher level of abstraction [Gerstner and Kistler, 2002].

2.4.1 Conductance-based neuron models

A conductance-based neuron model describes the behaviour of a neuron in terms of the ion flow through the membrane of a neuron. Hodgkin-Huxley model is a suitable example because of its simple approach and also for its popularity as a base model for most of the conductance-based models [Hodgkin and Huxley, 1952]. In order to investigate the flow of electric current through the neuron membrane, Hodgkin and Huxley performed a series of experiments [Hodgkin and Huxley, 1952] on the surface membrane of a squid's giant nerve fibre. Based on the experimental findings, a mathematical description of the behaviour of the membrane was developed. The Hodgkin-Huxley model can be explained with an equivalent circuit shown in Figure 2.6. In this model, three different types of ion channels are accounted for the flow of ions, namely, sodium channel, potassium channel and leakage channel. In figure 2.6, the sodium channel is denoted with subscript Na, the potassium channel with subscript K and an unspecified leakage channel with subscript L [Gerstner and Kistler, 2002]. This circuit describes each channel with a conductor and a battery. The conductor in figure 2.6 specifies the conductance of the channel and the battery specifies the potential induced by the concentration of ions. As the result of an injected external current $I(t)$, charges inside the membrane will increase, while a part of it leaks out through the ion channels. Similarly, in this circuit when a current $I(t)$ is injected, a part will be stored in the capacitor C_m and the remainder will leak through the conductors. The Hodgkin-Huxley model may be described with a set of four differential equations. For a detailed description of the Hodgkin-Huxley model, refer [Gerstner and Kistler, 2002]. There are a number of other models varying

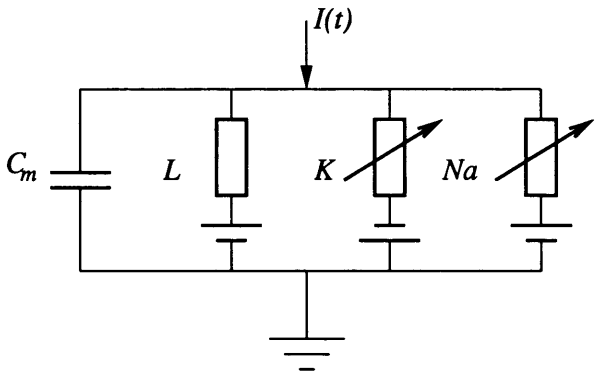


Figure 2.6. An equivalent circuit for the Hodgkin-Huxley model. Redrawn from [Gerstner and Kistler, 2002].

in complexity and descriptive detail, but they are not discussed here since they are outside the scope of this study.

2.4.2 Spiking neuron models

Conductance-based models can reproduce the electro-physiological measurements more accurately, but they are difficult to analyse because of their complex nature. Therefore, *integrate-and-fire models* were introduced, which are simplified models and considered as formal spiking neuron models. These models do not show the complex details explicitly and are very popular for studies of neural coding, memory and network dynamics. Here, the net current through all the ion channels is considered as a single leakage current. A popular example of the formal spiking neuron model is *leaky-integrate-and-fire model.* Generalisation of the leaky-integrate-and-fire model led to the introduction of the *Spike Response Model* (SRM). This model was generalised in such a way as to modify it to be time dependent, whereas the integrate-and-fire model depends on voltage [Gerstner and Kistler, 2002]. The following section introduces the SRM model in detail.

2.4.3 Spike response model

The spike response model (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 lap taken from the last firing event. The SRM model is the basis for the spiking neural network introduced later in this chapter. The model describes the state of a neuron j at time t by the state variable $u_j(t)$ [Maass, 2001a]. Let $\mathcal{F}_i$ be the inputs the neuron j receives from

pre-synaptic neurons $i \in \Gamma_j$, where $\Gamma_j = \{i \mid i \text{ pre-synaptic } to\ j\}$. 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 $w_{ji}\ \epsilon_{ji}(t - t_i^f)$, where ϵ_{ji} is called the *spike response function.* The input spikes can either increase or decrease the state variable u_j. If u_j exceeds a threshold value θ at some time t, then an output spike will be generated. At a particular time t, let all the previous output spikes of neuron j be $\mathcal{F}_j$ where $\mathcal{F}_j = \{t_j^f; 1 \leq f \leq n\} = \{t \mid u_j(t) = \theta \text{ and } u_j'(t) > 0\}$. Here the t_j^f is the time when the state variable u_j crosses the threshold value θ from below. Immediately after a particular firing event the neuron potential will be set to a very low value and will return to normal state after a significant amount of time in order to realise the refractory phenomenon of a biological neuron. The state variable $u_j(t)$ of a neuron j at time t can be defined by equation 2.4.

$$u_j(t) = \sum_{t_j^f \in \mathcal{F}_j} \zeta_j(t - t_j^f) + \sum_{i \in \Gamma_j} \sum_{t_i^f \in \mathcal{F}_i} w_{ji}\ \epsilon_{ji}(t - t_i^f) \tag{2.4}$$

where ζ_j is the function to reflect the refractoriness of the neuron j, the term w_{ji} is the strength of the connection between the neurons i and j, and ϵ_{ji} is the spike response function which can be either excitatory or inhibitory. Equations 2.5 and 2.6 specify typical examples of spike response functions which are described in figure 2.7 [Gerstner, 2001]. A more complex spike response function is given in equation 2.7 [Gerstner and Kistler, 2002]. Excitatory synapses will increase the potential of the receiving neuron. For an inhibitory synapse, the effect will be in negative direction which will decrease the state of the receiving neuron.

$$\epsilon(t) = \frac{t}{\tau_e}\ e^{1-\frac{t}{\tau_e}} \tag{2.5}$$

$$\epsilon(t) = -\frac{t}{\tau_i}\ e^{1-\frac{t}{\tau_i}} \tag{2.6}$$

$$\epsilon(t) = \frac{e^{\frac{-t}{\tau_b}}}{1 - \frac{\tau_b}{\tau_a}} [e^{\frac{-t}{\tau_a}} - e^{\frac{-t}{\tau_b}}] \tag{2.7}$$

where $\tau_e(>0)$ and $\tau_i(>0)$ are time constants for excitatory and inhibitory spike response functions respectively. $\tau_a(>0)$ and $\tau_b(>0)$ are two other time constants.

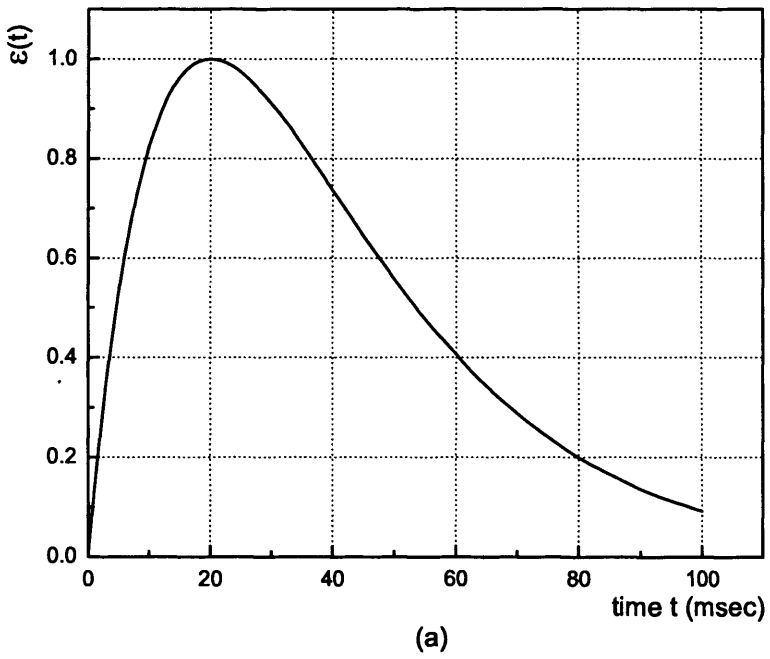
Although the threshold θ is specified as a fixed value, it can be viewed as a time varying function. This is to reflect the refractory behaviour of a biological neuron immediately after a spike event. In the spike response model there could be an additional effect due to an external current I^{ext} $(=\int_0^\infty k(t-\tilde{t}_i\,,\,s)\,I^{ext}(t-s)\,ds)$, which can be ignored if all the inputs to the neuron are only by means of spikes [Gerstner and Kistler, 2002].

2.5 The spiking neural network

Spiking neurons differ from conventional neurons such as McCulloch-Pitts or sigmoidal neurons in their characteristics. Even though spiking neuron models were introduced roughly at the same time as McCulloch-Pitts neurons, due to their complexity they were not widely used for applications in artificial neural networks. The introduction of SRM neurons paved the way for more research on spiking neural network models. As a result, a recent spiking neural network model introduced in [Maass, 1996] became more popular leading to further analytical studies [Maass, 1997a] and a significant increase in research on learning in spiking neural networks [Gerstner et al., 1996; Ruf and Schmitt, 1997; Bohte et al., 2002b,a]. The following sub-sections present the network model introduced by Maass in [Maass, 1996].

2.5.1 Definition of the spiking neural network

Maass [Maass, 1996] proposed a spiking neural network model which is biologically realistic and powerful yet easier to implement and analyse. This model is based on SRM neurons and incorporates the timing of the action potentials for



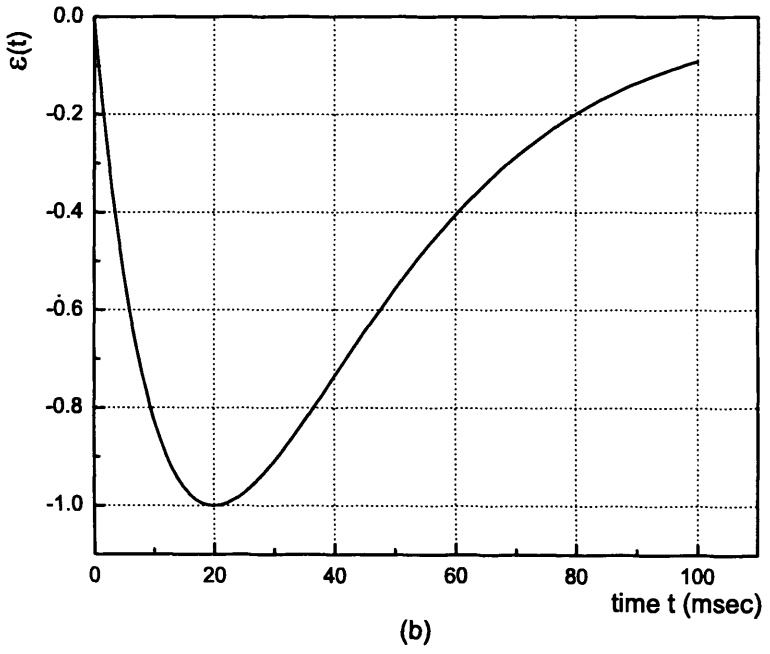


Figure 2.7. (a) Excitatory and (b) inhibitory spike response functions. Redrawn from [Gerstner, 2001].

computation and communication. The spiking neural network N can be defined as follows:

- a finite directed graph $\langle V, E \rangle$, where the elements of V are called as neurons and elements of E as synapses or connections
- a subset $V_{in} \subseteq V$ of input neurons
- a subset $V_{out} \subseteq V$ of output neurons
- a threshold function $\theta_j : \mathbf{R}^+ \rightarrow \mathbf{R} \cup \infty$ for each output neuron $j \in V_{out}$
- a spike response function $\epsilon_{ji} : \mathbf{R}^+ \rightarrow \mathbf{R}$ for each synapse $\langle i, j \rangle \in E$
- a weight function $w_{ji} : \mathbf{R}^+ \rightarrow \mathbf{R}$ for each synapse $\langle i, j \rangle \in E$

Assume firing of the input neurons $i \in V_{in}$ is independent of the SNN. Let the input for the network N is defined as a set $\mathcal{F}_i \subseteq \mathbf{R}^+$ of firing times for the input neurons $i \in V_{in}$. Assume $\mathcal{F}_j \subseteq \mathbf{R}^+$ is the set of potential firing times of the output neurons $j \in V_{out}$. The equation 2.8 specifies the potential $u_j(t)$ of the output neuron j at some time t.

$$u_j(t) = \sum_{i \in V_{in}} \sum_{s \in \mathcal{F}_i : s < t} w_{ji}(s)\, \epsilon_{ji}(t - s) \tag{2.8}$$

The set of firing times for an output neuron $j \in V_{out}$ is defined recursively. The first firing of neuron j occurs at time t when the potential $u_j(t) \geq \theta_0$ where θ_0 is the initial threshold value. The consequent firing events will be at t when $u_j(t) \geq \theta(t - s)$. Here $t > s$ and s is the time of the most previous firing event of the output neuron j. The firing times $\mathcal{F}_j$ of the output neurons $j \in V_{out}$ is the output of network N.

A number of researchers have carried out significant work on spiking neural networks based on the above described model. Several modifications and

assumptions were proposed in previous research to simplify the implementation of the above described model without losing its computational power. A major modification comes through the selection of the temporal coding scheme. If the information is coded through the timing of the first spike, it is sufficient for a neuron to fire once during a cycle of operation in order to convey the information. Hence, the refractory term can be removed [Ruf and Schmitt, 1997]. In this case, the threshold can be kept constant throughout the activity of the network [Ruf and Schmitt, 1998; Bohte et al., 2002a,b]. Another improvement is the inclusion of a delay d_{ji} for each synapse in addition to the strength [Maass, 1997b]. When a spike is passed through a connection, it will be delayed from reaching the target neuron by a finite time, specified by the delay value of that particular connection.

2.5.2 Computing with spiking neural network

A number of analytical studies were performed by other researchers on spiking neural networks in order to investigate their capability. The objective of some of these studies is to explore how they can be deployed to compute values similar to the output of conventional neural networks. They are also used to compare the computing capability of the spiking neural networks with other network models. These issues are discussed below in detail.

2.5.2.1 Realising a perceptron

A perceptron is the simplest computational unit of the conventional neural network. It is a threshold gate which generates an output of 1 if the sum of the weighted inputs exceeds a threshold value and otherwise an output of 0. A spiking neuron can act as a threshold gate for binary inputs. Here, the inputs to the neuron could be either an input spike at some time T_{in} or no spike at all. By keeping the spike response function $\epsilon_{ji}(t)$ identical, the effect of each input spike

can be considered as a constant ϵ. Hence, if the weighted sum of the active inputs $\sum_i w_{ji}\ \epsilon$ exceeds the threshold value θ, then the neuron j will generate an output spike. In this way, a single layer network of threshold gates can be realised with spiking neurons [Maass, 2001a].

2.5.2.2 Computing a weighted sum

A spiking neural network can produce an output of temporal patterns when presented with an analog vector encoded temporally. The output of a spiking neuron is the time of the generation of a spike. For computation in multilayer or recurrent networks, a mechanism is necessary in order to realise the output of a spiking neuron as an analog value encoded in the same way as the input. This mechanism can be viewed as a shifting of the firing time t_j of a spiking neuron j, depending on the input patterns t_i presented and the connection parameters w_{ji} and d_{ji}. A mechanism which computes the weighted sum $\sum_i \alpha_{ji}\ x_i$ through the firing time of neuron j is explained below. Here $\alpha_{ji} \in \mathbf{R}$ are some arbitrary parameters and $x_i \in [0, 1]$ are the inputs to the network for all i pre-synaptic to neuron j [Maass, 2001a].

Let the input pattern be temporally coded as $t_i = T_{in} - x_i$, with T_{in} as an independent reference signal. Assume that the spike response function $\epsilon_{ji}(t)$ increases linearly until its peak value with a gradient of $\lambda_{ji} \in \mathbf{R}$ and decreases linearly after crossing the peak value. With the selection of suitable parameter values for the spiking neuron model it is possible for the neuron to fire at some time when all the inputs to the neuron are in their linear increasing segment. For a spiking neuron j the potential at time t is defined by the equation 2.9:

$$u_j(t) = \sum_{i \in \Gamma_j} w_{ji}\ \epsilon_{ji}(t - t_i - d_{ji}) \tag{2.9}$$

where Γ_j is the set of neurons pre-synaptic to neuron j; w_{ji} are the connection

weights and d_{ji} are the connection delays for all $i \in \Gamma_j$. Let the neuron j fire at time t_j. At the time of firing the potential of neuron j at time t should be equal to the threshold, i.e., $u_j(t_j) = \theta$,

hence

$$\sum_{i \in \Gamma_j} w_{ji}\ \epsilon_{ji}(t_j - t_i - d_{ji}) \ = \ \theta \tag{2.10}$$

Since the spike response function is assumed to be linear with gradient λ_{ji},

$$\sum_{i \in \Gamma_j} w_{ji}\ \lambda_{ji}\ (t_j - t_i - d_{ji}) = \theta \tag{2.11}$$

hence

$$t_j = \frac{\theta}{\sum_{i \in \Gamma_j} w_{ji}\ \lambda_{ji}} + \sum_{i \in \Gamma_j} \frac{w_{ji}\ \lambda_{ji}\ (t_i + d_{ji})}{\sum_{i \in \Gamma_j} w_{ji}\ \lambda_{ji}} \tag{2.12}$$

let $\lambda = \sum_{i \in \Gamma_j} w_{ji}\ \lambda_{ji}$, and $\alpha_{ji} = \frac{w_{ji}\ \lambda_{ji}}{\lambda}$

$$\begin{aligned} thus, \quad t_j &= \frac{\theta}{\lambda} + \sum_{i \in \Gamma_j} \frac{w_{ji}\ \lambda_{ji}}{\lambda}\ (t_i + d_{ji}) \\ t_j &= \frac{\theta}{\lambda} + \sum_{i \in \Gamma_j} \alpha_{ji}(T_{in} - x_i + d_{ji}) \\ t_j &= T_{out} - \sum_{i \in \Gamma_j} \alpha_{ji}\ x_i \end{aligned}$$

where T_{out} is an independent constant defined as,

$$T_{out} = \frac{\theta}{\lambda} + \sum_{i \in \Gamma_j} \alpha_{ji}\ (T_{in} + d_{ji})$$

Since the parameter T_{out} is independent from the inputs, the firing time t_j represents an arbitrary weighted sum $\sum_{i \in \Gamma_j} \alpha_{ji}\ x_i$.

2.5.2.3 Coincidence detection

An important feature of spiking neuron is that it can act as a coincidence detector for the incoming pulses [Abeles, 1982; Maass, 2001a]. Spiking neuron can detect

the coincidence of the input signals with great ease, unlike the conventional neural networks, where it is computationally much expensive. A spiking neuron can be tuned to fire only when it receives inputs simultaneously, and the coincidence of the input signals can only then be detected. If the arrival time of the incoming spikes encodes analog numbers x_i, then a spiking neuron can detect whether some of these numbers have almost equal value. A neuron fires only when its potential due to the incoming spikes exceeds the threshold value. Consider a simple scenario where a spiking neuron needs at least two nearly simultaneous input spikes. In order to obtain a potential equal to or higher than the threshold, at least two inputs should reach the neuron within some acceptable time window. This is because the effect of an input spike will dissipate with time after reaching a peak value. The combined effect of the input spikes must reach the threshold in order to enable the neuron to fire. Figure 2.8 describes this typical scenario graphically. The example specified above handles the coincidence detection of two spikes, but a spiking neuron can also be deployed to detect more complex temporal patterns with more than two spikes [Maass, 1997b].

The functionality of a spiking neuron as a coincidence detector in the temporal domain is similar to a Radial Basis Function (RBF) [Haykin, 1999] unit in the continuous domain [Maass, 2001a]. The RBF unit is an artificial neural network paradigm which is based on the techniques for interpolation in multidimensional space with a set of functions called as *Radial basis functions*. Radial basis functions are generally a set of Gaussian functions which can compute the difference between an input vector and a specified vector, using an Euclidean norm. A typical RBF function for a single input x is given in equation 2.13 [Orr, 1996]:

$$h(x) = e^{-(x-c)^2/r^2} \tag{2.13}$$

where c is called the centre of the function and r is the radius. In spiking neural networks, the temporal delays play the role of the centre of an RBF neuron. When

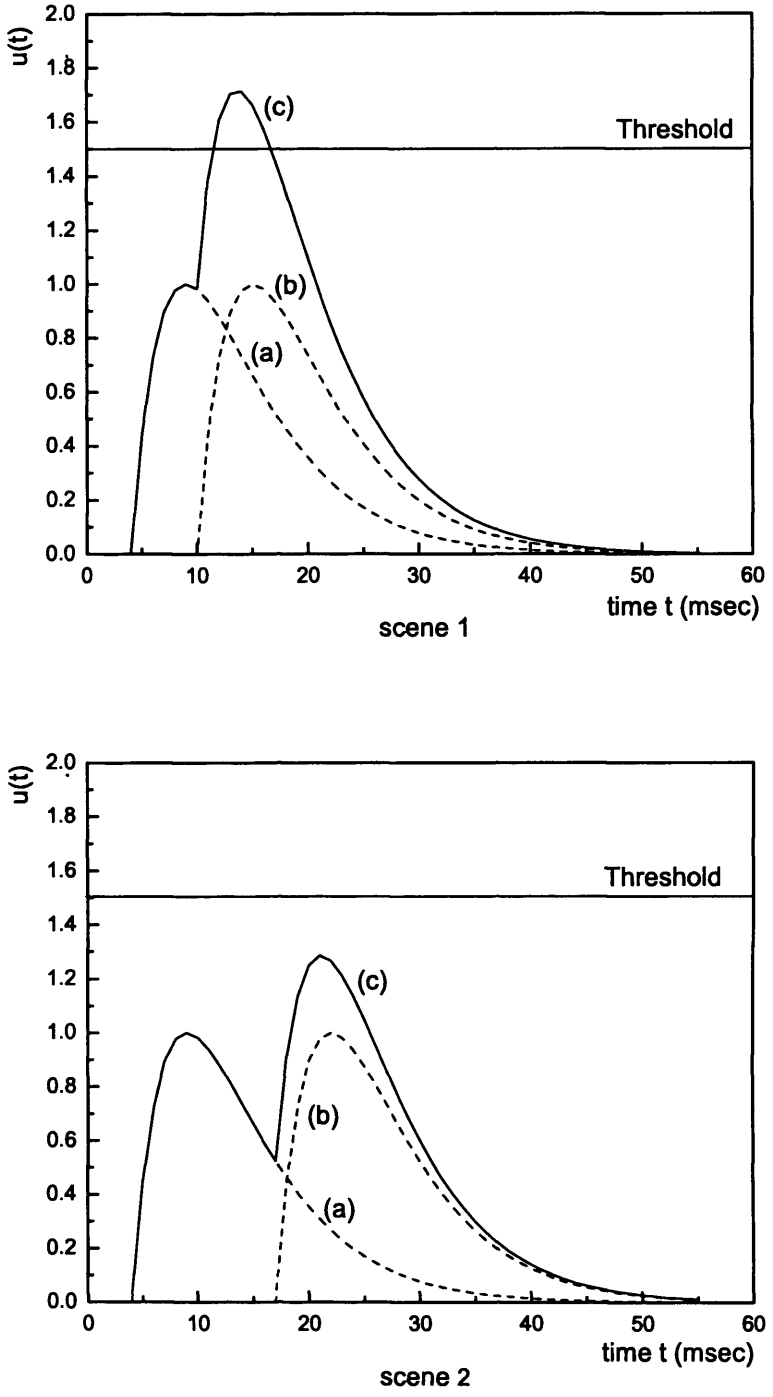


Figure 2.8. Coincidence detection by a spiking neuron. In the figure (a) and (b) show the effect of two input spikes, (c) shows the change in total potential. In scene 1, since the two spikes are close, the potential exceeds the threshold therefore an output spike can be generated, but not in scene 2.

correctly tuned, this delay pattern will compensate for the differences of the input pattern and enable all the inputs to coincide at the target neuron [Hopfield, 1995].

2.5.3 Spiking neural network simulators

In order to analyse the functionality of an SNN, it must be simulated by some means. A number of dedicated digital and analog pieces of hardware are available for this purpose [Ienne, 1997]. However, software is more popular for simulating and analysing neural networks due to its high degree of flexibility. Several software packages and libraries are available for this purpose. GENESIS (for *GEneral NEural SImulation System*) is well known, which has been developed at California Institute of Technology (CALTECH). In essence, GENESIS is a research tool to provide a standard and flexible means of constructing realistic simulations of biological neural systems. This was specifically designed to allow the construction of biological simulations at many different levels from sub-cellular components to whole cells and from single cells to networks of cells [Bower, 1991]. Another spiking neural network simulator in objective C for the MacOS platform is available which was developed in the department of Computing Science and Mathematics, University of Stirling, Scotland [Smith, 2004].

In addition, there are libraries available for the simulation of spiking neural networks. *SpikeNNS* is one of them which is capable of simulating multi-layer architectures with biologically inspired topologies and spike-timing dependent learning [Marian, 2001]. This is distributed as a library of C functions and requires *Stuttgart Neural Network Simulator* (SNNS) as the base program. SNNS is a software simulator, currently available for UNIX and Windows platforms, developed in 1990 at the Institute for Parallel and Distributed High Performance Systems (IPVR) at the University of Stuttgart [Zell et al., 1992]. Currently, SpikeNNS is available as a patch file which can be added to the SNNS. Another

spiking neural network library is *Amygdala* written in C++ for Linux platform. It includes several neuron models, Symmetric Multi-Processing (SMP) support and facilities for developing SNNs with genetic algorithms [Koch and Grover, 2002].

Although most of the systems available are capable of modelling and simulating huge networks with complex neurons and connections with realistic biological neural functions, their applicability is limited for real-world problems. An objective of this research is to develop a spiking neural network model for clustering and classification tasks. Hence, software has been developed as part of this research which gave great flexibility for all the types of experiment to be performed. This software realises the SNN model as a discrete model on a digital computer. The following section describes this in detail.

2.5.4 Discrete spiking neural network model

The functioning of a spiking neuron is time related. To compute with an SNN, it has to be activated over a period of time. During this time, the inputs are given at specified times as spikes. Output spikes are generated depending on the timing of the inputs, network connections and the parameters of the neuron. The output of the network is the timing of these spikes. The difference between biological systems and digital systems is that the signals in the former are continuous and in the latter are discrete. Hence, a suitable model is necessary to realise the continuous functionality of an SNN on a digital computer. In order to simulate an SNN, the continuous time window is divided into a number of discrete time intervals. Each time interval is considered as a step in the computation. For each step the state of the network is updated [Jahnke et al., 2001]. The input to the network is realised through the timing of input spikes. Hence, the continuous analog input values are coded using small temporal differences [Bohte et al., 2002b]. A high input value is represented by an early input spike and a low value

by a late spike with reference to an input time window [Gerstner and Kistler, 2002]. Hence an input value $x_i \in [0, 1]$ to the neuron i is represented by a spike at step n_i, which can be computed by the equation 2.14.

$$n_i = \lfloor \frac{(1 - x_i)\ t_{input\_window}}{dt} \rfloor \tag{2.14}$$

where t_{input_window} is part of the activity window starting from t_0 where all the input spike events occur. dt is the time interval between two consecutive steps. The firing time t_i of a spike at step n_i is an approximation of the time between $t_0 + (n_i - 1)\ dt$ and $t_0 + n_i\ dt$.

Let T be the duration of the activation time window and t_0 be the start of the activity. If the whole process expands over some adequate number of steps with each step having an equal time slice of dt, then the state of the neuron j at step n can be given by the equation 2.15:

$$u_j(n) = \sum_{i:1..m, s_i<n} w_{ji}\ \epsilon((n - s_i)\ dt) \tag{2.15}$$

where s_i is the step count where the input for neuron i is presented; m is the total number of input neurons. Here $\epsilon(t)$ is the spike response function which is 0 for $t \leq 0$ and reaches 0 when $t \rightarrow \infty$. If the simulation is carried out as a separate activity for each set of stimulation then t_0 can be set to zero. In addition, by selecting dt as a unit interval the equations 2.14 and 2.15 can be simplified. In this case the number of time steps will be equal to the duration of the activation time window. The precision of the network can be increased by decreasing the time slice dt. This equates to increasing the activation time steps. The duration of the activation window has to be chosen to be large enough to accommodate the full activity of the neuron.

2.5.5 Learning in spiking neural networks

A number of researchers have carried out significant work on learning in spiking neural networks. This section summarises their research, which includes research on the modelling of biological neural systems with spike-time-based networks as well as work on learning with SNN for several applications.

Pioneering work on learning in neural systems was performed by Hopfield [Hopfield, 1995] and a computational model for pattern recognition computation was proposed. An important argument put forward by Hopfield is that the choice of information representation in a system should be based on its ease of use for computing useful results rather than for their efficient transmission properties. Hence, the usage of the timing of action potentials to represent the information in the network instead of the very popular analog value representing the rate of firing, was introduced. This approach later inspired much of the research on spiking neural networks. The Hopfield model computes by combining information through pathways of different delays, where a computational unit behaves more like a radial basis function unit rather than an elementary sigmoid unit [Maass, 1997a]. Gerstner et al. [Gerstner et al., 1996] performed a modelling study of the barn owl's auditory system and introduced a mechanism for learning in neural systems. The network considered is comprised of a broad distribution of delayed interconnections and trained with an unsupervised Hebbian learning rule. The learning rule is based on the firing times of the pre- and post- synaptic neurons. If a post-synaptic spike precedes the pre-synaptic spike by a specified time, then that connection is strengthened, otherwise the connection is weakened. The learning selects matching delays and enhances it while weakening the others. This mechanism is considered to be an implementation of the model proposed in [Hopfield, 1995] [Ruf and Schmitt, 1998]. Some other significant modelling studies

have been reported in [Blum and Abbott, 1996; Gerstner et al., 1996; Eurich et al., 1999; Kistler and van Hemmen, 2000; Song et al., 2000; van Rossum et al., 2000]. A modelling study for the activity-dependent development of directional selective cells in the primary visual cortex using self-organising delay adaptation maps can be found in [Tversky and Miikkulainen, 2002].

Ruf and Schmitt [Ruf and Schmitt, 1998] realised self-organisation in a network of spiking neurons using action potential timings. In their work, an unsupervised learning mechanism was proposed to train a network of spiking neurons with temporally coded inputs. Based on the firing times, a winner neuron is determined among the competing neurons and its connections' strength is increased. Usage of the spike times enabled the model to identify the winner neuron quicker with locally available information. Self-organisation is realised by using lateral excitation and inhibition. The neurons which are topologically closer to the winner neuron are strengthened while neurons which are away from the winner neuron are weakened. It was shown that the training mechanism creates a topology-preserving map similar to the Kohonen's self-organising map.

Based on the work presented in [Hopfield, 1995; Gerstner et al., 1996], further research was performed by Natschläger and Ruf [Natschläger and Ruf, 1998] on the spatial and temporal pattern analysis capabilities of spiking neurons. This was an extension of the work of previous research by considering the firing times of the neurons in addition to the information about whether a neuron is active or not. Using artificially constructed data, it was found that the neurons converged to the centres of clusters even in the presence of noise. The model was also able to reconfigure the clusters if new classes of data were added or removed during the learning process. This model claimed to be a feature-extraction technique, since the learning algorithm was able to find proper delays for the coordinates of the sub-space and deactivate the remaining inputs.

Bohte et al. [Bohte et al., 2002b] further investigated unsupervised clustering using spiking neurons. The model presented in [Natschläger and Ruf, 1998] failed to cluster more realistic data due to the low precision of the input encoding which in turn limited the clustering capability. It was found that this model had a lack of clustering capacity since it failed to detect small clusters. To improve the encoding precision and clustering capability, a coding scheme was proposed in [Bohte et al., 2002b], which uses a population of neurons to encode the input. It was shown that this new coding strategy enhanced the model's capability in clustering data with high precision. However, due to the increased number of input neurons and the multiple sub-connections, the network size is greatly increased.

Ruf and Schmitt [Ruf and Schmitt, 1997] proposed a supervised learning mechanism for spiking neural networks with temporally coded inputs. The proposed method uses a Hebbian-based learning rule which utilises the single spiking events of pre- and post- synaptic neurons. Here the connection strengths are updated with a value which is proportional to the difference between the output spike time and a reference spike time. It was shown that the training brings the connection weight values closer to a value which represents the difference between the pre- and post-synaptic firing times.

Error back-propagation learning is a popular and widely applied classification technique. Bohte et al. [Bohte et al., 2002a] achieved this type of learning in a network of spiking neurons with temporally encoded inputs. A special feed-forward network structure was proposed which can have one or more hidden layers in addition to the input and output layers. Here a single connection between two neurons in adjacent layers is comprised of several sub-connections with different delays. In order to achieve high temporal resolution for the input patterns, each input value is distributed over multiple input neurons using a population cod-

ing scheme. Each input dimension is coded using an array of one dimensional receptive field as in [Bohte et al., 2002b]. Learning rules were derived for updating the connection weights of neurons in the hidden layer and output layer. The derivation of the rules is analogous to the derivation of the error back-propagation learning rules. During the training, the interconnection weights are updated using the learning rules in such a way as to force the output neurons to fire at a pre-specified time. The model was applied to classify data sets and achieved accuracy comparable to a sigmoidal network with error back-propagation learning.

Two improvements were suggested by Jianguo and Embrechts [Jianguo and Embrechts, 2001] for the supervised learning model proposed in [Bohte et al., 2002a]. Here a momentum term was added to the learning rule with the aim of improving the learning process. However, it was found that this momentum term degrades the results instead of enhancing them. The other improvement suggested was an adaptive learning rate parameter instead of a fixed one. The learning rate parameter is increased if the error correction is positive and decreased if it is negative. It was found that this modification enables the model to learn much faster in the initial state but has achieved the same highest accuracy achieved by the model with a static learning rate parameter. Another set of improvements were proposed in [Schrauwen and VanCampenhout, 2004]. Here, in addition to modifying the connection weights, rules were proposed to adapt the connection delays, the time constant and the threshold. The derivation of these rules are similar to the one proposed in [Bohte et al., 2002b]. However, the validity of these rules was not verified. Further improvement for the model proposed in [Bohte et al., 2002a] through multiple spikes is reported in [Bohte and Kok, 2005].

The model proposed in [Bohte et al., 2002b] uses an error back-propagation technique to compute the error gradient. In place of this, a stochastic approximation to the gradient was introduced in [Xie and Seung, 2004]. Here a learning

rule was derived, based on a special class of model networks in which neurons fire spike trains with Poisson statistics. There are a few other learning models which were also based on statistical measures [Pfister et al., 2003, 2006]. Another model was proposed in [Carnell and Richardson, 2005], which applied linear algebra apparatus to train spiking neural networks. The practicability of evolutionary techniques was investigated in [Belatreche et al., 2003].

Much of the previous research in the learning of spiking neural networks can be grouped into two categories based on the method of learning, namely, unsupervised learning and supervised learning. Each of these two groups can be further divided into two groups as models which adapt the connection weights and models which adapt connection delays. Since the input information can be encoded either in the connection weights or delays, the models can also be categorised based on their encoding strategy.

Research work reported in [Ruf and Schmitt, 1997; Bohte et al., 2002a; Jianguo and Embrechts, 2001] falls into the category of supervised learning. Here the model proposed in [Ruf and Schmitt, 1997] is a supervised model based on the Hebbian rule which adapts and encodes the input information in the connection weights. All the others are based on the error gradient technique and adapt the connection weights. The models proposed in [Natschläger and Ruf, 1998; Bohte et al., 2002b] come under the category of unsupervised learning and adapt the connection weights. Even though all of these adapt the connection weights, the models attempt to encode the input information in the connection delays.

2.6 Research application

Spiking neural networks have been applied to solve various problems in different domains [Bohte and Kok, 2005]. This research considers the clustering and

classification of data sets as the main application and develops learning models accordingly. This section briefly explains these learning tasks along with the description of the data sets used in the research.

2.6.1 Clustering and classification

Clustering can be considered as an unsupervised learning problem. Like every other problem of this kind, it deals with finding groups in a collection of unlabelled data. The term *clustering* can be loosely defined as the process of organising objects into groups whose members are similar in some way and dissimilar to the objects belonging to other groups. There are a number of popular clustering methods, including k-means and Kohonen's self-organising map.

A classification task is generally a supervised learning problem which can be described as identifying the class of a particular input pattern. A neural network which performs classification goes through a training session in which a group of input patterns are presented along with their class information. After training, the network will be able to identify the class of an unseen pattern. Multilayer perceptron networks with error back-propagation learning, radial-basis function networks and Learning Vector Quantisation (LVQ) networks are some of the well known artificial neural networks for classification tasks [Haykin, 1999].

There are numerous data sets available for training and testing artificial networks. In this research, a set of popular data sets were selected for training, testing and analysing the proposed models. The details on the selected data sets are given below.

2.6.2 Description of the data sets

Three bench marking data sets, namely, *Wisconsin Breast Cancer Database*, *Iris Plants database* and the *Wine recognition data* were selected from the *UCI Machine Learning Repository* [Newman et al., 1998] for the analysis of the proposed models. In addition, a high dimensional data set called *Control chart data* was also selected. These were specifically selected due to their wide usage in the neural network community for bench marking the learning models. All the four data sets contain continuous attribute values. The temporal coding method employed in the research cannot handle non-continuous values. However, if an effective coding scheme is deployed, data sets with nominal values can also be clustered or classified. A description for each of the data set is given below and a summary is provided in table 2.1.

The Wisconsin breast cancer database contains data representing the samples of breast cancer cells in 699 records. Each record belongs to one of 2 possible classes; *benign* or *malignant*. Each record contains 11 attributes, where the first one is an identification number and the last one specifies the class, 2 if it is benign and 4 if it is malignant. The other 9 elements describe the features of the cancer cell. Out of the 699 records, there were 16 records with missing values which were removed from the training set. Out of this, the benign class is represented with 444 records and malignant class with 239 records. This data base will be referred to as Cancer data in this study.

The Iris plants database contains data collected from three classes of plants, namely, *Iris Setosa*, *Iris Versicolour* and *Iris Virginica*. Each class contains 50 plants making a total of 150 records. Each record contains 5 attributes, 4 describing the plant features and the fifth specifying their class. This data base will be referred to as Iris data in this study.

The Wine recognition data set contains data belonging to wines from three different cultivators from a region. The quantities of 13 constituents found in the wine make up the data set in addition to the type identifier of the wine. The data set contains 178 records of the three types of wines which includes 59 records of Type 1 wine, 71 records of type 2 wine and 48 records of type 3 wine. This data will be referred to as Wine data in this study.

A control chart is a tool used in statistical quality control [Pham and Sagiroglu, 2001]. In a practical sense, a control chart is a graphical display of a quality characteristic that has been measured or computed from a sample. Patterns in a control chart can be detected and the information can be utilised to control a process. The control chart patterns are generally grouped into six main categories, namely, *normal, cyclic, downward shift, upward shift, increasing trend* and *decreasing trend* which are shown in figure 2.9. The Control chart data is a collection of artificially created high dimensional data representing the six classes. Equations are available to create data for each of the six patterns [Pham and Sagiroglu, 2001]. The Control chart data used in this study contains 1500 patterns in total, with 250 patterns in each class. Each pattern was generated with 60 samples, thus each pattern contains 60 attributes.

2.7 Research outline

The main objective of this research is to develop alternative learning models for temporal coding spiking neural networks. The spiking neural network considered here is based on the network introduced in [Maass, 1996] (see section 2.5). The temporal coding strategy is selected as the coding scheme and in particular the information is coded by the firing time of the first spike of a neuron. Each activity of the network is considered independent. Therefore, the refractory term

Data set	Number of		
	input parameters	classes	samples
Iris	4	3	150
Cancer	9	2	683
Wine	13	3	178
Control chart	60	6	1500

Table 2.1. Description of the data sets utilised in the research.

is neglected and all the neurons are realised with equal fixed threshold values during a cycle of activity. The term *temporal coding spiking neural network*, also referred to as *spiking neural network*, is used interchangeably in this study. The connections of the network are characterised by positive weight and delay values and realised with identical excitatory spike response function defined by equation 2.5. The spiking neurons are realised either as integrators or as coincidence detectors for different learning models.

This study concentrates on mechanisms which modify each connection only with the locally available information provided by the timing of the spikes produced by the two neurons concerned. Hence, the Hebbian rule, which formulates the above mentioned mechanism, is selected as the basis of the learning strategy on which to build the new learning models. The connection weights and delays are considered to be configurable and adapted in order to encode the input information. The adaptation of these entities is to be performed both through unsupervised and supervised methods. In particular, the research was conducted in three distinct forms, namely, unsupervised weight adaptation, unsupervised delay adaptation and supervised delay adaptation. Work on supervised weight adaptation has already been carried out by other researchers. This type of learning was not, therefore, considered in this study. Clustering task is selected as the application for the proposed unsupervised models and classification for the supervised model.

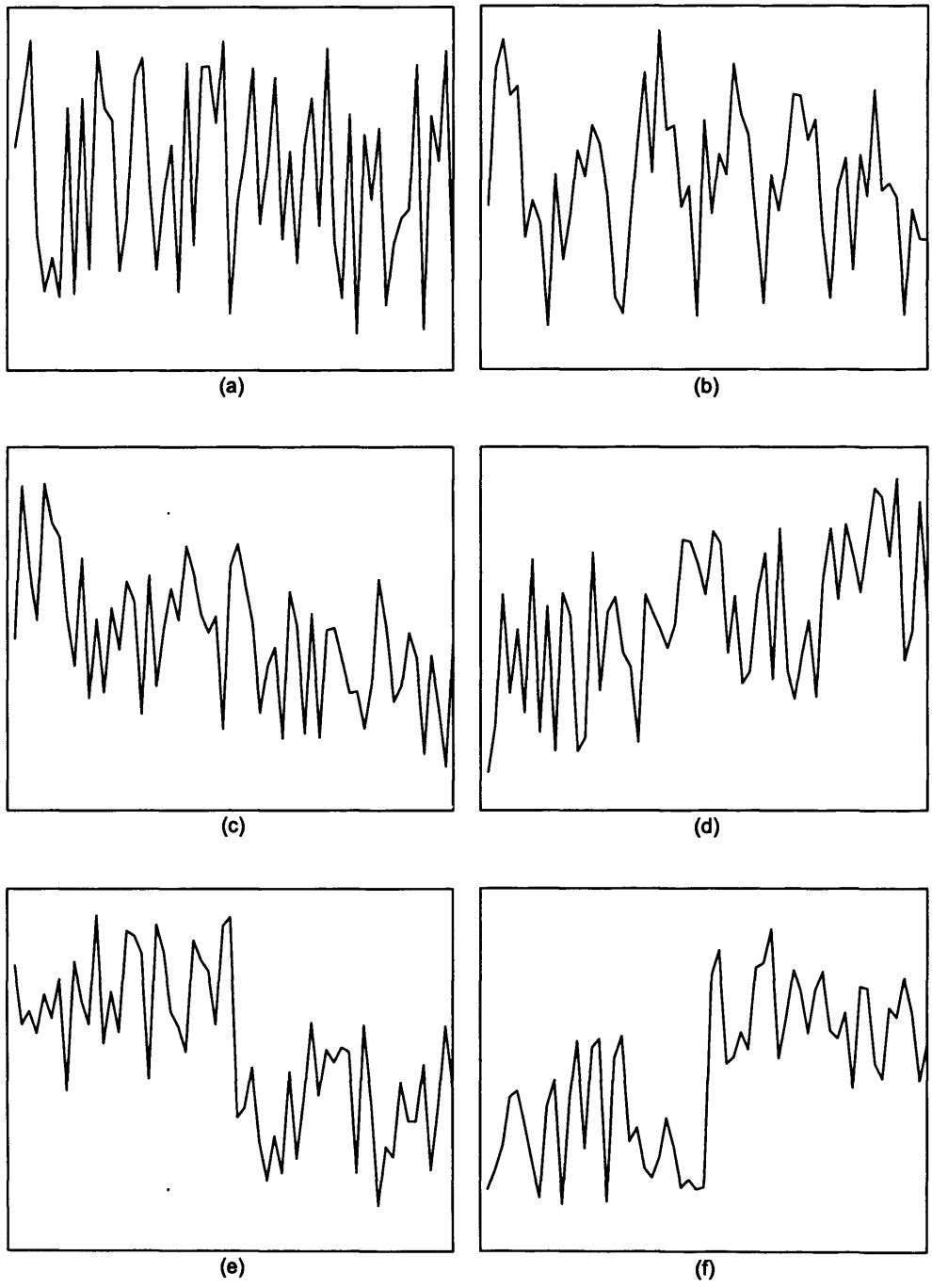


Figure 2.9. The six categories of the control chart patterns, (a) normal pattern, (b) cyclic pattern, (c) decreasing trend, (d) increasing trend, (e) downward shift and (f) upward shift.

Chapter 3

Self-Organising Weight Adaptation Spiking Neural Network (SOWA_SNN)

3.1 Introduction

This chapter investigates unsupervised weight adaptation learning in spiking neural networks and proposes a novel self-organising weight adaptation spiking neural network for clustering. The proposed temporal coding SNN is structured similar to Kohonen's self-organising map and encodes input information in its connection weights. Recent findings in computational neuro science are incorporated for modelling the proposed learning paradigm. This novel learning model was found to be better than the previous unsupervised SNN models as well as the Kohonen's self-organising map in terms of clustering accuracy and efficiency.

This chapter is structured as follows: the structure and functionality of the Kohonen's self-organising map is explained in section 3.2. Section 3.3 summarises the previous work done in the area of unsupervised learning in spiking neural networks. This section also describes the learning strategies in neural systems and two findings which formed the basis of this study. The proposed model

is introduced and analysed in detail in section 3.4. Section 3.5 describes the implementation details of the proposed model. The model has been tested in a number of simulations in order to investigate its characteristics and clustering capability. Section 3.6 describes these simulations in detail, presents the results obtained and discusses them. Finally, the conclusions are given in section 3.7.

3.2 Kohonen's self-organising map

Kohonen's self-organising map is an artificial topographic mapping paradigm which learns through self-organisation in a neurobiologically inspired manner [Haykin, 1999]. In the human brain it was observed that different sensory inputs are mapped onto corresponding areas of cerebral cortex in an orderly fashion. According to the principle of topographic map formation in human brain, the spatial location of an output neuron in a topographic map corresponds to a particular domain or features drawn from the input space [Kohonen, 1990]. Hence, the basic idea behind the SOM is that an input space can be topographically mapped, where data objects with similar features are represented with distinct regions in the map. A model for the SOM was initially proposed by Willshaw and von der Malsburg [Willshaw and von der Malsburg, 1976] and later by Kohonen [Kohonen, 1982]. The Kohonen model, which is more general than the previous one, has received much attention and is widely used in practice. The principle goal of Kohonen's SOM is to transform a signal pattern of arbitrary dimension into a one- or two-dimensional discrete map and to perform this transformation adaptively in a topologically ordered fashion.

Like all the neural network models, the SOM is composed of a network structure and a learning algorithm. The network is constructed with a layer of input neurons and an output layer of computing neurons. The neurons in the output

layer can be arranged in a single array or in a two dimensional lattice. Generally, a rectangular lattice is used for ease of implementation. The input neurons are connected to all the output neurons with feed-forward connections. Each connection is assigned a weight value to represent its strength. In addition, the neurons in the output layer have a short range excitatory mechanism and a long range inhibitory mechanism through lateral connections. Figure 3.1 shows a typical structure of the SOM.

The self-organising algorithm produces a topographical map of the arbitrary dimensional input space through the response of one- or two-dimensional arrays of output neurons. This learning algorithm for forming the topographical map is composed of three essential processes, namely, competition, cooperation and adaptation. In competitive mode, the computing neurons in the output layer compete with each other during the learning process. The winner neuron is determined by comparing the input and the connection weights of a particular neuron which are two vectors. The winner is the one whose weight vector is closest to the input vector in terms of Euclidean distance. The amount of modification for the connections to the winner neuron is relatively higher than other neurons. In general, the weights of neurons are updated according to their closeness to the input vector.

Cooperation among neurons is essential in order to form regions in the output map for representing clusters in the input space. The competition identifies the centre of a topological neighbourhood and the neurons in this neighbourhood need to be enhanced to cooperate with the winner while suppressing the neurons which are located outside this neighbourhood. This is achieved by means of the lateral excitation and inhibition mechanism in the output layer.

The adaptation mode governs the updating of the network's connection weights

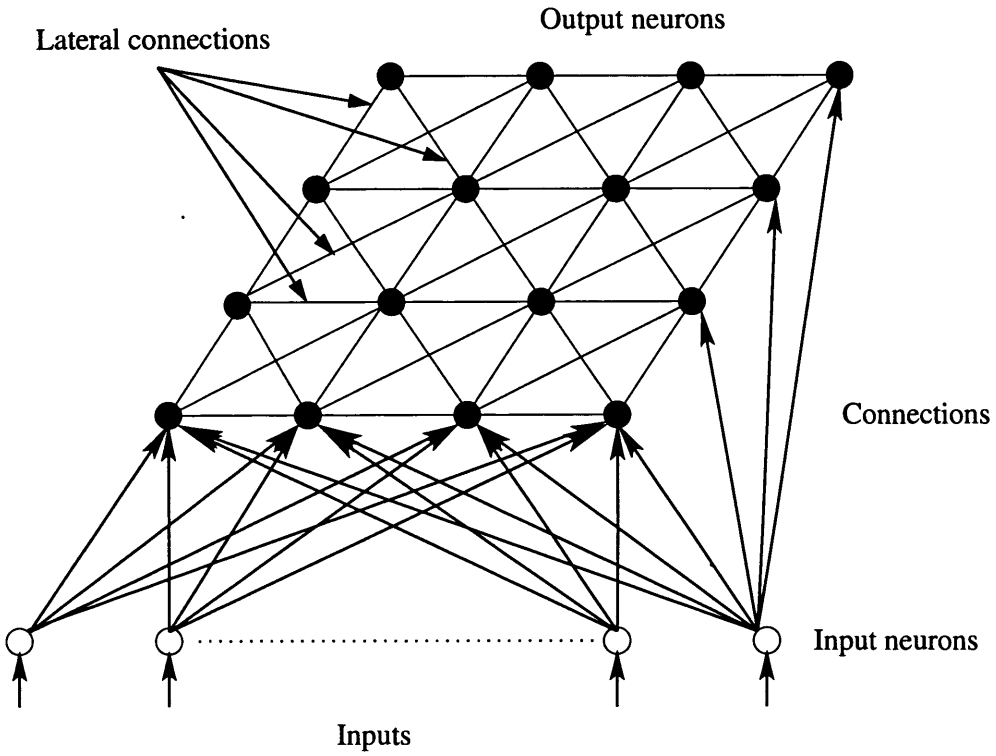


Figure 3.1. Kohonen's self-organising network. Redrawn from [Pham and Liu, 1999].

in relation to an input vector. This is achieved according to Hebb's postulate, which describes the learning mechanism found in biological neural systems. It states that a synaptic connection's efficacy is increased if the pre-synaptic activity and post-synaptic activity occur simultaneously. The self-organising algorithm based on a modified Hebb's rule is given below in detail [Haykin, 1999].

The self-organising algorithm

step 1: *Initialisation.* Create a network with a number of input neurons equal to the size of the input vector and an adequate number of output neurons. Assign random values for the connection weight vectors $\mathbf{w}_j(0)$. The weight vectors $\mathbf{w}_j(0)$ should be different for output neurons j, $j = 1, 2...., m$ where m is the number of output neurons and it is advantageous if the initial weight vectors are small in magnitude.

step 2: *Sampling.* Draw an input vector $\mathbf{x}$ from the input space with a certain probability. Activate the network with that pattern and compute the outputs.

step 3: *Similarity matching.* Find the best matching neuron $i(\mathbf{x})$ at time step n using the minimum distance Euclidean criterion specified in equation 3.1. This will select the neuron with the closest match between the weight vector and the input pattern, which is equivalent to finding the neuron with the maximum output.

$$i(\mathbf{x}) = arg\ min_j \|\mathbf{x}(n) - \mathbf{w}_j(n)\|, j = 1, 2, ..., m \tag{3.1}$$

step 4: *Updating.* Update the connection weights of all the neurons with the following formula given by equation 3.2:

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n) h_{ji(\mathbf{x})}(n)(\mathbf{x}(n) - \mathbf{w}_j(n)) \tag{3.2}$$

where $\eta(n)$ is the learning-rate parameter. The function $h_{ji(\mathbf{x})}(n)$ realises cooperation among neurons in a neighbourhood around the winner neuron $i(\mathbf{x})$ as defined by equation 3.3:

$$h_{ji(\mathbf{x})}(n) = exp(-l_{ji}^2/2 * \sigma(n)^2) \tag{3.3}$$

where l_{ji} is the lateral distance between neurons i and j; $\sigma(n)$ defines the neighbourhood of interest. Both $\eta(n)$ and $h_{ji(\mathbf{x})}(n)$ are varied dynamically during learning for best results.

step 5: *Continuation.* Continue from step 2 until no noticeable changes in the feature map are observed.

3.3 Research on unsupervised SNN models and biological neural networks

One of the differences between the SNNs and the previous artificial network models is the representation of the information within the network. There are several neuron models which use the spikes as the means for computation and communication. But much of these neuro computational models utilise the amplitudes of the spikes rather than their firing time. Hopfield [Hopfield, 1995] introduced the idea of using the timing of action potentials to represent the values for computation within the network. Instead of the usual approach of representation of rate of firing, the variables were represented by the timing of action potentials with respect to an ongoing oscillatory activity. Much of the later research on learning in spiking neural networks was based on the notion introduced in [Hopfield, 1995]. This section presents a summary of the unsupervised learning models for SNNs and biological neural networks utilising the timing of action potentials. In the context of SNNs, two kinds of models can be found in the literature. One category encodes the input information in the connection weights and the other in

the connection delays, but both achieve this through some form of Hebbian-based self-organised weight adaptation.

3.3.1 Weight-based learning

A model for self-organisation in a network of spiking neurons which encodes the input information in connection weights was proposed by Ruf and Schmitt [Ruf and Schmitt, 1998]. Their work showed that the spiking neural network model along with temporal coding is capable of preserving the topology of the input space similar to Kohonen's self-organising map. The network was similar to Kohonen's SOM except that the output neurons were spiking neurons proposed by Maass [Maass, 1996]. In order to realise cooperation among neurons, lateral connections were employed in the output layer. Training the network with an input vector spans over a time window. Inputs were converted into temporal code and during the training the input spikes were presented to the network accordingly. For an input value x_i an input spike was generated at t_i $(= T_1 - x_i)$ where T_1 is some constant referred to by the timing of a reference spike by a neuron additional to the input neurons. With the assumption that the potential rise due to an input spike is linear, each competitive neuron can be expected to fire at $T_2 - \sum_i w_i x_i$ with T_2 being a constant. It was shown by Maass [Maass, 1997b], that a higher potential in a spiking neuron represents an early output spike. Hence it was assumed that if the input vector and the weight vector are normalised then the value $\sum_i w_i x_i$ represents the similarity between the two vectors in terms of the Euclidean distance. Therefore, the neuron which fires first was considered to have a weight vector which is closer to the input vector. The lateral connections were realised as strongly inhibitory and a straightforward *winner-takes-all* learning rule specified in equation 3.4 was utilised.

$$\delta w_{ji} = \eta(x_i - w_{ji}) \tag{3.4}$$

where δw_{ji} is the modification for the connection between neurons i and j with weight w_{ji}. $\eta\,(>0)$ is the learning rate and x_i is the input.

In order to realise cooperation among neurons in the neighbourhood of the winner neuron, two simple measures were implemented. Neurons which were topologically closer were assigned to strong excitatory lateral connections and remote neurons were assigned to strong inhibitory lateral connections. Through these connections, the neurons which are closer to the winner neuron are encouraged to fire while the other neurons are discouraged. In addition, the neurons which fire temporally closer to the winner neuron are encouraged more than the neurons which fire later after the winner neuron. The self-organising rule proposed in [Ruf and Schmitt, 1998] is specified in equation 3.5.

$$\delta w_{ji} = \eta \frac{T_{out} - t_j}{T_{out}} (x_i - w_{ji}) \tag{3.5}$$

where δw_{ji}, η and x_i are same as defined in equation 3.4. The synaptic modification specified by the equation 3.5 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 $(T_{out} - t_j)/T_{out}$, defines the effect on neighbouring neurons based on the spike time. Since the winner is the one which fires first, this term will be high for neurons which fire early and low for neurons which fire late. The research described above analysed several aspects of learning with temporal coding spiking neural networks through testing the model on one- and two-dimensional artificial input patterns.

3.3.2 Delay based learning

A number of researchers have investigated the learning in SNNs through delay adaptation. In fact, most of these models select suitable delayed connections from

a set of different delayed connections through unsupervised learning rules. These rules enhance the strength of some connections while weakening the others, resulting in some selected delayed connections with high strength and the rest with very low strength. This technique was first introduced by Hopfield [Hopfield, 1995], which showed that the input spike patterns can be stored in the delays across the synapses. The notion behind this strategy is that an encoded delay pattern will compensate for 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 [Gerstner et al., 1996]. This type of learning is claimed to be more like the learning in a RBF network. This approach is supported by neurobiological findings reported in [Habberly, 1985; O'Keefe and Reece, 1993]. The novel learning model proposed in this chapter encodes the input information in the connection weights. The reason for summarising the delay-based learning here is to investigate the spike-time-based learning strategies used in the past. The Hebbian-based learning models described in this sub-section modify the connection weights based on the time difference between the pre- and post-synaptic firing of a neuron.

Gerstner et al. [Gerstner et al., 1996] performed a modelling study through computer simulations of the barn owl's auditory system. A Hebbian-based unsupervised learning mechanism was proposed to train a single integrate-and-fire neuron. Here, a single connection to a neuron was composed of several sub-connections. Each sub-connection was characterised with a weight and a delay value. The learning rule proposed in their model enhances the strength of the connections which are repeatedly active shortly before a post-synaptic spike event. Connections which are active shortly after the post-synaptic event are weakened. In this way the learning model selects connections with suitable delays from a distribution of connections with different delays. It was also showed that this

learning method can choose the correct delays for inputs from two independent groups. Figure 3.2 shows the learning rule proposed in their study.

Natschläger and Ruf [Natschläger and Ruf, 1998] extended the approach reported in [Gerstner et al., 1996] for spiking neural networks. In addition, the firing times of the output neurons were considered instead of the approach in [Gerstner et al., 1996], where the firing or non-firing of a neuron was taken into consideration. The network architecture used here was a two layered fully connected feed-forward network. The output layer was constructed with spiking neurons. A sample network is shown in figure 3.3. As in the previous model a single connection was composed of several sub-connections. The learning rule proposed in [Natschläger and Ruf, 1998] to calculate the amount of change in the synaptic strength is given by the equation 3.6 and graphically shown in figure 3.4.

$$\delta w_{ji}^{(k)} = \eta\, L((t_i + d^{(k)}) - (t_j + d^{back})) \tag{3.6}$$

where $\delta w_{ji}^{(k)}$ is the amount of modification to the synaptic strength $w_{ji}^{(k)}$ of the sub-connection k between neurons i and j; $d^{(k)}$ is the delay for the sub-connection k between two neurons and η is the learning rate. A spike at time t_i is the input to the neuron j and t_j is the time of an output spike. The delay d^{back} is the time a post-synaptic spike takes to propagate backwards to the synapse. Here the function L is given by equation 3.7.

$$L(\delta t) = (1 - b)exp(-(\delta t - c)^2/\sigma^2) + b \tag{3.7}$$

where,

$$\delta t = t_{pre} - t_{post} \tag{3.8}$$

$$t_{pre} = t_i + d^{(k)} \tag{3.9}$$

$$t_{post} = t_j + d^{back} \tag{3.10}$$

where b, c and σ are constants.

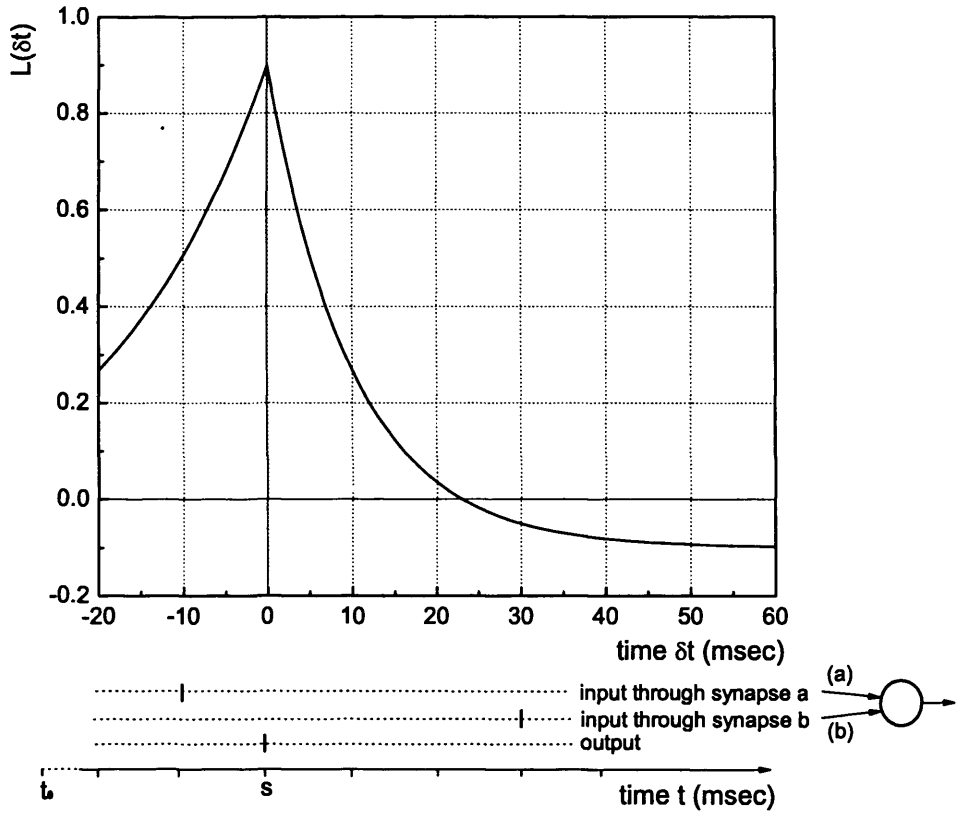


Figure 3.2. Gerstner et al.s' learning rule. Here a post-synaptic spike occurs at time s. Connections are strengthened if pre-synaptic spikes arrive shortly before the post-synaptic neuron starts firing as in synapse (a). The connections are weakened if pre-synaptic spikes arrive after the post-synaptic firing as in (b). Redrawn from [Gerstner et al., 1996].

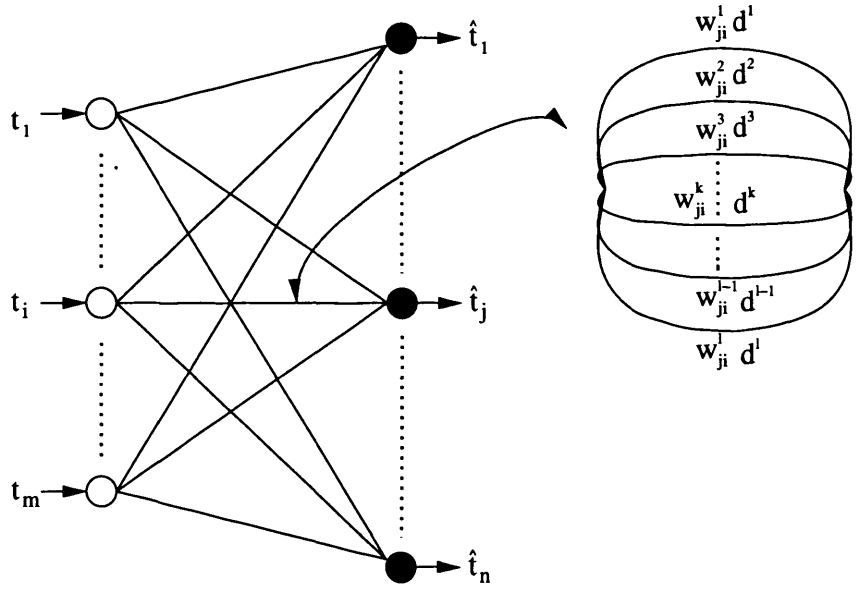


Figure 3.3. Natschläger and Rufs' network model with multiple sub-connections. Redrawn from [Natschläger and Ruf, 1998].

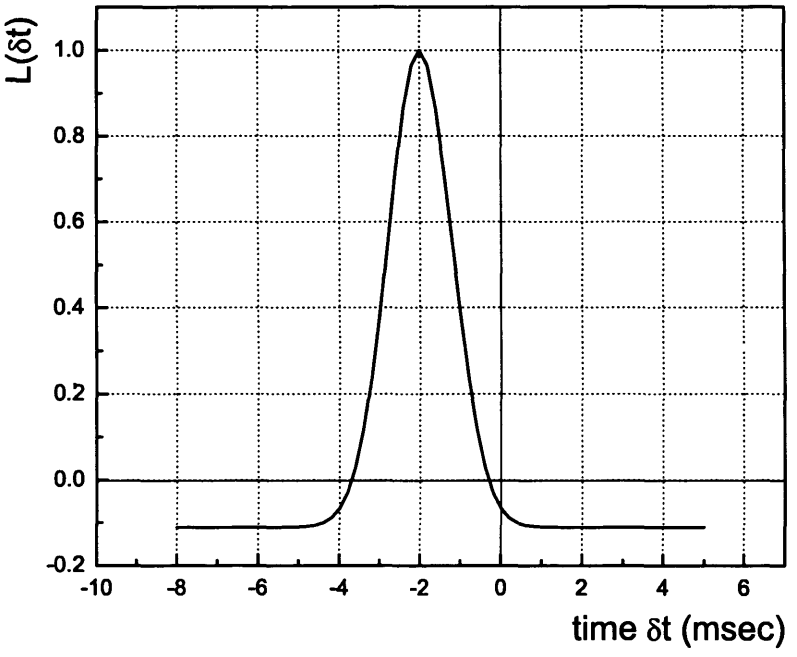


Figure 3.4. Natschläger and Rufs' learning rule for unsupervised learning. Redrawn from [Natschläger and Ruf, 1998].

The model proposed in [Natschläger and Ruf, 1998] was further improved by Bohte et al. [Bohte et al., 2002b] 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 tested by clustering several data sets.

3.3.3 Research on biological neural networks

The concept of artificial neural networks stems from various disciplines including neuroscience, mathematics, statistics and computer science. The creation and the ongoing research on artificial neural networks was mainly based on the astounding findings from research on biological neural networks. The modern era of neural networks began with the pioneering work by McCulloch and Pitts [McCulloch and Pitts, 1943] which united the knowledge from neurophysiology and mathematical logic [Haykin, 1999]. This section describes the basics on learning in biological neural systems and some recent findings which provided significant contributions to this research.

3.3.3.1 Learning in biological neural networks

The plasticity property of synapses is the basis for learning, memory and development in neuronal circuits. Synaptic plasticity is the ability of a synapse to modify itself due to the activity it is involved in so that it adapts to its environment and retains information. These changes are likely to be both functional and structural but in most cases are considered as the modification of its conduction ability. It is commonly accepted that the correlation between the activities of neurons is the basis for synaptic modifications after Hebbs postulate. This postulate, which was formulated on purely theoretical grounds, is still accepted even though this was proposed long before the actual recordings of neuronal events

became available. Learning in neural systems is supported by a rich collection of information obtained through vast research in diverse disciplines. However, there are no definite rules available to define the synaptic plasticity in neural systems, but only general guidelines. These can be summed up into three rules as given below [van Rossum et al., 2000]:

Rule 1: The synaptic strength should be allowed to change in response to the inputs depending on their correlation with the post-synaptic firing or with the activity of other neurons [Sejnowski, 1977].

Rule 2: The outcome of these changes should result in a stable distribution of synaptic strength [Abbott and Nelson, 2000].

Rule 3: The learning rules should create competition among the group of neurons concerned, and hence some synapses should be strengthened while some others should be weakened [Carla, 1990; Miller, 1996].

One or more of these rules formed the basis for the models of biological systems as well as for the artificial neural network models proposed in the past. In most cases, the neuronal activity was measured through the firing rate of neurons. However, in recent years there has been a growing interest in neuronal models which are based on the timing of spikes instead of the firing rate of neurons. Learning is achieved in these models through the correlation between the timing of pre- and post-synaptic spikes. Two concepts, namely spike-time based learning and stabilising the learning, proposed in modelling studies by Song et al. [Song et al., 2000] and van Rossum et al. [van Rossum et al., 2000], formed the basis for the self-organised learning paradigm proposed in this study. The following sub-sections discuss these two concepts in detail.

3.3.3.2 Spike-time-based learning

Synaptic modification due to the timing of synaptic activity has been observed by several researchers [Bell et al., 1997; Markram et al., 1997; Bi and Poo, 1998]. This type of modification is termed as *Synaptic Timing Dependent Plasticity* (STDP). Bi and Poo [Bi and Poo, 1998] performed experiments in cultures of dissociated rat hippocampal neurons and observed that the relative timing between the pre-synaptic and post-synaptic firing events determine the direction and amount of synaptic change. Repetitive post-synaptic firing within a time window of 20 msec after pre-synaptic activation strengthened the synapse, while the post-synaptic firing within a window of 20 msec before the pre-synaptic activation resulted in synaptic depression.

Several modelling studies have been performed to analyse spike-dependent plasticity rules [Blum and Abbott, 1996; Gerstner et al., 1996; Eurich et al., 1999; Kistler and van Hemmen, 2000; Song et al., 2000; van Rossum et al., 2000]. Song et al. [Song et al., 2000] performed a modelling study on competitive Hebbian learning through synaptic timing-dependent plasticity. It was found that this form of learning modifies the system to be sensitive to pre-synaptic spikes. The Hebbian learning rule proposed in [Song et al., 2000], given by equation 3.11, defines the amount of synaptic modification arising from a single pre- and post-synaptic spike pair separated by a time δt. The figure 3.5 illustrates this learning rule.

$$F(\delta t) = \begin{cases} A_+ \, e^{\delta t/\tau_+} & : \quad \delta t < 0 \\ -A_- \, e^{-\delta t/\tau_-} & : \quad \delta t > 0 \end{cases} \tag{3.11}$$

where $\delta t = t_{pre} - t_{post}$; t_{pre} is the firing time of the pre-synaptic spike and t_{post} is the timing of the post-synaptic spike. The parameters τ_+ and τ_- determine the ranges of pre- to post-synaptic inter spike intervals over which synaptic strengthening and weakening occur. A_+ and A_- determine the maximum amount of synaptic

modification per spike pair.

3.3.3.3 Stabilising the Hebbian learning

Activity dependent modification of synapses is a powerful and widely accepted mechanism for shaping and modifying the response properties of neurons. However, there is no controlling mechanism to restrict the modification. Unregulated modification leads to uncontrolled expansion (or depression) of synaptic strength [Song et al., 2000] which leads to an unstable system. To control this unstable nature of Hebbian learning, a common practice is to impose a hardbound for the upper and lower values of the synaptic strength [Blum and Abbott, 1996; Gerstner et al., 1996; Song et al., 2000]. A Hebbian-based rule, when applied repeatedly on a network, will increase the strength of some of the connections while decreasing the others. Hence, when a hardbound is applied, the connection strengths will be driven to the defined upper or lower values, resulting in a bimodal distribution of the connection weights. It is unlikely for this bimodal distribution to reflect the synaptic strength distributions observed in biological systems where the distribution of synaptic weights is observed to be unimodal [van Rossum et al., 2000].

A phenomenon observed in previous studies [Debanne et al., 1996; Bi and Poo, 1998] could be a key for solving the instability problem. Observations show that a significant amount of strengthening occurs on relatively weak synapses. For strong synapses, the change due to the potentiation was observed to be low. However, the depression was found to be independent of the initial synaptic strength. van Rossum et al. [van Rossum et al., 2000] incorporated the above mentioned findings in a modelling study to investigate the stability of Hebbian-based spike time dependent plasticity. The learning rule proposed in their study

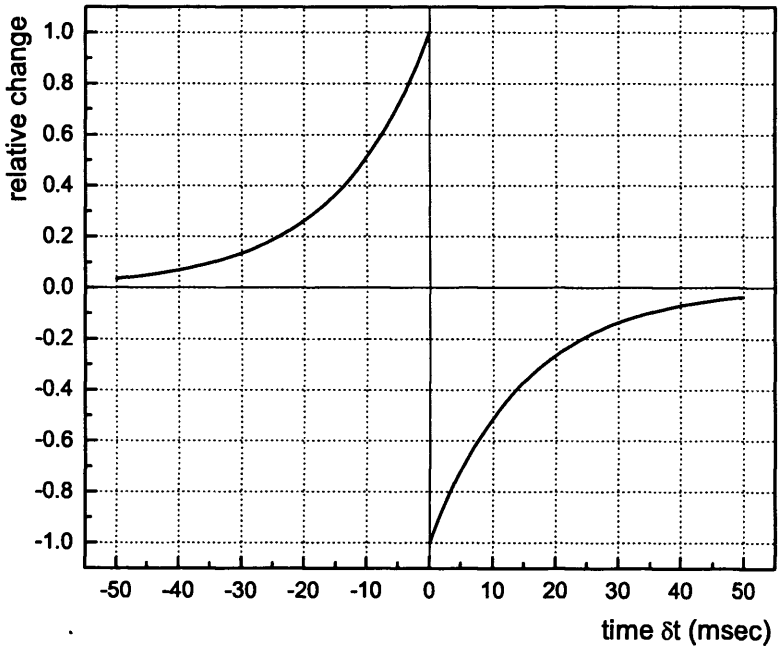


Figure 3.5. Song et al.s' learning rule for unsupervised learning. Redrawn from [Song et al., 2000].

is given by equations 3.12 and 3.13.

$$w_p = (c_p + vw)e^{-\delta t/\tau} \tag{3.12}$$

$$w_d = (-c_d w + vw)e^{\delta t/\tau} \tag{3.13}$$

where w is the strength of the connection between two neurons. w_p is the amount of modification for potentiation and w_d is for depression. c_p and c_d are average amount of potentiation and depression respectively. v is a Gaussian random variable with zero mean and standard deviation of 0.015. δt is the time difference between the pre- and post-synaptic spikes of the neuron in concern. τ is the time constant for potentiation and depression. For potentiation $w(t+1) = w(t) + w_p$ and for depression $w(t+1) = w(t) + w_d$.

Figure 3.6 graphically describes the weight-dependent learning concept proposed by van Rossum et al. [van Rossum et al., 2000]. It was found that the learning rules produced a stable unimodal distribution of connection strengths. The underlying mechanism in the above mentioned scheme is that it gradually slows down the rate of change if the weight approaches one of the boundaries. This type of stabilisation is known as *softbounding*. A simple model for realising this softbounding is given by equations 3.14 and 3.15 [Kistler and van Hemmen, 2000]:

$$f_+(w_{ji}) = (w_{max} - w_{ji})\, a_+ \tag{3.14}$$

$$f_-(w_{ji}) = -w_{ji}\, a_- \tag{3.15}$$

where w_{ji} is the strength of the connection between neurons i and j. Function f_+ is the stability measure for potentiation and f_- for depression. The two parameters a_+ and a_- are constants.

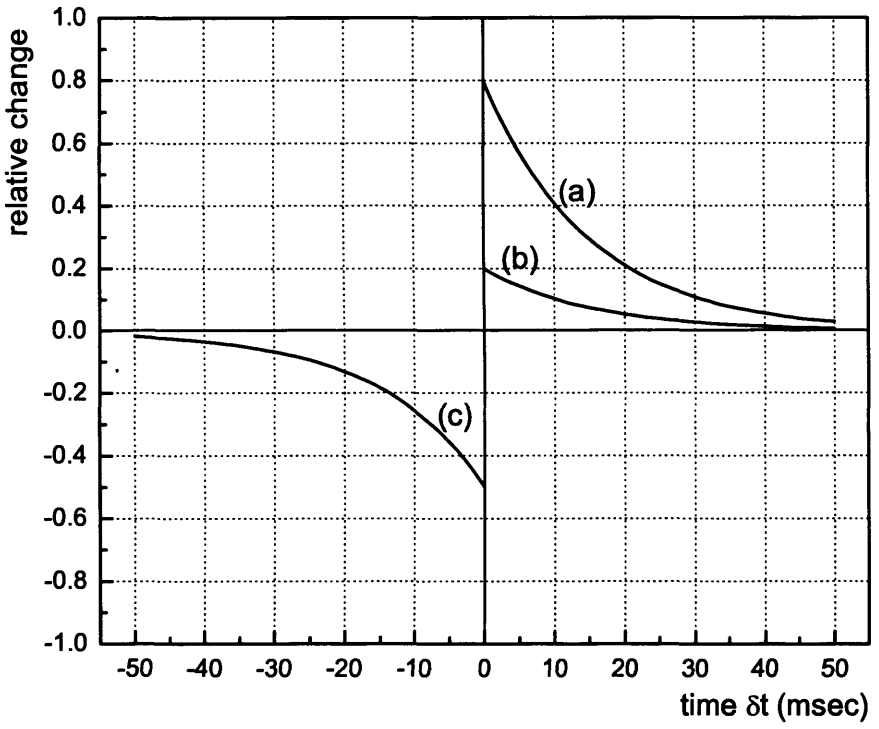


Figure 3.6. van Rossum et al.s' rule for synaptic modification. (a) Relative strengthening of weak connections, (b) relative strengthening of strong connections and (c) relative weakening of all connections. Redrawn from [van Rossum et al., 2000].

3.4 Proposed self-organising weight adaptation SNN for clustering

The proposed model is presented in this section. Modelling of the network has been approached according to the three rules specified in subsection 3.3.3.1. The new model is realised with four distinct and necessary features. They are spike-time-based learning, self-organisation, stabilising the learning and spike-time-based output mapping. Based on the findings described in subsection 3.3.3.2, a new approach on spike-time-based learning is proposed. The self-organisation is realised in the proposed model in a way similar to the popular approach in Kohonen's SOM. Stabilising the connection weights during learning is achieved based on the findings reported in subsection 3.3.3.3. For mapping the output, the actual firing times of the output neurons are utilised, in addition to the common practice of the spatial location of the competing neurons. These issues are explained clearly in the following sub-sections.

3.4.1 Network architecture

The network architecture chosen for the proposed SOWA_SNN is similar to the Kohonen's SOM. A two layered fully connected feed-forward network of spiking neurons is selected for this purpose. The first layer is the input layer with the number of neurons equal to the number of input parameters. The output layer is constructed with spiking neurons as described in section 2.5. Neurons are arranged in a two dimensional grid but more complex or simpler grid structures can be adopted. Connections from the input neurons to the output neurons are characterised by a weight value and a delay value. Since the learning model proposed in this chapter considers only the connection weights, the connection delays can be removed.

3.4.2 Spike time based learning

A variant of the Hebb's postulate is the basis of the proposed learning model. A connection weight is increased if an input spike through that connection precedes an output spike of the receiving neuron. If the input spike follows the output spike then the connection is weakened [Song et al., 2000; van Rossum et al., 2000]. The amount of change due to the spike activity is defined in equation 3.16 which is shown graphically in figure 3.7:

$$g(\delta t) = \begin{cases} e^{-\delta t/\tau_{stdp}} - b & : \quad \delta t \geq 0 \\ -e^{\delta t/\tau_{stdp}} & : \quad \delta t < 0 \end{cases} \tag{3.16}$$

where δt is the difference between the timing of the input spike and the time of generation of an output spike; τ_{stdp} is the time constant for potentiation and depression. The parameter b is the bias, a positive term used to control the learning.

3.4.3 Self-organisation

Self-organisation is achieved through competition and cooperation among the spiking neurons. In the Kohonen's SOM a winner neuron is the one which produces the highest output. Neurons in the neighbourhood of the winner neuron are encouraged to cooperate. The neighbourhood is realised through the physical location of a neuron relative to the winner neuron [Haykin, 1999]. A similar approach is utilised in the proposed model. The winner neuron is selected as the one which fires first among the spiking neurons [Bohte et al., 2002b], since a highly activated neuron tends to fire earlier [Thorpe et al., 2001]. During the training process, connections of the neurons which are laterally closer to the winner neuron are updated proportional to its distance from the winning neuron. The effect on neuron j relative to neuron k is given by equation 3.17.

$$h(j,k) = e^{-(d(j,k))^2/2\sigma} \tag{3.17}$$

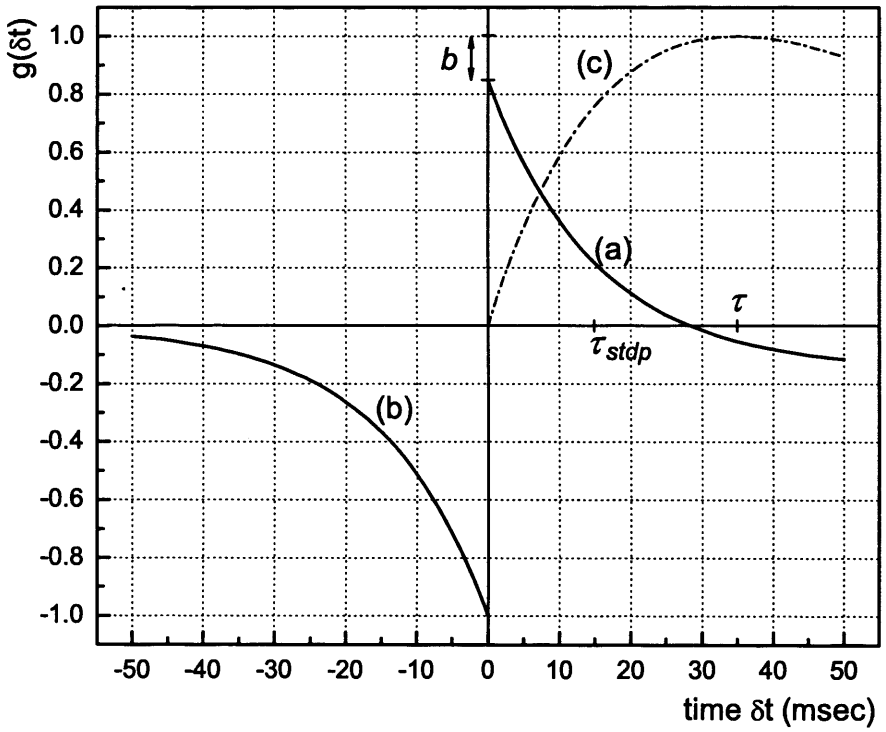


Figure 3.7. Learning rule for the proposed model. (a) shows part of the learning rule for $\delta t \geq 0$, (b) for $\delta t < 0$. Curve (c) shows the excitatory spike response function, which is given here for ease of comparison. Here, the time constant for the learning rule (τ_{stdp}) is set to 15 and the time constant for the spike response function (τ) to 35.

where $d(j,k)$ is a function to compute the lateral distance between neurons j and k; the parameter σ specifies the width of the neighbourhood of cooperation centred at k, which is reduced with training epochs. In equation 3.17 the parameter σ was used instead of σ^2 in order to provide high lateral excitation for neurons which are closer to the winner neuron and low excitation for neurons which are further away from the winner neuron.

3.4.4 Stabilising the weight adaptation

An important improvement proposed in this study is the stabilisation of the connection weights during the training process, which is a major problem encountered in previous models. This problem arises due to the true nature of Hebbian-based learning, since it does not have a constraint to restrict the growth or reduction of the synaptic strength. Some previous models applied a hard limiting constraint to solve this phenomenon. If the synaptic strength exceeds (or falls below) the allowed maximum (or minimum) value, then it is set to that maximum (or minimum) value. This hard limiting constraint leads to a bimodal distribution of weights. Normalisation of the connection weights after each learning cycle has been utilised as another means for solving this drawback but with much computational expense [Haykin, 1999]. The above mentioned methods, namely applying a hard limiting constraint and normalising of weights, considered the synaptic strength as the one being modified without providing a control mechanism for the weight adaptation. A suitable method is incorporated in this study where the change in weight for a particular connection is controlled by the current weight value. In other words, the change realised for a weak connection is relatively higher than for a strong connection. This approach is supported by the findings in [Bi and Poo, 1998; van Rossum et al., 2000] which were explained in subsection 3.3.3.3. In the proposed model, the change in a connection weight

during strengthening is constrained by a factor which is inversely proportional to the current weight as formulated in equation 3.18. Depression is defined as to be independent of the current connection weight. This equation specifies the maximum change attainable for a particular connection weight.

$$f(w_{ji}) = (w_{max} - w_{ji})/w_{max} \tag{3.18}$$

where w_{ji} is the current strength of the connection between input neuron i and output neuron j, w_{max} is the maximum value for any connection strength.

3.4.5 Learning rule

In the above three sub-sections, the three main aspects of the self-organised learning in spiking neural networks, namely activity-based learning, self-organisation and stabilisation were discussed. A novel learning rule for the proposed model is defined by combining these three aspects, since they all influence the learning process. The proposed learning rule, which is given by equation 3.19, specifies the amount of weight change to the connection from neuron i to j when the firing time of the input spike from neuron i and the firing time of the output spike from neuron j differ by time δt.

$$\delta w_{ji} = \eta \; g(\delta t) \; h(v, j) \; f(w_{ji}) \tag{3.19}$$

where the functions g, h and f are defined in equations 3.16, 3.17 and 3.18 respectively. δw_{ji} is the amount of weight change; η is the learning rate parameter; v is the winner neuron, which fires first among the competing neurons; δt is the time difference.

3.4.6 Learning strategy

Learning in the artificial neural networks is generally supported with sound mathematical and/or statistical concepts and theories. These concepts and theories

clearly define the relation between the input, the output and the network parameters. For example, the output of a sigmoidal neuron is in fact a representation of the Euclidean distance between the input vector and the connection weight vector [Haykin, 1999]. An input vector in the input space is represented by an output neuron which has the closest weight vector to that input. The training of the network tends to minimise the difference between the input vectors and weight vectors.

Since SNNs are different from sigmoidal networks, it is important to understand the relation between the connection weights and the temporally coded inputs [Ruf and Schmitt, 1997]. The model proposed in [Ruf and Schmitt, 1997] modifies the connection weights to approach some value given by the difference between the pre-and post-synaptic firing times in an inversely proportional manner. However, the applicability of their model for complex applications was not verified. The self-organising model proposed in [Bohte et al., 2002b] encodes the input patterns in the connection delays through weight adaptation where the SNN functions similar to RBF network.

The learning process of the model proposed in this study is based on the Kohonen's SOM. In Kohonen's SOM, the neuron with the highest output is selected as the winner neuron since the maximisation of the inner product of two vectors corresponds to the minimisation of the Euclidean distance of those vectors [Haykin, 1999]. In the proposed model, the winner neuron is selected as the one which fires first among the competing neurons [Bohte et al., 2002b]. If the input vector and the weight vector are normalised, then the neuron potential computed by equation 2.8 represents the similarity between the two vectors in terms of the Euclidean distance [Ruf and Schmitt, 1998]. Also, it was shown in [Maass, 1997b] that a higher potential in a spiking neuron represents an early output spike. Hence it can be assumed that the neuron which is first to fire has the

weight vector with the closest match to the input pattern. It is notable here that in the proposed spiking neural network, connection weight values are confined within a range [0, 1) and the effect of each input spike is also in the range [0, 1]. Therefore, the learning strategy of the SOWA_SNN can be considered as similar to Kohonen's SOM.

3.4.7 Interpreting the output and cluster identification

Out of a number of available methods for cluster identification in Kohonen's SOM, the method of visualising the location of the winner neurons and their frequencies is widely employed [Vesanto and Alhoniemi, 2000]. The output of the SNN is the firing times of its output neurons. The exact firing time of a neuron conveys significant amount of information [Thorpe et al., 2001]. This work incorporates the exact firing time of the winner neuron to map the clusters in addition to the winner neuron's spatial location. In this study, it was found that similar input patterns tend to excite a particular output neuron to fire within some well defined time interval. The firing time of a winning neuron, along with its position in the output layer lattice, formed clusters which reflected the topographical mapping of the input space. Hence, a cluster can be specified with a particular set of neurons and a particular firing time interval. For mapping the clusters after the training phase, the input patterns were presented to the network in the recall mode and the outputs were recorded. The location and the firing time of the winner neuron for each input pattern was found and plotted in a three-dimensional space. In this three-dimensional plot, clusters were identified with groups of adjacent neurons firing within a particular time window.

3.5 Implementation details

The SNN with the proposed self-organising learning model was implemented in software. The continuous SNN was realised as a discrete model in order to simulate it on digital computer. Realisation of the spiking neural network through a discrete model was discussed in section 2.5.4. More detail on this software and the program listings are given in Appendix A.

In the SNN model each cycle of activity takes a period of time where an input pattern is presented to the network to produce an output. Similarly, each cycle of activity in the discrete model takes a number of steps. In each step, input neurons are scanned for the availability of input spikes and the potential of each neuron is updated based on the timing of the input spikes. If any of the neurons' potential exceeds the threshold, an output spike is generated. For ease of implementation each step was considered to take one unit time to complete. The resolution of the temporally coded input and output was scaled to one unit time. During each cycle of activity, the network was allowed to operate for an adequate number of steps in order to realise the full effect of each input and to reflect this effect through its output. This number of steps is referred to as *activation time window* (t_{window}) in this study.

Another important aspect of implementing the model is the temporal coding of the continuous input values. The issues regarding temporal coding were explained in section 2.3. A straightforward coding scheme was used where a relatively high input value was represented with an early input spike and a low value with late spike [Bohte et al., 2002a]. The continuous values were normalised and temporally coded using the linear coding scheme specified in equation 2.14 over a number of time steps referred to as *input time window* (t_{input_window}). The coding resolution affects the performance of the network. Since the temporal coding

is some form of discretisation, an increase in the number of steps in the input time window will increase the precision. However, the increase of the number of steps will decrease the computational efficiency of the model. Hence, precision of the temporal code should be selected in such a way as to attain adequate accuracy with optimal computational efficiency. Initial investigations revealed that an input time window with 30 units was adequate in most cases and even less in some cases.

The SNN has 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. Even though a number of researchers have done significant work on SNNs, there is a lack of clear guidelines available for selecting these parameters. In this research these parameters were found by analysing the preliminary results obtained from initial trials. The strategy followed in this study was to set all the neurons with the same parameter values, particularly for the time constant and threshold. Although theoretically it is possible to have different parameter values for each neuron, having same parameter values for all the neurons will simplify the network and aid the understanding of the results.

Generally, the spike response function of a synapse can be either excitatory or inhibitory, which will decide the effect of an input spike instead of the sign of the connection weight [Maass, 1997a]. In the proposed model, the connection weights were assigned with positive values and all the connections were realised as excitatory with spike response function $\epsilon(t) \in [0,\ 1]$. This is to ensure that the effect of most of the input parameters contribute positively to the output. The selection of an appropriate value for the time constant τ depends primarily on the size of the input time window. The selection should ensure that the output of a neuron reflects the effects of all its inputs. The selection should also enable

the neuron to fire at a time when the effect of all its inputs is in their increasing segment. The spiking neuron functions as an integrator in this proposed model. Based on the suggestions proposed in [Konig et al., 1996], the time constant was selected to be slightly greater than the input time window in order to allow the spiking neuron to function as an integrator. Section 2.5.2 introduced the two modes of operations of the spiking neuron, namely, integration and coincidence detection. The two modes of operation of a spiking neuron will be discussed in more detail in chapter 4 (Section 4.2.3). A notable consideration here is that an increased input time window will increase the precision of the inputs which will in turn increase the value for the time constant. But a high value for the time constant will result in reduced precision of the spike response function. Hence a trade-off must be found to obtain an optimum result. Selecting the values for the time constant, the threshold, initialising the network connection parameters and selection of suitable values for the learning equation parameters, are discussed in the following sub-sections.

3.5.1 Initialising the connection weights and delays

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 connection weights are allowed in the range [0, 1) and the delays in the range of [0, t_{input_window}]. Initial connection weights were randomly assigned within a small interval in order to ensure that no neuron can dominate all the other neurons for all or most of the input patterns. The objective of the initial weight assignment is to have the winner neurons evenly spread all over the output grid for the input patterns. Since the connection delays were not considered in this proposed model, they were set to zero.

3.5.2 Setting the threshold value

More care is needed in the selection of the threshold value for a neuron. If the value is too low, the neurons will fire prematurely without reflecting the entire input pattern. On the other hand, a high threshold value will prevent neurons from firing and will block the learning process because a neuron can learn only when it is active. Hence the threshold was set to an appropriate value to capture the effect of all or most of the inputs while ensuring that it can fire. A suitable threshold value could be the number of inputs multiplied by the average of the connection weights multiplied by the average spike response value.

3.5.3 Setting the parameters of the learning rule

Selection of suitable values for the parameters of the learning rule (equation 3.19) is critical for better learning and stability. The two important parameters, namely the time constant τ_{stdp} and the bias term b were selected with the help of some guidelines provided in [van Rossum et al., 2000] and some initial trials of the proposed model on various data sets. A synapse was heavily strengthened if a spike through it reached a neuron just before a post-synaptic event. The amount of strengthening was decreased with the increase in the difference between the arrival time of the input spike and the generation of the output spike. The amount of change was decreased and at some point made to become negative in order to stabilise the learning process. Better learning was observed when the change in weight was made to reach zero when the time difference was approximately equal to the time constant τ for the spike response function, as shown in the figure 3.7. It was found that a suitable choice for τ_{stdp} is roughly the half of the value of τ. The value for the bias term b was selected such that it forces the weight change to become zero and then negative when the time difference becomes approximately

equal to the time constant τ and higher. A suitable value for b was found to be in the range [0.1, 0.2].

3.6 Simulation results and discussion

The SOWA_SNN was implemented and applied to clustering tasks in order to investigate its characteristics. The main focus was on assessing its clustering capability and determining the behaviour of the proposed model due to the measures introduced in the learning rule. The following sub-sections discuss these issues in detail.

3.6.1 Clustering capability

The clustering capability of the SOWA_SNN was analysed by applying it to cluster a number of data sets. For each data set a custom made network was constructed. The number of input neurons for each network was set to be equal to the number of input attributes of the data set in concern. The number of output neurons and the size of the output grid were selected after several trials. Out of the total available data, approximately 66% of the records were randomly selected and used for training for a fixed number of epochs. The number of training epochs was determined by trial and error such that towards the end of the training the change in the total weight adaptation became very low according to a predefined value. On completion of the training, the network's connection weights and the output of the network for each sample were recorded. The output of the network was analysed and clusters were identified. Details regarding cluster identification were discussed in sub-section 3.4.7. In order to determine the clustering accuracy, the identified clusters were verified using their class information. The *clustering accuracy* is defined as the percentage of correctly classified input patterns from

the entire input data.

The generalisation capability of the proposed model and the validity of the formed clusters were also analysed by clustering a test data set using the trained network. The test set contains approximately 33% of the total data which was separated from the whole data set and not used for training. The process of training and testing the proposed model was repeated five times and the average accuracies were calculated for each data set. The number of attributes, number of classes and the number of records used for training and testing are given in table 3.1. The data used for training and testing is given in Appendix B.

The clusters formed within the training and testing set of the Iris data are shown in figures 3.8 (a) and figure 3.8 (b) respectively. Likewise the clusters formed within Cancer, Wine and Control chart data sets are shown in figures 3.9, 3.10 and 3.11 respectively. Using the information gained from the firing time of the winner neurons, clusters could be separated as shown in these figures. These results suggest that the use of the firing time of the winner neurons reduces the number of output neurons required to map the clusters.

Figure 3.8 shows the formation of clusters within the Iris data set. Here most of the input samples belonging to class 1 are represented by the neuron at grid position (5, 2). Two samples belonging to class 1 are represented each at positions (2, 3) and (4, 2). Input samples from class 2 and class 3 are indicated by neurons at (2, 3), (3, 3) and (4, 4). Further, class 2 and class 3 are separated by the firing time of the neurons. In this case, if the output firing time is less than or equal to 16, then the sample belongs to class 3. If the firing time is equal or greater than 17 then that sample belongs to class 2.

The average clustering accuracy obtained for each data set is given in table 3.2. This table lists the average clustering accuracy obtained on both the train-

Data set	Number of		Total	No. of records	
	attributes	classes	records	training	testing
Iris	4	3	150	105	45
Cancer	9	2	683	450	233
Wine	13	3	178	120	58
Control chart	60	6	1500	1000	500

Table 3.1. Number of samples used for training and testing.

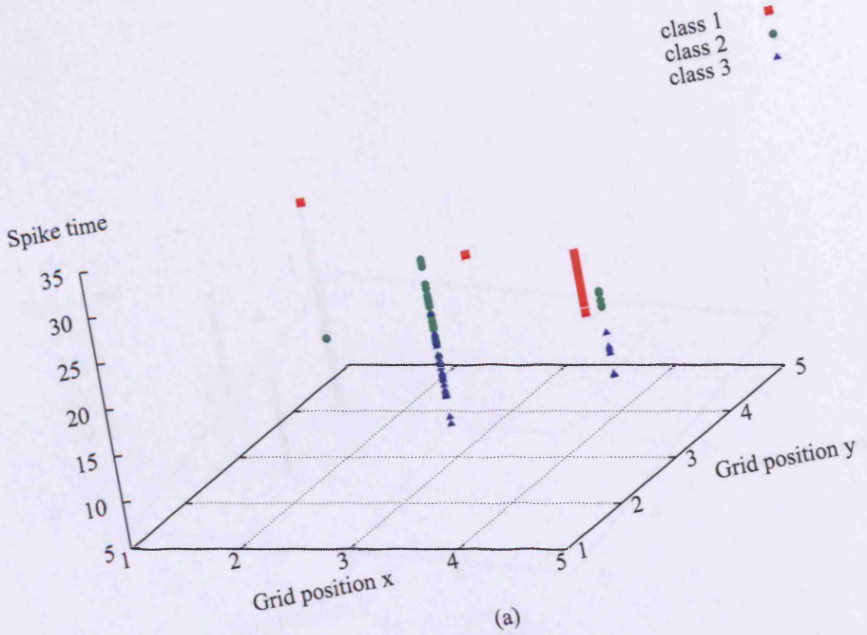


Figure 3.8. The clusters formed within (a) the training set and (b) the test set of the Iris data using SOWA_SNN.

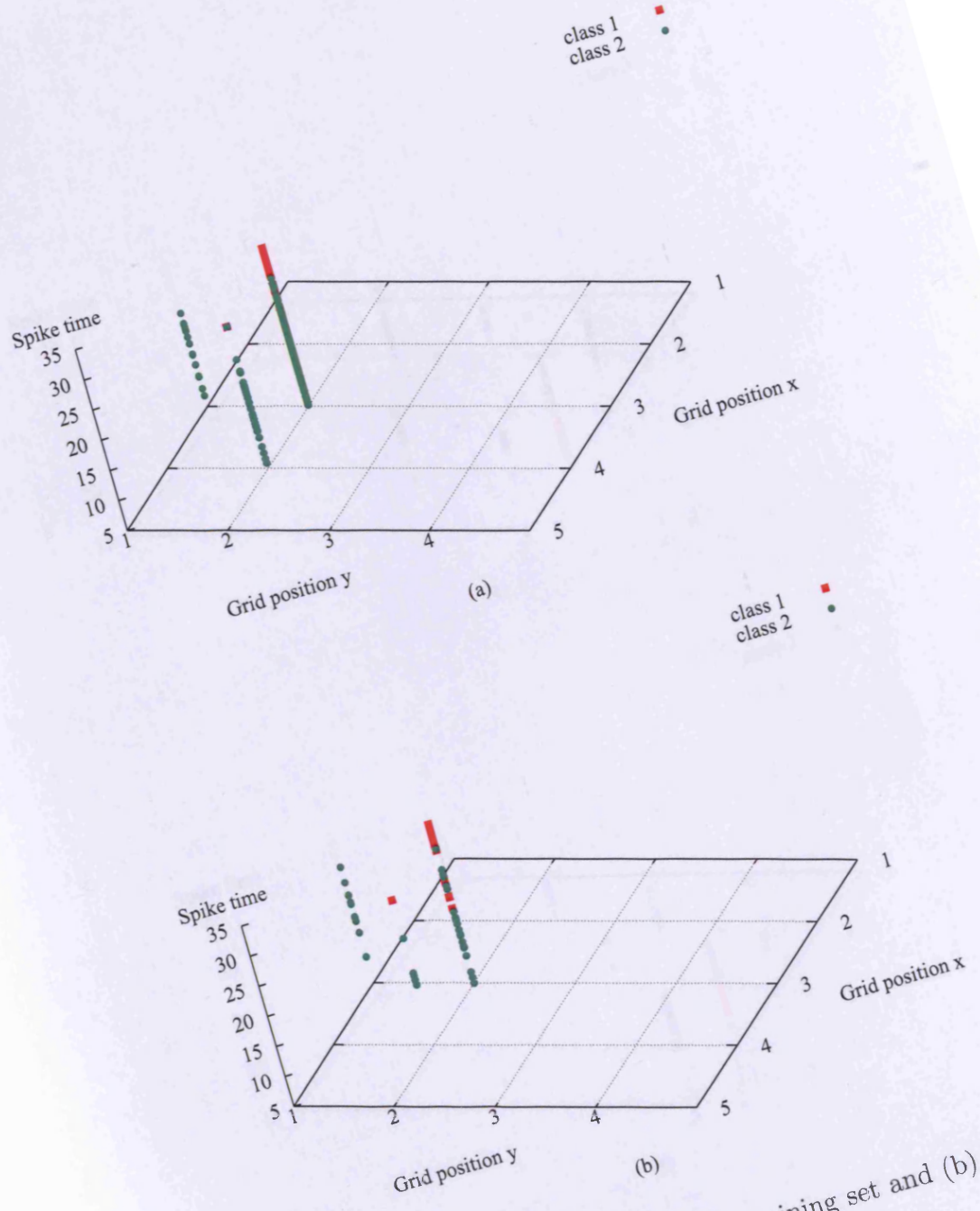


Figure 3.9. The clusters formed within (a) the training set and (b) the test set of the Cancer data using SOWA_SNN.

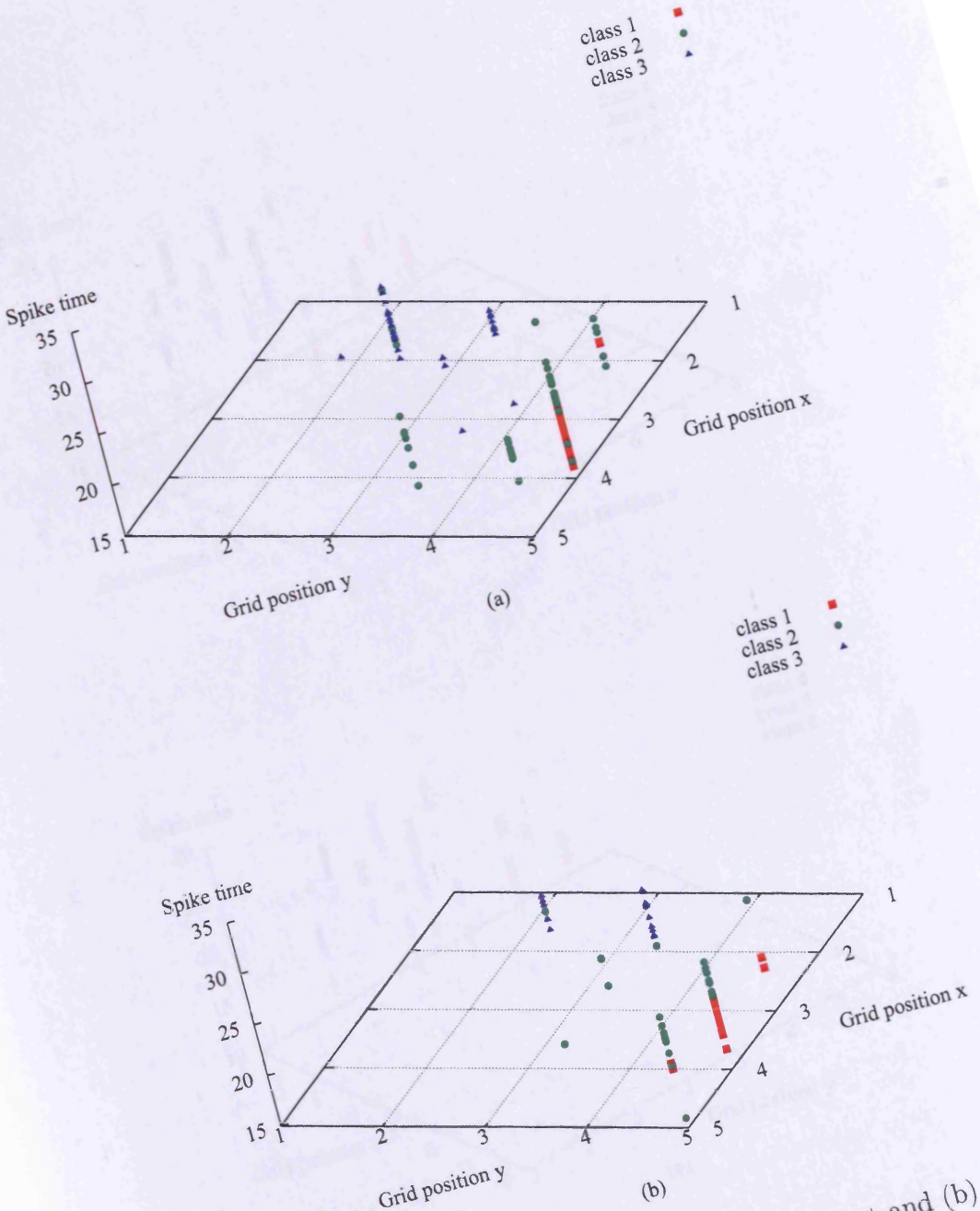


Figure 3.10. The clusters formed within (a) the training set and (b) the test set of the Wine data using SOWA_SNN.

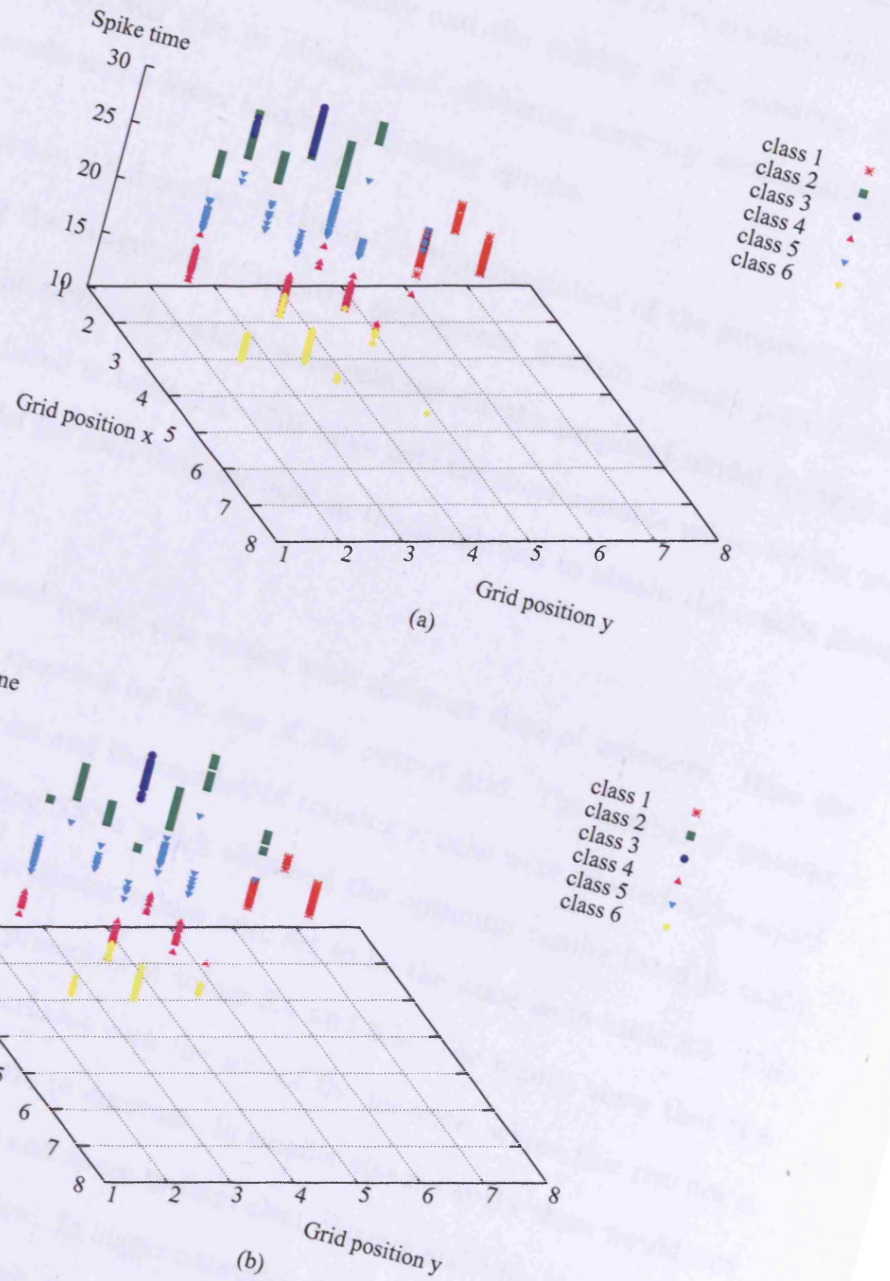


Figure 3.11. The clusters formed within (a) the training set and (b) the test set of the Control chart data using SOWA_SNN.

ing and the test sets along with the standard deviation. The results show that the SOWA_SNN achieved better clustering accuracy for all the data sets. The accuracy obtained for the test sets can be viewed as an evidence for the proposed model's generalisation capability and the validity of the clusters. In addition, the model was able to obtain good clustering accuracy with relatively smaller networks and a lesser number of training epochs.

Section 3.5 describes in detail the implementation of the proposed model including the assignment of network parameters. Certain network parameters described in section 3.5 which were selected for the proposed model by trial and error are listed in table 3.3. This table lists the most suitable values for the proposed model for each data set used in the simulation to obtain the results listed in table 3.2.

The proposed model was tested with different sizes of networks. Here the network size is specified by the size of the output grid. The number of training and testing samples and the number of training epochs were selected to be equal to the corresponding SNNs which obtained the optimum results listed in table 3.2. The network parameter values were set to be the same as in table 3.3. The results obtained are presented in tables 3.4 and 3.5. The results show that the clustering accuracy increases with the size of the network, where this reaches a maximum and then starts to decrease. In smaller size networks there would not be adequate randomness and space to form clear clusters and thus the clustering accuracy was found to be low. In bigger networks, the training would be dispersed all over the network forming several overlapping sub-clusters, thus resulting in reduced accuracy.

Self-organising SNNs proposed in [Ruf and Schmitt, 1997, 1998] were applied to artificial data sets to prove their clustering ability. The unsupervised model

Data set	No. of neurons		No. of training epochs	Clustering accuracy(%)	
	Input	Output		Training	Testing
Iris	4	5x5	20	96.4±0.4	96.9±1.2
Cancer	9	5x5	15	98.0±0.4	96.9±0.7
Wine	13	5x5	25	94.5±0.5	94.4±0.7
Control chart	60	8x8	25	95.8±0.6	95.7±0.7

Table 3.2. Average clustering accuracy obtained for the SOWA_SNN.

Data set	t_{input_window}	τ	τ_{stdp}	b	η	Threshold
Iris	30	35	15	0.15	0.007	4 x 0.5 x 0.5
Cancer	30	35	17	0.1	0.001	9 x 0.5 x 0.5
Wine	30	35	17	0.15	0.02	13 x 0.5 x 0.5
Control chart	30	35	15	0.15	0.003	60 x 0.5 x 0.5

Table 3.3. Values assigned for the parameters of the network used to obtain the results listed in table 3.2.

Network size	Clustering accuracy(%)					
	Iris		Cancer		Wine	
	training	testing	training	testing	training	testing
3x3	92.8±3.5	94.4±5.9	97.8±0.7	96.7±1.0	92.2±2.1	89.1±1.0
4x4	94.2±2.0	91.7±1.9	97.6±0.3	97.1±0.9	93.0±2.7	92.0±3.6
5x5	96.4±0.4	96.9±1.2	98.0±0.4	96.9±0.7	94.5±0.5	94.4±0.7
6x6	95.2±0.9	94.2±3.2	95.6±1.2	93.6±3.1	93.6±0.5	90.8±2.0
7x7	94.7±1.1	93.6±3.5	93.8±1.4	92.1±3.7	92.8±0.7	89.4±2.3

Table 3.4. Average clustering accuracy obtained for different size of SOWA_SNNs.

Network size	Clustering accuracy(%)	
	Control chart	
	training	testing
6x6	84.2±2.4	80.7±1.8
7x7	95.3±2.6	94.7±1.9
8x8	95.8±0.6	95.7±0.7
9x9	89.8±2.2	88.8±1.9
10x10	89.6±2.4	87.9±1.8

Table 3.5. Average clustering accuracy obtained on Control chart data for different size of SOWA_SNNs.

proposed in Bohte et al. [2002b] encodes the input patterns primarily through the connection delays. The model proposed in [Bohte et al., 2002b] obtained a clustering accuracy of $92.6 \pm 0.9\%$ for the Iris data set, where the inputs were encoded with 4x8 input neurons. The clustering accuracy of the proposed model for Iris data set has been found to be better than the model proposed in [Bohte et al., 2002b]. In addition, the SOWA_SNN uses single connections and a temporal coding scheme which represents an input value by a single spike time. Whereas in the model proposed in [Bohte et al., 2002b], a single connection is made up of a group of sub-connections and requires a population coding scheme for temporal code the input values. Hence the performance of the proposed model can be considered to be better than the previous SNN models.

For comparing the clustering capability of the proposed model, the same data sets were clustered with Kohonen's self-organising map using *ESOM Analyzer* - a shareware program from the Databionics research group [Ultsch and Möerchen, 2005]. Initially, the size of the network and the training epochs were selected to be equal to the corresponding SNN models. Each data set was clustered with ESOM Analyser five times and the average accuracy was found. The clustering accuracies obtained for the training and testing sets are listed in table 3.6. With the specified size of the network and training epochs, the clustering accuracy for the proposed model has been found to be better than the SOM for all the data sets.

The SOMs mentioned above were further trained until they reached their optimum performance. Table 3.7 lists the average highest clustering accuracies obtained for SOM. Although the performance of SOM on Iris data and Control chart data was better on the training data, the clustering accuracy of SOWA_SNN on the test data was higher than the SOM. SOWA_SNN obtained better accuracy than the SOM on both training and test sets of Cancer and Wine data. For all

the data sets, the SOWA_SNN required a smaller number of training epochs than the SOM to reach the highest clustering accuracy. The results obtained demonstrate the superior clustering and generalisation capability of the proposed model compared to the standard Kohonen's SOM.

3.6.2 Stability

Uncontrolled Hebbian learning could lead to an unstable system. Issues regarding stability were discussed in sections 3.3.3.3 and 3.4.4. In order to determine the behaviour of the stability measures in the proposed model, the distribution of the connection weights of the trained networks were analysed. The weight distribution of the network trained on the Iris data set is shown in figure 3.12. Likewise, the weight distributions for the networks trained on Cancer, Wine and Control chart data are shown in figures 3.13, 3.14 and 3.15 respectively. Here the SOWA_SNNs analysed are those obtained the optimum results in the previous sub-section. These figures show that during training, while some of the connection weights were strengthened, others were weakened, allowing the network to reach a stable condition with unimodal weight distribution. This shows that the stabilisation measures incorporated in the proposed model effectively contribute to the learning process, resulting in an acceptable distribution of the connection weights to achieve a stable network.

3.7 Conclusion

This chapter proposed the Self-Organising Weight Adaptation Spiking Neural Network (SOWA_SNN) for clustering. The proposed model's structure and functionality is similar to Kohonen's SOM. In addition, a number of biological findings were incorporated in this temporal coding SNN model.

Data set	No. of neurons		No. of training epochs	Clustering accuracy(%)	
	Input	Output		Training	Testing
Iris	4	5x5	20	95.0±1.4	93.3±2.7
Cancer	9	5x5	15	96.8±0.3	96.1±0.2
Wine	13	5x5	25	77.4±1.9	73.2±3.2
Control chart	60	8x8	25	95.1±0.4	94.6±0.7

Table 3.6. Average clustering accuracy obtained for Kohonen's SOM. The software package ESOM was utilised for this purpose.

Data set	No. of neurons		No. of training epochs	Clustering accuracy(%)	
	Input	Output		Training	Testing
Iris	4	5x5	30	96.8±0.5	96.4±1.2
Cancer	9	5x5	20	97.6±0.4	96.3±0.5
Wine	13	5x5	30	78.7±0.5	78.3±0.9
Control chart	60	8x8	30	96.1±0.9	95.5±0.9

Table 3.7. Average highest clustering accuracy obtained for Kohonen's SOM.

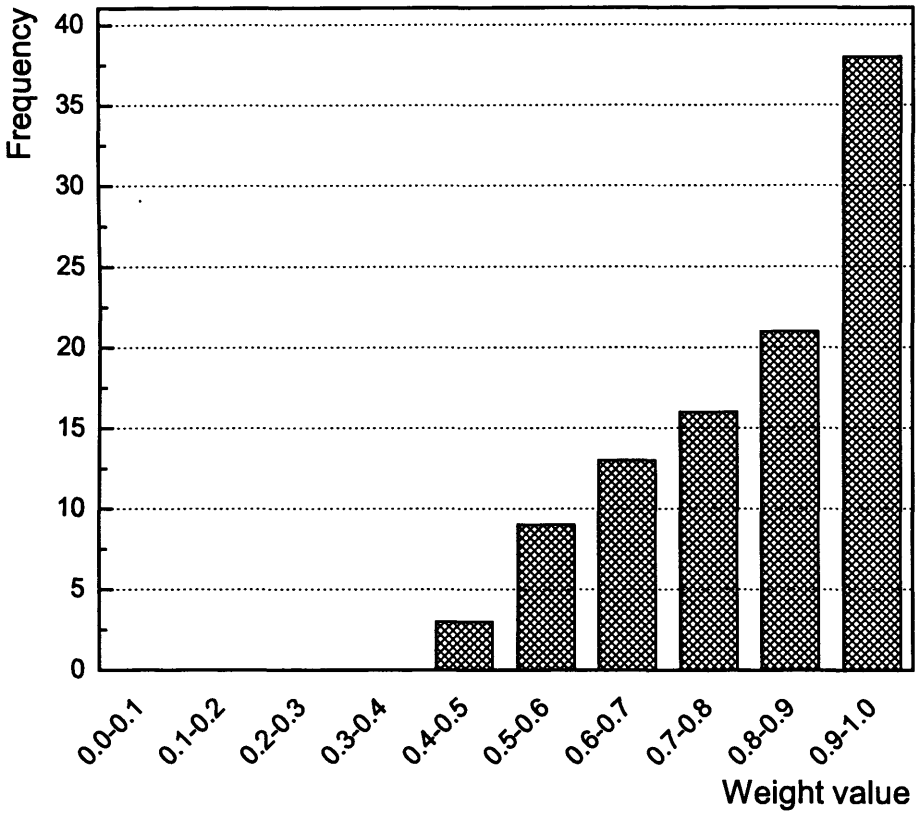


Figure 3.12. Distribution of the connection weights of the SOWA_SNN trained on Iris data.

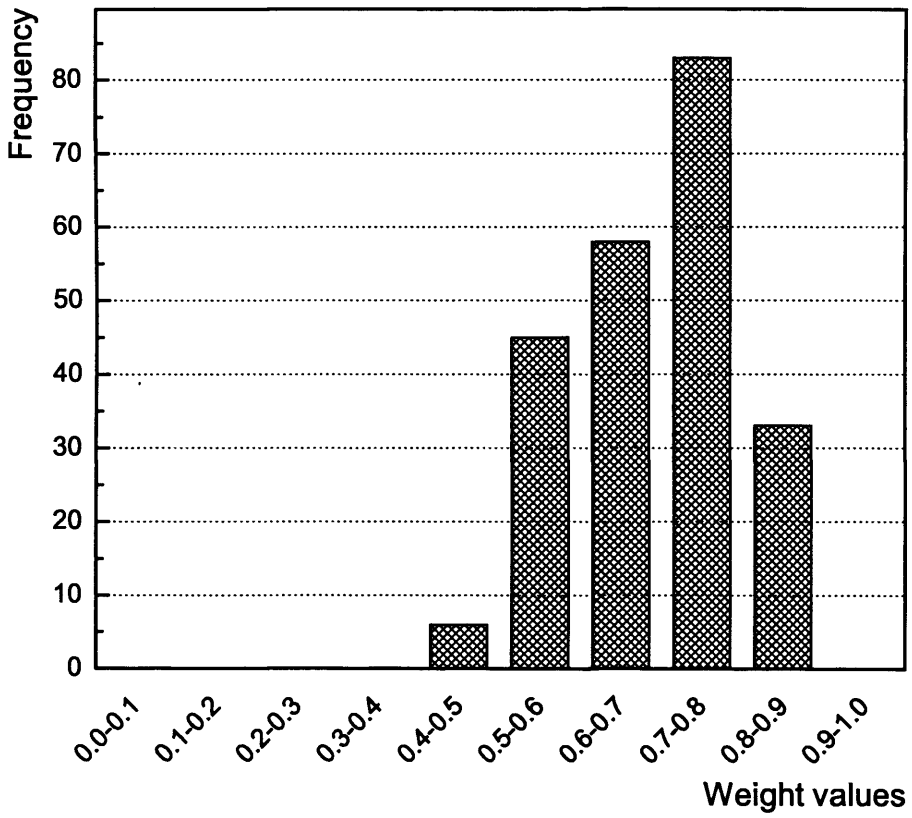


Figure 3.13. Distribution of the connection weights of the SOWA_SNN trained on Cancer data.

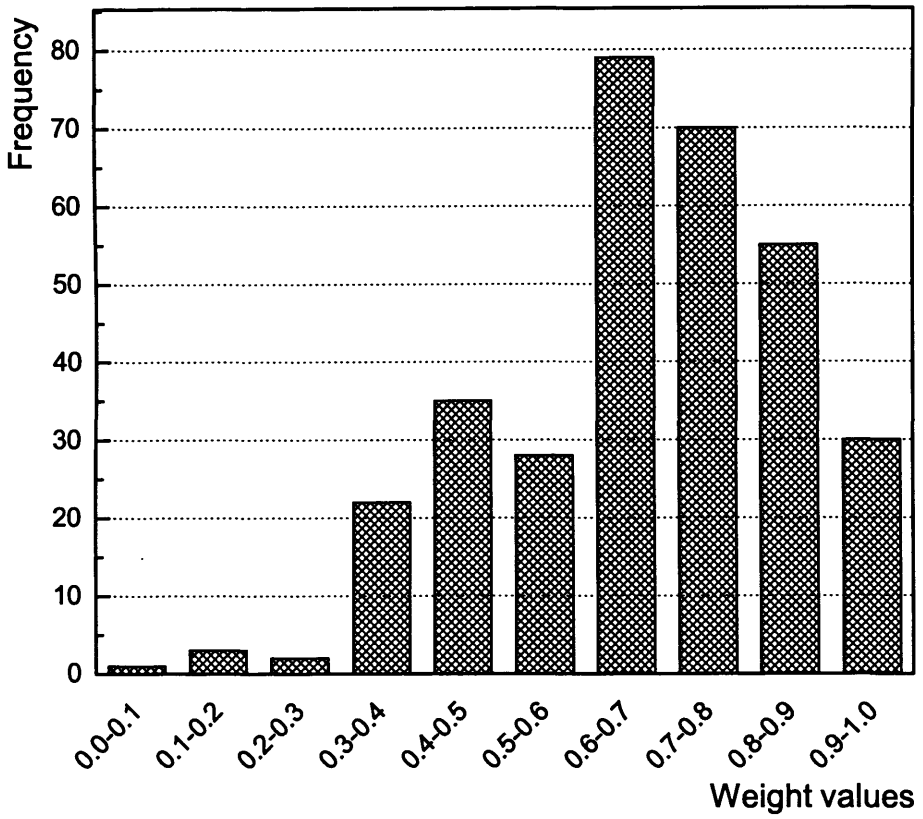


Figure 3.14. Distribution of the connection weights of the SOWA_SNN trained on Wine data.

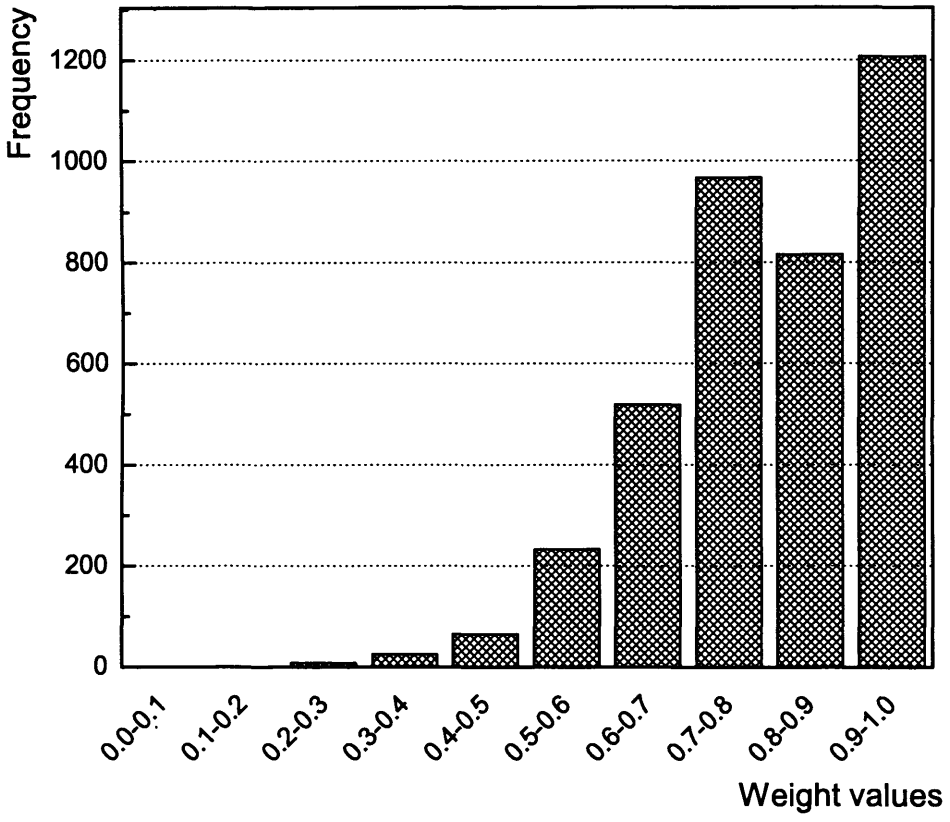


Figure 3.15. Distribution of the connection weights of the SOWA_SNN trained on Control chart data.

The SOWA_SNN adapts the connection weights in order to encode the input information in its connection weights. The model employs a Hebbian-based learning rule utilising the timing of the pre- and post-synaptic spikes. A soft bounding approach has been incorporated for stabilising the weights during learning. The locations of the winner neurons and their firing times have been utilised effectively to identify the clusters within the input data.

The performance of SOWA_SNN has been tested by clustering four data sets and the model exhibited high degree of clustering accuracy. The proposed network has been found to be performing better than the previous unsupervised spiking neural network models developed for clustering. The performance of the proposed model has been compared with Kohonen's SOM and found to be better. The weight distribution of the trained networks was analysed. The unimodal distribution of the connection weights shows that the learning is well stabilised with the incorporated measures. In essence, the SOWA_SNN can be considered as an alternative tool for clustering tasks.

Chapter 4

Self-Organising Delay Adaptation Spiking Neural Network (SODA_SNN)

4.1 Introduction

This chapter investigates the unsupervised delay adaptation learning in spiking neural networks and proposes a novel self-organising delay adapting spiking neural network for clustering. The proposed temporal coding spiking neural network realises the spiking neurons as coincident detectors rather than as integrators. A delay shifting rule has been employed to adapt the connection delays in order to detect input patterns through enabling coincidence among inputs. The SODA_SNN was implemented in software and experimentally analysed to determine its characteristics. It was successfully applied to cluster several data sets and the results demonstrated its capability and efficiency over other existing models.

This chapter is structured as follows: section 4.2 reviews the notion of delays, coincidence detection and delay adaptation learning. Section 4.3 summarises the previous research on delay based learning. The new model is proposed in section 4.4. The implementation details are given in section 4.5 and section 4.6 presents

the results and discussion. Finally the conclusions are given in section 4.7.

4.2 Neuronal delays and coincidence detection

The concept of delays is quite common in biological neural systems. Not only in biological systems but in general for any system which conveys signals through some medium, the delays are inevitable. This section gives a review of neuronal delays and coincidence detection in the biological systems as well as in the artificial systems. In addition, the concept of delay-based learning is also introduced here.

4.2.1 Neural systems and Delays

Unlike artificial systems, biological systems are non-uniform in a structural sense. In a biological neural network, the spatial distribution of neurons is not consistent. Neurons are connected in a very complex pattern, with a varying length of connectors. The conductance of the neuronal connections significantly varies due to the presence of different type of ions and a varying number of ion channels. Neuronal time delays, or simply delays, arise due to these finite differences in neural systems [Eurich et al., 1999]. Because of these differences, a propagating neural signal will not reach all the receiving neurons at the same time. It is evident that delays are present in electrical and electronic circuits due to the length of connections and their conductivity similar to the biological neural circuits. While the delays in these man-made circuits are often considered as a hindrance to the circuits' computational capability, in some biological systems they are essential for coordinating the activity. Good examples are the auditory system of barn owls, the echo location of bats and the lateral line systems found in weakly electric fish [Carr, 1993]. A possible way of using these neuronal de-

lays in neural processing was identified in [Abeles, 1982]. It was found by Konig et al. [Konig et al., 1996] that the cortical neurons act as coincidence detectors and it was suggested that this could be the appropriate code for higher cortical functions. This seems to be in contrast to the notion of a neuron acting as an integrator. Nevertheless, biological neurons are known to operate in two modes, namely, as a coincidence detector and as an integrator.

4.2.2 Neuron as a coincidence detector

Carr [Carr, 1993] analysed the processing of temporal information in the nervous system belonging to three species, namely, weakly electric fish, barn owls and echo locating bats. These three species are well known because of their superior sensing ability to detect minute changes in the environment. It was pointed out that the temporal information processing mechanisms depend on some form of delay lines and coincidence detection. Here, neurons were found to be excited maximally only when they receive input spikes simultaneously and are thus functioning like coincidence detectors. In other words, these neurons generate output spikes if and only if their inputs coincide. In reality, however, the inputs to the neurons are not evenly distributed in time and thus the delays of the connections through which these input spikes travel to the receiving neuron compensate for these differences and make them coincide at some point in time.

A good example to illustrate the importance of delays and the role of a neuron as a coincidence detector is the functioning of *auditory place cells* found in the barn owl. These cells are believed to be performing the important task of computing the azimuthal position of a sound source. This is achieved by detecting the time coincidence between the signals originated from each ear due to the reception of sound from the source. The coincidence will be detected in a cell whose connections from each ear have delays representing the different lengths of

the paths between the sound source and each ear [Hopfield, 1995].

4.2.3 The operating mode of a neuron

Neurons have been found to be operating in two distinct modes, as an integrator and as a coincidence detector [Konig et al., 1996]. The previous sub-section (sub-section 4.2.2) described the functioning of the neuron as a coincidence detector. In the coincidence detecting mode, the neuron fires when its inputs coincide or when it receives synchronised inputs or simply when it receives its inputs within a small time window. In the integrator, mode the neuron integrates or sums up the inputs it receives and generates an output when its potential exceeds a threshold value. Notice that in both cases the underlying mechanism is based on summing up the effect of all inputs received by the neuron and generating an output if it satisfies the threshold criterion. The effect of an input at some time t is defined by the spike response function. The potential of a neuron represents the collective effect of all the inputs it receives. This potential increases with the arrival of input spikes and dissipates with time, due to the non-linear nature of the synapse. Figure 2.7 describes the effect of an input spike over time. The distinction between the two modes of operation depends on the configuration of the neuron and the connections.

An acceptable explanation for the two distinct modes of operation can be found in [Konig et al., 1996]. They made the distinction based on the duration of the operating interval or interaction interval of a neuron. If the operating interval is equal to or greater than the mean inter-spike interval, then the neuron will act as temporal integrator. On the other hand, the neuron will function as coincidence detector if the integration interval is shorter. Due to the short integration time, the output will become sensitive to the arrival timing of the incoming spikes, causing the neuron to act as coincidence detector [Hopfield, 1995].

This phenomenon of delay lines and coincidence detection of neural systems was first proposed by Jeffress [Jeffress, 1948] and was further researched by Abeles [Abeles, 1982].

4.2.4 Pattern detection with coincidence detecting spiking neuron

Coincidence detection is an important feature of a spiking neuron. A good example of how this could be realised was given in sub-section 2.5.2.3. This feature has no analogy in the computational units of conventional neural network models [Maass, 2001a]. It was proved that the spiking neurons outperform sigmoidal neurons in this regard [Maass, 1997b]. As a coincidence detector, a neuron can generate an output spike only when it receives coinciding inputs. This feature can be utilised efficiently to detect input patterns using the connection delays. The strategy here is that it is possible to find a set of delays which can compensate for the differences in the firing times of the input pattern and enable them to coincide at an output neuron. Hence, when similar patterns are temporally coded, a spiking neuron can detect them with a finely tuned set of delayed connections. The finely tuned delays can be viewed as a representation of similar input patterns and can be utilised for encoding information [Hopfield, 1995]. Obtaining the correct delay patterns can be achieved through a suitable learning method. A scenario which explains the coincidence detection by a spiking neuron can be found in figure 2.8 of chapter 2. Figure 4.1 shows a simple scenario where connections with tuned delays enable the inputs to coincide.

4.2.5 Delay adaptation learning

The connection strength in network models represents the amount of conductance of a connection in biological neural networks. In biological systems, appropriate

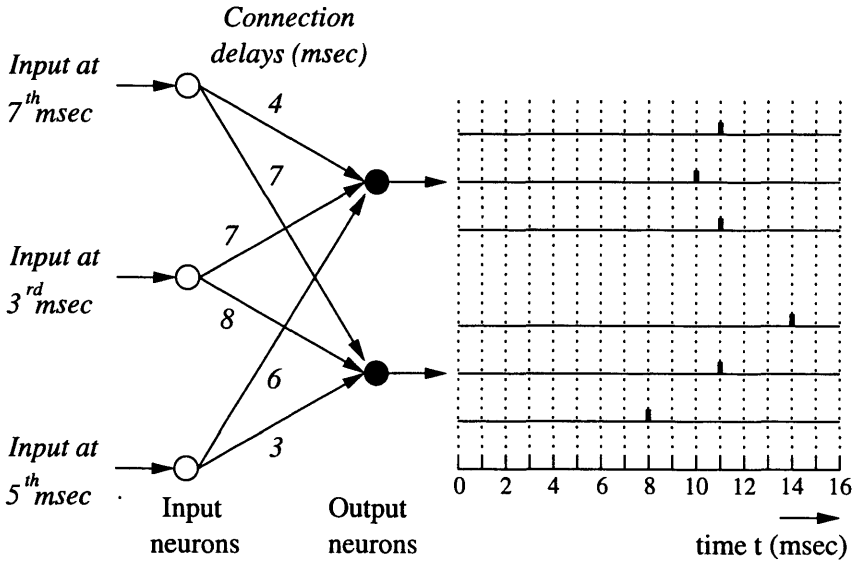


Figure 4.1. Pattern detection by spiking neural network. Right hand side of the figure shows how the differences in the timing of input spikes are compensated by the delays.

stimulation can induce changes in the transmission efficacy of the connections. These changes can affect the connections for a short time or for longer period, which are thought to be the factors responsible for learning and memory [Gerstner and Kistler, 2002]. Hence, most of the learning models proposed for biological networks encoded information in their connection strength [Bi and Poo, 1998; van Rossum et al., 2000]. Furthermore, new evidence available from the study of biological systems shows that the temporal delays in neuronal connections can also be adapted. Target localisation in neural systems requires precise neuronal signal integration, which can be achieved through delay adaptation [Eurich et al., 2000]. A study on optic nerves shows that the time delays in them are equalised [Stanford, 1987]. Innocenti et al. [Innocenti et al., 1994] found that the signals in visual collosal axons arrive simultaneously at all axonal endings. These findings indicate that the delays are adaptable and could be used as an alternative mechanism for learning in neural circuits.

In the context of neural learning, two methods using delay-based learning schemes can be found in the literature, namely, delay shifting and delay selection. In the delay shifting mechanism, the connection delays are adapted in such a way as to reflect the learning. The delay selection mechanism on the other hand, chooses a sub-set of connections with appropriate delays from a large set of connections. The selection process can be performed through altering the connection strengths, which enhances the connections with appropriate delays while pruning the others [Eurich et al., 2000]. The next section summarises in detail the work on delay adaptation learning reported in the literature.

4.3 Research on delay adaptation learning

This section summarises the previous work on delay adaptation learning, including both delay selection and delay shifting schemes. These range from biological modelling studies to real world applications. The models discussed here are not restricted to spiking neural networks. Some of the delay selection techniques reported so far have been used for applications such as clustering and classification. However, most of the delay shifting learning models reported so far for spike-based neural networks are for analytical studies. This section first describes the delay adaptation learning models utilised in biological modelling studies. Most of these learning models are unsupervised and rely on Hebbian-type learning rules. The work on classical neural networks with delays are reported next, which is followed by a detailed description of previous unsupervised delay adaptation learning in spike-based neural network models.

4.3.1 Delay-based modelling studies

Baldi and Atiya [Baldi and Atiya, 1994] performed a study to explore how delays affect neural dynamics and learning. The investigation was focused on the stability and convergence properties of the network with delays. Eurich et al. [Eurich et al., 1999] studied the dynamics of delay adaptation through self-organisation. The objective of their proposed model was to optimise the coincidence of the inputs through delay adaptation. This was achieved through a Hebbian-based delay selection learning rule which modifies the connection weights in order to select suitable delays, which will in turn enable the inputs to coincide at the post-synaptic neuron. The connections through which input spikes arrive simultaneously with the output spike are strengthened while the others remain unchanged or even weakened. Further, a delay shift scheme was also proposed,

where the delays were modified instead of the connection strengths. It was shown that stable solutions exist for arbitrary temporal inputs. The proposed training scheme was able to select the shortest possible set of delays which yields coincidence. Another delay adaptation learning rule was introduced by Tversky and Miikkulainen [Tversky and Miikkulainen, 2002] for the activity-dependent development of directionally selective cells in the primary visual cortex. This delay shift learning model adapts the delays in order to align the input spikes at the target neuron. A modelling study by Gerstner et al. [Gerstner et al., 1996], which selects appropriate delayed connections through weight adaptation, was described in sub-section 3.3.2 of chapter 3.

4.3.2 Learning models for conventional networks with delays

A significant amount of research was conducted on the classical neural network models with delays. In general, conventional neural networks are considered to be static. Since they have no internal time delays, they respond immediately to a particular static input. These networks can handle temporal inputs if the input is a delayed sample of the temporal pattern [Day and Davenport, 1993]. This technique, along with standard back-propagation learning, has been used in various applications [Lapedes and Farber, 1987; Widrow and Winter, 1988; Weigend et al., 1990]. A drawback with these static network models is that they cannot respond to the temporal patterns in the hidden layer neurons' outputs. Time Delay Neural Networks (TDNNs) were introduced by Waibel et al. [Waibel et al., 1989] to address this problem. In these networks, fixed integral multiples of unit delays were included in each layer of the network. This model exhibited the benefits of internal delays and was used for speech recognition tasks. However, this model encountered problems of spatial expansion and weight constraints. These were

overcome with a temporal back-propagation algorithm presented by Wan [Wan, 1990], where a model was proposed with fixed time delays and discrete-time representation of input signals. Later, this algorithm was generalised by Day and Camporese [Day and Camporese, 1991] in order to handle continuous-time signals and further generalised by Day and Davenport [Day and Davenport, 1993] to accommodate adaptable delays, which can be used to train a network for signal prediction.

4.3.3 Delay adaptation learning in SNNs

Hopfield is one of the pioneers who inspired the research on computing based on the timing of action potentials. In his classic paper, *"Pattern recognition computation using action potential timing for stimulus representation"*, the coincidence detection capability of a neuron was described and the potential of this new computational technique for pattern recognition was explained [Hopfield, 1995]. This model was claimed to be functioning more like a Radial Basis Function unit [Bohte et al., 2002b] rather than as a sigmoidal unit. The strategy behind this is that during the learning phase a particular neuron encodes a given input pattern with its connection delays. After learning when an input pattern which is closer to the stored pattern is presented, the connection delays compensate for the differences of the inputs and make the arrival of these inputs at the neuron coincide, enabling the neuron to fire [Natschläger and Ruf, 1998].

Based on the research reported in [Hopfield, 1995; Gerstner et al., 1996], Natschläger and Ruf [Natschläger and Ruf, 1998] performed a study on spatio-temporal pattern analysis specifically to cluster high dimensional data. Their work was explained clearly in sub-section 3.3.2 of chapter 3. This model's capability was improved through a population coding scheme by Bohte et al. [Bohte et al., 2002b]. A supervised error-back propagation learning was also proposed by

Bohte et al. [Bohte et al., 2002a] for the network model introduced in [Natschläger and Ruf, 1998]. Their model also incorporated the population coding scheme proposed in [Bohte et al., 2002b]. Here, the learning process performed a form of delay selection through weight adaptation. Improvements to this model were proposed in Schrauwen and VanCampenhout [Schrauwen and VanCampenhout, 2004], in which an error gradient-based delay adaptation method was introduced, but its applicability was not verified.

4.4 Proposed self-organising delay adaptation SNN for clustering

The self-organising delay adaptation spiking neural network (SODA_SNN) is introduced in this section. This temporal coding spiking neural network employs spiking neurons as coincidence detectors which can identify correct delay patterns corresponding to the input patterns. The proposed learning model organises a network of spiking neurons in a way similar to Kohonen's self-organising map. Rather than selecting appropriate delay values from a set of delays, the proposed learning model shifts connection delays using a learning rule based on the modelling study reported in [Tversky and Miikkulainen, 2002]. Clustering of data sets is considered as the target application and the model was successfully applied to the clustering of several data sets. In the following sub-sections the proposed model is introduced and discussed.

4.4.1 Network architecture

A two-layer network similar to Kohonen's SOM was chosen as the network architecture of the SODA_SNN. The input layer was assigned with a number of input neurons equal to the number of input parameters. The output layer was con-

structed with spiking neurons as defined in section 2.5. Here, the spiking neurons were realised as coincidence detectors. The neurons in the output layer were arranged in a two-dimensional rectangular grid. The input layer was fully connected to the output layer with feed-forward connections. A connection here is a single entity with no multiple sub-connections. Each connection was characterised with a weight and a delay value. The connection strengths can be represented with values in the range of $[0, 1)$. The connection delays can be assigned with values in the range of 0 to some predefined positive value depending on the other network parameters such as the input time window and the time constant. In the proposed model, the learning rule adapts the delays while keeping the connection strengths fixed.

4.4.2 Integration and coincidence detection with a spiking neuron

The spiking neurons in chapter 3 were realised as integrators. In this chapter the spiking neurons are implemented as coincidence detectors. The two modes of operation of the spiking neuron are realised based on the suggestion presented in [Konig et al., 1996]. In the proposed spiking neural network models, a neuron is activated for a period of time window defined as t_{window} over which the neuron's potential is summed up and an output spike is generated if it satisfies the threshold criterion. During this period, inputs are presented to the neurons within an input time window defined as (t_{input_window}).

The coding scheme proposed in this research considers only the timing of the first spike of a neuron. Hence, it is sufficient for a neuron to fire only once during a particular cycle of activity. Since the inputs are given to the network within an input time window, this time window can be considered as the inter-spike interval. According to the suggestions presented in [Konig et al., 1996], integration can be

achieved when the activation period is greater than the inter-spike interval and coincidence detection is achieved when it is less than the inter-spike interval. But for the spiking neural network concerned, if the activation period is brought down below the input time window, then the effect of some of the inputs would be missed. However, the effective summing-up period depends on the activity period of the input spikes. Hence, instead of varying the activation period, the period of activity of an input spike can be varied to realise both modes of operation of the spiking neurons. Since the time constant τ of the activation function defines the period of activity of an input spike, the two modes of operation can be realised effectively by varying the τ relative to the input time window.

When the value chosen for τ is equal or greater than the t_{input_window}, the effect of the spikes will span over this window. Therefore, the effective summing period would be greater than t_{input_window}. This causes the spiking neuron to operate as an integrator. On the other hand, when τ is chosen to be shorter than t_{input_window}, the effective period of each spike will be shorter. Hence, the effective summing period will be shorter than t_{input_window} and the spiking neuron will operate as a coincidence detector.

4.4.3 Spike time-based delay adaptation learning

A Hebbian-type spike-timing based learning rule is employed for training the network. The learning rule encodes the input information in the connection delays such that the delayed inputs coincide at some neuron. The learning rule is based on the timing of the input and output spikes. In typical neuron activity, if an input spike arrives at a neuron through a delayed connection before there is an output spike, then the delay associated with that connection is increased. On the other hand, if the input spike arrives after the generation of an output spike, then the connection delay is decreased. In both cases the amount of modification

is determined by the difference between the time when an input spike arrives and the time when an output spike is generated [Eurich et al., 1999; Tversky and Miikkulainen, 2002]. The learning equation is based on the modelling study as reported in [Tversky and Miikkulainen, 2002], which analysed the activity-dependent development of directional selective cells in the primary visual cortex. Equations 4.1 and 4.2 specify the learning rule which is graphically described in figure 4.2.

$$g(\delta t) = \delta t \left(\frac{e^{-\delta t^2/\tau_{stdp}^2}}{\tau_{stdp}} \right) - b \tag{4.1}$$

$$\delta t = t_j^{'} - (t_i + d_{ji}) - s \tag{4.2}$$

where δt is the difference between the timings of the post- and pre-synaptic firing events and τ_{stdp} is the time constant for depression and potentiation; b is a positive bias value. t_i is the timing of an input to neuron j from neuron i, $t_j^{'}$ is the time of the output spike. d_{ji} is the delay of the connection between the neurons i and j. s is a positive value which shifts the learning rule to the right along the axis.

4.4.4 Self-organisation

Self-organisation in the SODA_SNN is achieved through competition and cooperation. The competition is created based on the firing times of the output neurons. In a practical situation, more than one neuron will fire for an input pattern. A highly activated neuron will tend to fire earlier [Thorpe et al., 2001]. The neuron that fires first (the winning neuron) will have a high degree of coincidence among its inputs, thus having the most accurate delay pattern to compensate for differences in input timings. Therefore, the winning neuron is more likely to represent the input pattern [Natschläger and Ruf, 1998; Bohte et al., 2002b,a].

Cooperation among the neurons is achieved through lateral excitation. Neurons which are closer to the winner neuron are given precedence over the other

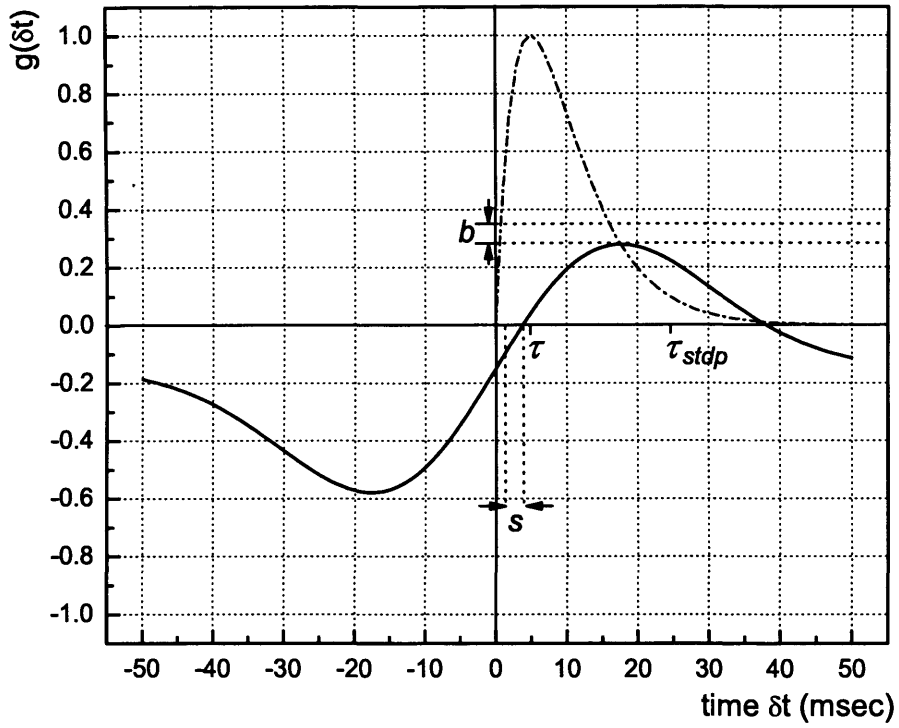


Figure 4.2. Learning rule for delay adaptation. Spike response function is shown in dotted lines for ease of comparison. Here τ, τ_{stdp}, b and s are assigned with values 5, 25, 0.1 and 2 respectively.

neurons. Lateral excitation is realised through the physical location of a neuron relative to the winning neuron as in the Kohonen's SOM [Haykin, 1999]. During the training process, connections of the neurons which are laterally closer to the winner neuron are updated proportional to its distance from the winner neuron. A simple strategy given by equation 4.3 which specifies the lateral effect on neuron j relative to neuron k is incorporated into the learning rule.

$$h(j,k) = e^{-(d(j,k))^2/2\sigma} \tag{4.3}$$

where $d(j,k)$ is the lateral distance between the neurons j and k in the output lattice. The parameter σ is the maximum width of lateral excitation which could be reduced with training epochs from a wide area of interest to a single neuron.

4.4.5 Stabilising the delay adaptation

Stabilising the learning is achieved through two measures. A stability measure which is local to a particular connection is incorporated in the spike time-based rule given in equation 4.1. Modification of a connection delay could be stabilised through properly assigning values for τ, τ_{stdp}, b, and s in consideration with the selected input time window. Better results can be achieved when the values for these parameters are assigned such that the learning rule given by equation 4.1 reaches its maximum when the spike response function begins to saturate as shown in figure 4.2. Here, τ is assigned a value such that the period of the spike activity is a fraction of the input time window in order to ensure the operation of the spiking neuron as a coincidence detector. A value for τ_{stdp} is selected such that the learning rule could be applicable over the full activity period of the spike. Bias b and shift s play an important role in stabilising the delay adaptation. The bias restricts the expansion of the delays when the difference between the timing of the input spike and the time of generation of the output spike exceeds a finite value. In fact, the delay change is made negative if the time difference exceeds

this limit. The shift s restricts the continued increase of tuned delays. If properly tuned, a set of connections will enable inputs to reach a target neuron just before the firing of that neuron. Due to the shift, the connections' delay through which spikes reach a neuron just before firing will not be increased. If not controlled by local stability measures the delays will increase indefinitely.

The second measure considers a neuron as a whole entity and stabilises the delay adaptation based on the amount of coincidence of the inputs to that neuron. A simple linear measure given by equation 4.4 is utilised, which is based on the time between the first and last delayed input spikes to the neuron.

$$f(d_{ji}) = \frac{max\left[t_i + d_{ji}\right] - min\left[t_i + d_{ji}\right]}{t_{input\_window}} \quad i = 1, 2, ..., n \tag{4.4}$$

where t_i is the time of an input spike to neuron j through input neuron i. d_{ji} is the delay of the connection between neuron i and j. t_{input_window} is the input time window. n is the number of input neurons. If the delays had been tuned to enable the input spikes to coincide then further modification is not necessary. Hence, the amount of change for a particular connection delay can be decreased while achieving the same degree of coincidence. Therefore, if the total change of connection delays falls below a minimum value or the change remains constant, then the training process can be terminated.

4.4.6 Learning rule

The proposed learning rule is based on the three main aspects, namely spike-time based learning, self-organisation and stabilisation, discussed in sections 4.4.3, 4.4.4 and 4.4.5 respectively. The learning rule is given by equation 4.5, which specifies the amount of change to the connection delay from neuron i to j.

$$\delta d_{ji} = \eta\ g(\delta t)\ h(v, j)\ f(d_{ji}) \tag{4.5}$$

where g, h and f are defined by equations 4.1, 4.3 and 4.4 respectively. δd_{ji} is the amount of change for the connection delay from neuron i to neuron j; η is the learning rate; v is the winner neuron, which fires first among the competing neurons; δt is the time difference as defined in equation 4.2.

4.4.7 Interpreting the output and cluster identification

The output of the SNN is the timing of the spikes generated by its output layer neurons. Since the neurons function as coincidence detectors, a neuron which fires first contains a delay pattern which is the closest match for an input pattern in terms of coincidence. In a similar fashion to the self-organising weight adaptation SNN, it was observed in this model that similar input patterns tend to excite a particular output neuron to fire within some well defined time interval. Hence, the mapping of the output is based on the position of the winner neuron in the output grid and their firing times as described in chapter 3 (sub-section 3.4.7). Here, a cluster is specified with a particular set of neurons and a particular firing time interval.

4.5 Implementation details

The SODA_SNN was realised as a discrete model, described in chapter 2 of section 2.5.4 and implemented in software. A separate computer program was written for this purpose and used for the analysis of the model. The spiking neurons were realised as coincidence detectors by assigning a relatively low value for the time constant of the spike response function, with respect to the input time window as described in sub-section 4.4.2.

Setting the values and range for the connection weights and delays also played an important role in the operation of the model. In addition, the threshold

values were set appropriately in order to obtain better performance from the network. The selection of these parameters was carefully made in order to detect the coincidence as accurately as possible and to give adequate space to reflect the cluster distribution. The optimum values were found effectively by trial and error. This optimisation process was enhanced by analysing the inputs, and in particular the distribution and range of values for each input feature. In addition to setting the parameter values, the implementation of the learning process encountered some interesting and critical issues which are discussed in detail in the following sub-sections.

4.5.1 Initialising the connection weights and delays

The structure of the SODA_SNN is similar to that of a Kohonen's SOM except that the computing neurons are coincidence detecting spiking neurons. The output neurons were placed in a two-dimensional rectangular grid. The input and output layers were fully connected through feed-forward connections and the connections were characterised by a weight and a delay value. The values for the connection weights were assigned randomly within a small predefined range in order to prevent the connection weights dominating the effect of the connection delays. A suitable range for the weights was found to be between 0.4 and 0.6.

The delays were assigned with random values closer to the middle of the input time window. An input time window with 30 unit intervals was found to be adequate to represent the input vectors temporally in most cases, and the delays were assigned within the range of 10 to 20. The delays could also be assigned using a Gaussian distribution, where the mean and the standard deviation of the distribution can be selected to reflect the distribution of temporally coded inputs. The main concern here is to ensure an initial evenly distributed representation of the input vectors over the output space. In addition, the initial distribution

should give adequate space for the adaptation of delays in order to achieve better coincidence detection and thus better clustering. Due to the nature of the learning strategy, it is possible for a connection to have a delay value equal to the input time window. This is to accommodate the worst case scenario where the first input at the beginning of the input time window is followed by another input at the end of this time window. Hence, the activation window was selected to be greater than twice the duration of the input time window.

4.5.2 Setting the threshold value

The threshold value is a critical factor for the learning process. Initially in the untrained network, it is unlikely that any neuron's delayed connections are able to make all the inputs to coincide. In order to achieve high accuracy in coincidence detection, it is necessary to keep the threshold value of the neurons high. However, this will prevent most of the neurons from firing at the early stage of learning. As a result, these neurons will remain dead throughout the learning process. It is a necessity for a neuron to be active in order to be trained, since the learning rule is based on the input and output spike times. Even if some neurons managed to fire due to the random assignment of delay values, there is a high possibility that these neurons could be surrounded by dead neurons. This will prevent cooperation among neurons and will result in a poorly distributed output space. As a result the training will produce many dispersed and overlapped clusters. This problem could be solved effectively if an adequate number of output neurons can be kept active throughout the learning process. This can be easily achieved by starting with a low threshold value and later increasing it by a fraction with each epoch of learning. It was found experimentally that by adopting this method the model learned the input information effectively and formed clear clusters.

Initially a low threshold value was selected to enable most of the neurons to

fire for almost all of the input vectors. This value was increased gradually with training to a value which could detect around 75% to 85% of the effect of each input. A suitable maximum value for the threshold was found to be *number of inputs* x 0.5 x 0.8. Here 0.5 is the average weight and the value of 0.8 reflects the 80% effect of an input.

4.5.3 Setting the parameters of the learning rule

Setting the learning equation parameters at their optimum level is the key to obtaining better results. In general, the parameters were assigned with values which cause the spike response function and the learning equation approximately to overlap as shown in figure 4.2. The exact values for the time constant τ and τ_{stdp} were determined by trial and error. The shifting of the learning equation to the right was performed appropriately in order to minimise the delay change for coinciding inputs. A shifting value, which is a fraction of the time constant for the spike response function, was found to be suitable. The value for bias is another factor which affects the learning process. A low bias value of around 0.1 was found to be adequate. In addition, the learning rate is also an important parameter. Better stability was achieved in learning when the learning rate was set in the range of 0.1 to 0.5.

4.6 Simulation results and discussion

The SODA_SNN was analysed through a number of simulations in order to determine its clustering capability, activity, coincidence accuracy and stability. The analyses were performed using the three bench marking data sets and the high dimensional control chart data which were described in chapter 2 of section 2.6.2. Table 2.1 summarises the details of these data sets. For the simulations, approx-

imately 66% of the data set was randomly selected and used for training and the remaining set was used for testing. Description of the number of samples used for training and testing is given in table 3.1. The simulation results were used to investigate the effectiveness of the learning model and of the strategies incorporated in the learning model. The following sub-sections discuss these issues in detail.

4.6.1 Clustering capability

The SODA_SNN was applied to each data set specified in section 2.6.2 and the clusters were identified based on the location of the winner neurons and their firing time. Clustering accuracy for the training set was found using the class information of the training samples. Validity of the clusters was verified with the test set which was not used for the training. After the training process was completed, the test set was presented to the network and the clusters within the input data were identified. The clustering accuracy of the test set was found using the class information of the testing samples. For each data set the training and the testing process was repeated five times and the average clustering accuracy was computed. The network's connection delays and weights were reassigned in each trial. Table 4.1 lists the optimum clustering accuracy obtained by SODA_SNN for each data set. The data used for training and testing is given in Appendix B.

The results show that the SODA_SNN achieved better clustering accuracy with a relatively small network and fewer training epochs. In addition, the model obtained high accuracy for the test sets, which confirms the validity of the clusters and the generalisation capability of the model. The SODA_SNN obtained better clustering accuracies than the weight adaptation model SOWA_SNN for Iris, Wine and Control chart data. For Cancer data, SODA_SNN obtained marginally lower clustering accuracy than SOWA_SNN. Further, SODA_SNN was found to

be utilising the network efficiently to form well separated clear clusters. Clusters formed using the SODA_SNN are given in the next sub-section. Hence it can be considered that the SODA_SNN is a viable tool for clustering applications and the coincidence detection feature of the spiking neuron enables the SODA_SNN to perform better than the SOWA_SNN.

As discussed earlier in section 4.5, the values for the network parameters affect the clustering capability of the model. The parameter values of the network for clustering different data sets were found by analysing the data set and assigned by trial and error. Table 4.2 lists these parameter values. A number of trial runs were performed to find the near optimum values for each parameter. It is notable here that the values for the parameters τ, τ_{stdp}, s and b are critical for optimum performance. They were chosen along with the learning rate in order to achieve maximum clustering accuracy while keeping the delay values to a minimum.

Further experiments were performed to determine the effect of the network size on the clustering capability. Here, the network size is specified by the size of the output grid. Data sets were clustered with different sized networks and their average clustering accuracies were found over five trials. The number of training and testing samples, number of training epochs and the values for the network parameters, were kept the same as those of the corresponding SNNs which obtained the optimum results. Results obtained for the Iris, Cancer and Wine data are given in table 4.3. Results for the Control chart data are given in table 4.4. These results show that when the network size was reduced from its optimum, the clusters were found to be overlapping and resulted in reduced clustering accuracy. When the network size was increased from its optimum, the clustering accuracy was also found to be reducing. Here several sub clusters were identified, which were scattered over the output space and the network was underutilised, resulting in low clustering accuracy.

Data set	No. of neurons		No. of training epochs	Clustering accuracy (%)	
	Input	Output		Training	Testing
Iris	4	5x5	20	96.9±0.8	96.9±1.2
Cancer	9	3x3	10	97.4±0.3	96.8±0.2
Wine	13	5x5	20	95.3±0.7	95.2±0.8
Control chart	60	8x8	20	96.4±0.3	96.1±0.4

Table 4.1. Average clustering accuracy obtained for the SODA_SNN.

Data set	t_{input_window}	τ	τ_{stdp}	b	s	η	θ
Iris	30	5	25	0.05	2	0.1	0.7~0.8
Cancer	30	5	25	0.05	2	0.3	0.6~0.8
Wine	30	5	25	0.1	2	0.3	0.6~0.67
Control chart	30	5	25	0.1	2	0.4	0.7~0.83

Table 4.2. Values assigned for the parameters of the network used to obtain the results listed in table 4.1.

Network size	Clustering accuracy(%)					
	Iris		Cancer		Wine	
	training	testing	training	testing	training	testing
3x3	88.6±1.0	87.4±1.3	97.4±0.3	96.8±0.2	88.8±2.8	87.4±4.0
4x4	89.2±2.0	89.6±3.4	96.3±0.7	95.6±1.0	92.0±2.7	93.1±2.4
5x5	96.9±0.8	96.9±1.2	96.5±0.9	96.5±0.6	95.3±0.7	95.2±0.8
6x6	93.7±2.8	91.1±2.7	96.3±1.1	94.9±2.6	90.5±1.9	91.7±4.3
7x7	93.2±2.7	91.6±2.4	96.1±1.4	93.4±3.3	90.1±2.2	90.1±4.9

Table 4.3. Average clustering accuracy obtained for different size of SODA_SNNs.

Network size	Clustering accuracy(%)	
	Control chart	
	training	testing
6x6	93.5±2.5	93.6±3.5
7x7	93.1±0.2	92.7±1.5
8x8	96.4±0.3	96.1±0.4
9x9	92.3±1.6	93.9±0.6
10x10	91.8±0.7	92.5±0.8

Table 4.4. Average clustering accuracy obtained on control chart data for different size of SODA_SNNs.

The approach taken in the proposed delay adaptation model is novel for learning in spiking neural networks. The model proposed in [Bohte et al., 2002b] obtained a clustering accuracy of $92.6 \pm 0.9\%$ for the Iris data set, where the inputs were encoded with 4x8 input neurons through a population coding scheme. Each connection was composed of several sub-connections, thus increasing the size of the network even more. For the Iris data set, the clustering accuracy obtained for the proposed model is greater than for the results presented in [Bohte et al., 2002b]. In addition, the size of the network was much smaller without any additional sub-connections. The model proposed in [Bohte et al., 2002b] required a population coding scheme to obtain better clustering accuracy, which increased the number of input parameters, thus increasing the required computational effort. The SODA_SNN utilises a linear temporal coding scheme which represents an input value through a single spike time. In addition, the clustering accuracy of the SODA_SNN on other data sets, including the high dimensional Control chart data, was found to be high. These results show that the clustering capability of SODA_SNN can be considered better than the previous unsupervised SNN models.

For comparing the capability of the proposed model with conventional network models, the same data sets listed in table 2.1 were clustered using Kohonen's SOM. For this purpose the freeware program ESOM Analyser was utilised. The data sets were clustered using this program and the results obtained are given in table 4.5. The size of the network, number of training samples and training epochs were kept same as used for the corresponding SNN model. The proposed model achieved better clustering accuracies on all the data sets than the SOM in this configuration.

The SOMs specified above were further trained until they reached their highest clustering accuracy. The results obtained are listed in table 4.6. For Iris, Cancer

and Control chart data sets, the SODA_SNN obtained slightly higher clustering accuracy on both training and testing sets. SODA_SNN obtained much better results than the SOM on the Wine data set. For all the data sets SODA_SNN required fewer training epochs than SOM to reach the highest accuracy. These results show that the performance of the proposed model is better than the Kohonen's SOM.

4.6.2 Network activity

The activity of the SODA_SNN was examined in order to investigate the effectiveness of the dynamic threshold strategy introduced in sub-section 4.5.2. The investigation was performed on four different SNNs trained on the four data sets concerned. For analysing the proposed model with the Iris data set, a network with 4 input neurons and 5x5 output neurons was constructed. The network parameters were assigned with values as described in the section 4.5. Initial threshold value was set to detect 70% effect of each input and further increased to 80% linearly with each training epoch. The data samples were temporally coded and the network was trained with the temporal data for a total of 20 epochs. During the training phase, the lateral excitation was initially set to realise within a neighbourhood spanning the whole network and was reduced gradually with each epoch of training to a single neuron. After each training epoch, the whole training set was presented to the network and the average number of active neurons was found. Similarly, SNNs were constructed for analysing the other three data sets and the above mentioned procedure was followed. Details regarding the four SNNs analysed are given in table 4.7. The network size and other parameters are the same as used to obtain the optimum results of the SODA_SNN in sub-section 4.6.1.

In order to compare the effect of learning with the dynamic threshold, net-

Data set	No. of neurons		No. of training epochs	Clustering accuracy(%)	
	Input	Output		Training	Testing
Iris	4	5x5	20	95.0±1.4	93.3±2.7
Cancer	9	3x3	10	96.5±0.4	96.1±0.2
Wine	13	5x5	20	75.3±2.7	73.0±3.5
Control chart	60	8x8	20	94.8±0.5	94.4±0.9

Table 4.5. Average clustering accuracy obtained for Kohonen's SOM. The software package ESOM was utilised for this purpose.

Data set	No. of neurons		No. of training epochs	Clustering accuracy(%)	
	Input	Output		Training	Testing
Iris	4	5x5	30	96.8±0.5	96.4±1.2
Cancer	9	3x3	20	97.3±0.3	96.3±0.9
Wine	13	5x5	30	78.7±0.5	78.3±0.9
Control chart	60	8x8	30	96.1±0.9	95.5±0.9

Table 4.6. Average highest clustering accuracy obtained for Kohonen's SOM.

Data set	No. of neurons		No. of training epochs	Threshold	
	Input	Output		Initial	Final
Iris	4	5x5	20	4x0.5x0.7	4x0.5x0.8
Cancer	9	3x3	10	9x0.5x0.6	9x0.5x0.8
Wine	13	5x5	20	13x0.5x0.6	13x0.5x0.67
Control chart	60	8x8	20	60x0.5x0.7	60x0.5x0.83

Table 4.7. Details of the SODA_SNNs for the analysis of network activity.

works with the same topology specified in table 4.7 were trained using a fixed threshold equal to the maximum value assigned in the case of dynamic threshold. Figure 4.3 shows the average number of active neurons after each epoch in both networks trained on the Iris data set. Likewise, figures 4.4, 4.5 and 4.6 show the network activity observed in both cases of the networks trained on Cancer, Wine and Control chart data respectively. It can be seen from the figures that, for the networks with the dynamic threshold, the average number of active neurons were high at the beginning and they were reduced with the increase of threshold over training. However, for the networks with the fixed threshold, the average number of active neurons remained approximately at the same level, with only a slight increase. Since the number of active neurons was high for the networks with dynamic threshold, they formed clear and finely distributed clusters, while the networks with fixed threshold formed several poorly distributed sub-clusters. Figures 4.7, 4.8, 4.9 and 4.10 show the clusters formed in both cases within Iris, Cancer, Wine and Control chart data respectively. These results show that the proposed learning method with dynamic threshold effectively kept the network with an adequate level of activity and enabled formation of clear and finely distributed clusters. The drop in the number of active neurons for the network with dynamic threshold is due to the restriction imposed on learning within a small neighbourhood. Another reason is that after learning, a single or group of neurons will fire only for input vectors belonging to a particular class.

4.6.3 Degree of coincidence

The measure given in equation 4.4 was employed to specify the degree of coincidence among the delayed inputs. This measure utilises the difference between the arrival times of the first and the last input spike to a neuron. This section analyses the degree of coincidence achieved through learning. For this purpose,

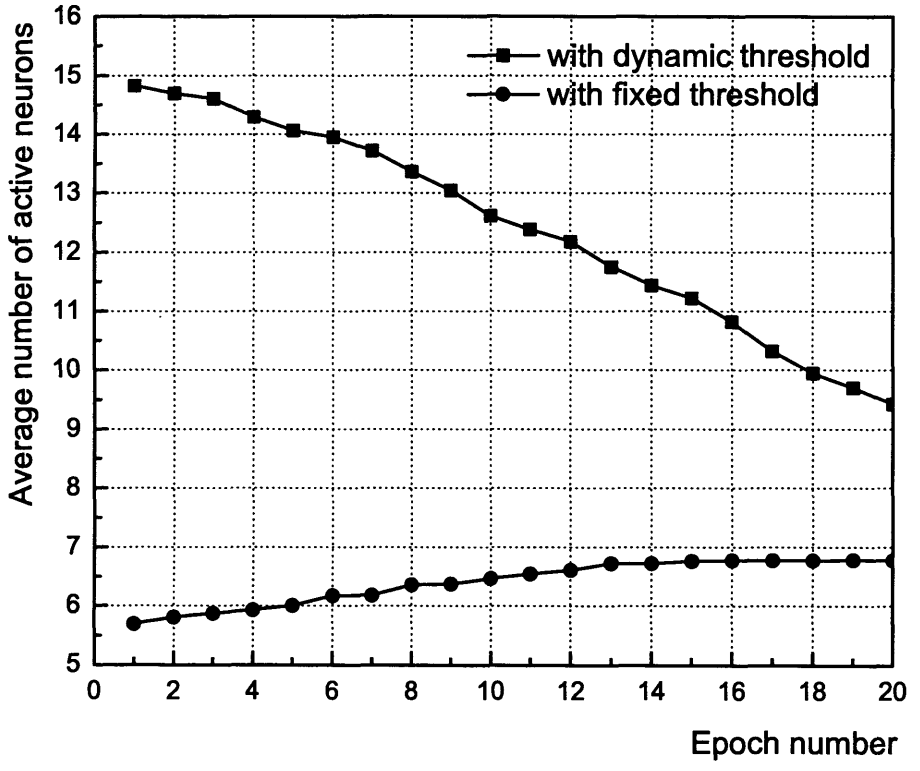


Figure 4.3. Activity of the SODA_SNN trained on Iris data with dynamic and fixed threshold values.

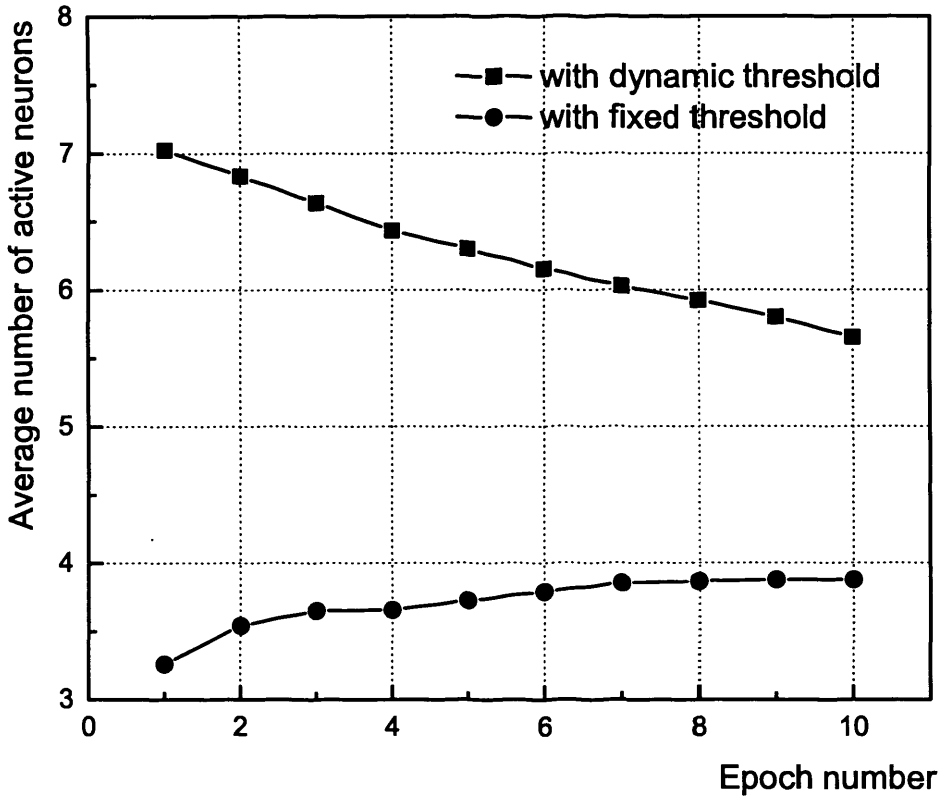


Figure 4.4. Activity of the SODA_SNN trained on Cancer data with dynamic and fixed threshold values.

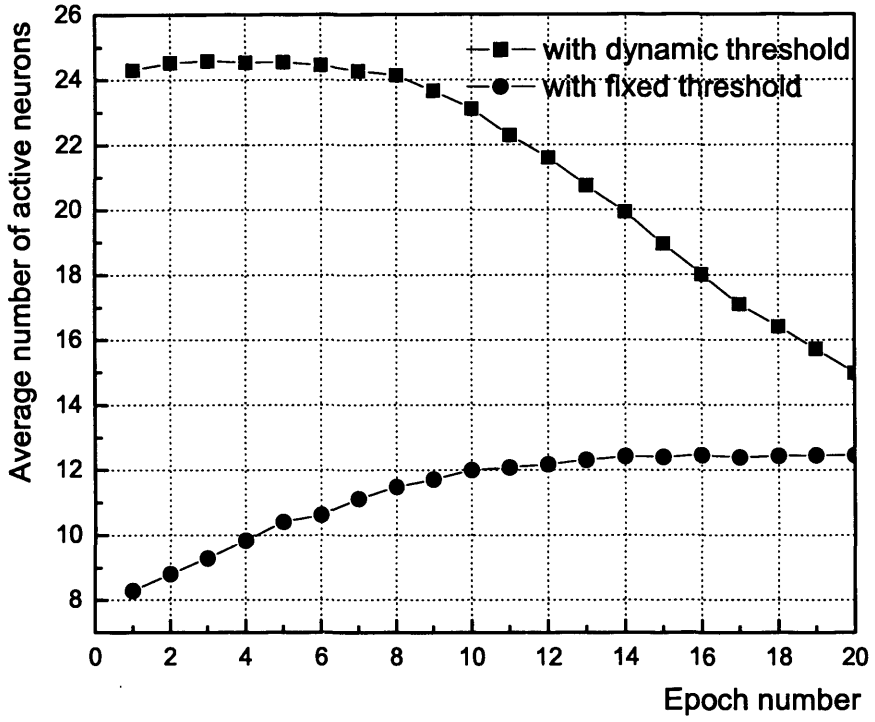


Figure 4.5. Activity of the SODA_SNN trained on Wine data with dynamic and fixed threshold values.

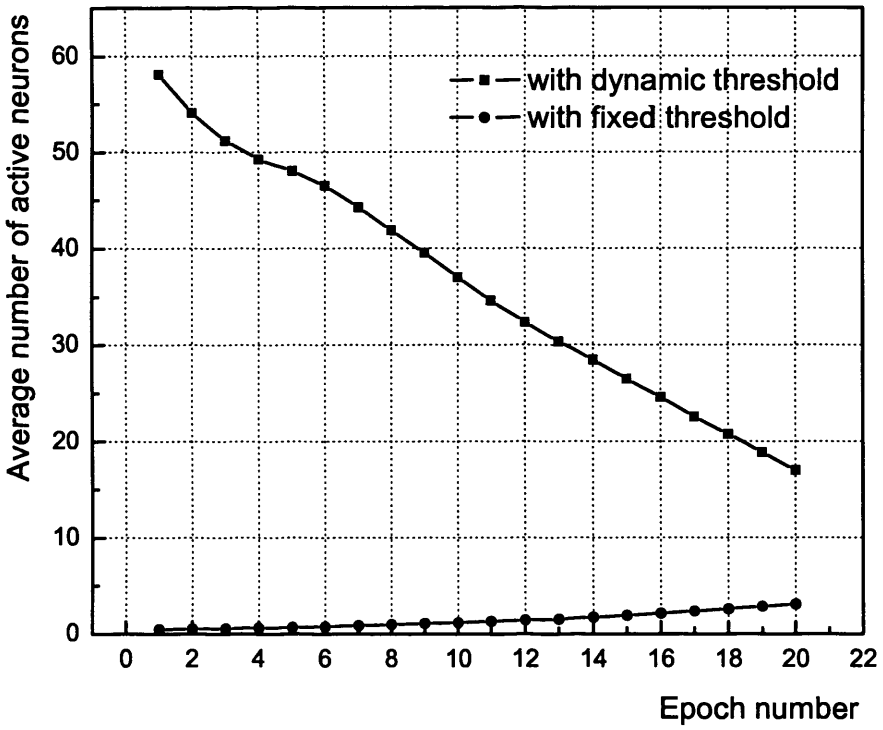
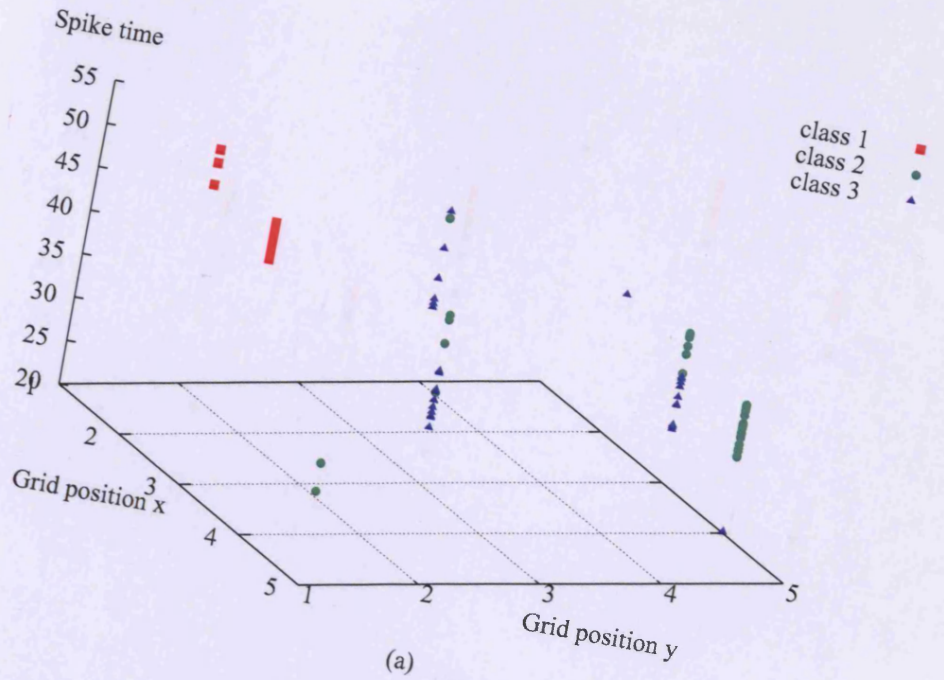
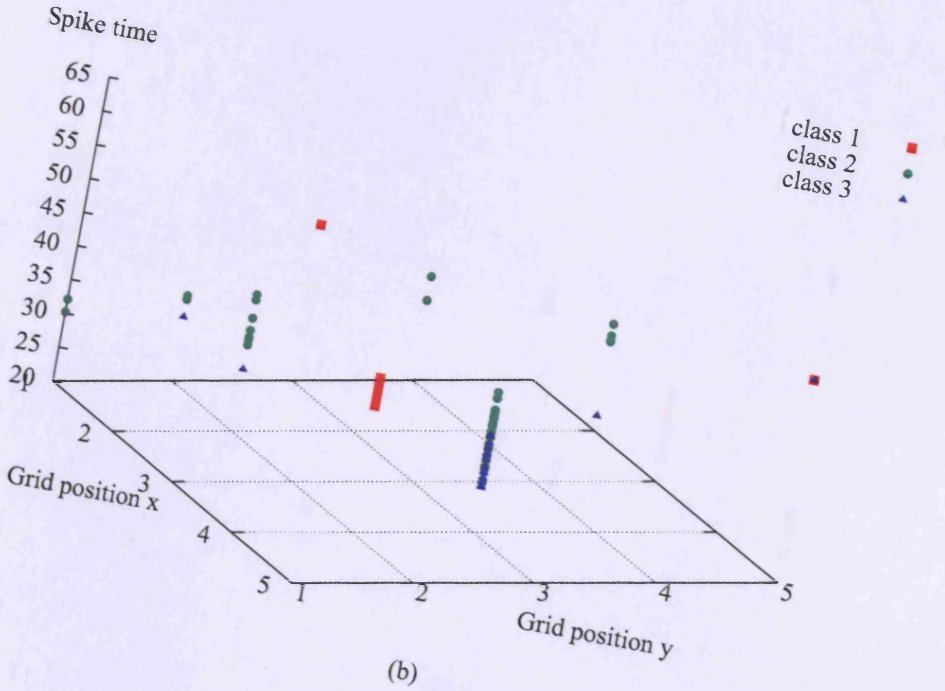


Figure 4.6. Activity of the SODA_SNN trained on Control chart data with dynamic and fixed threshold values.

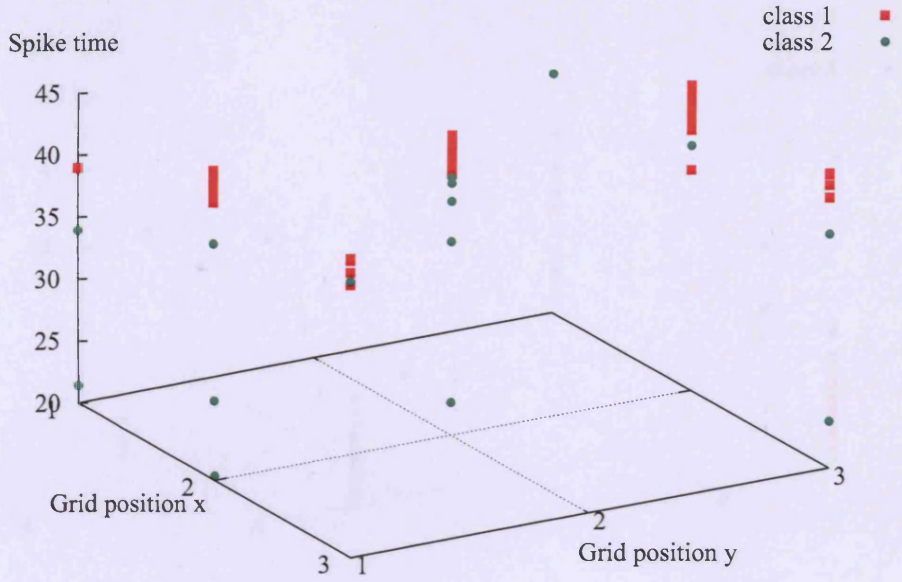


(a)

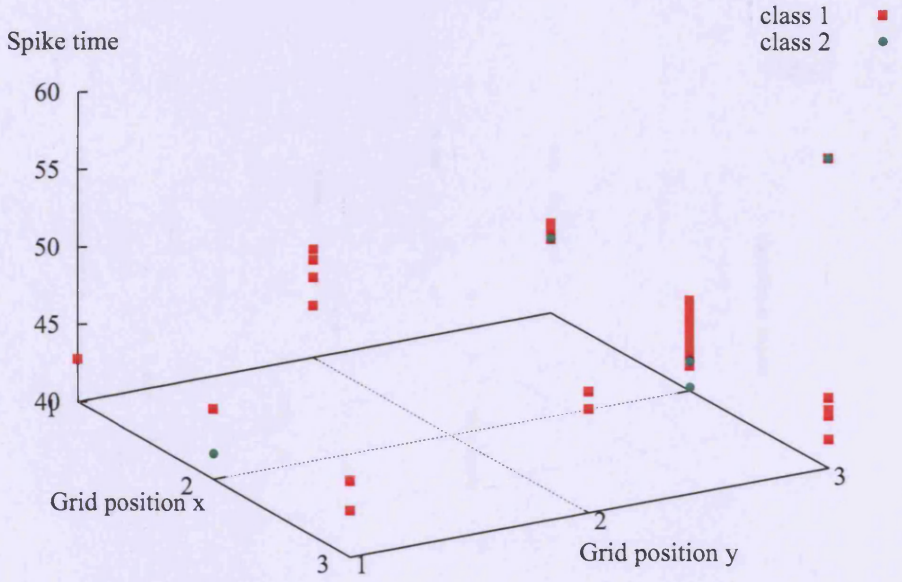


(b)

Figure 4.7. Clusters formed within the Iris data using SODA_SNN (a) with dynamic and (b) with fixed threshold.



(a)



(b)

Figure 4.8. Clusters formed within the Cancer data using SODA_SNN (a) with dynamic and (b) with fixed threshold.

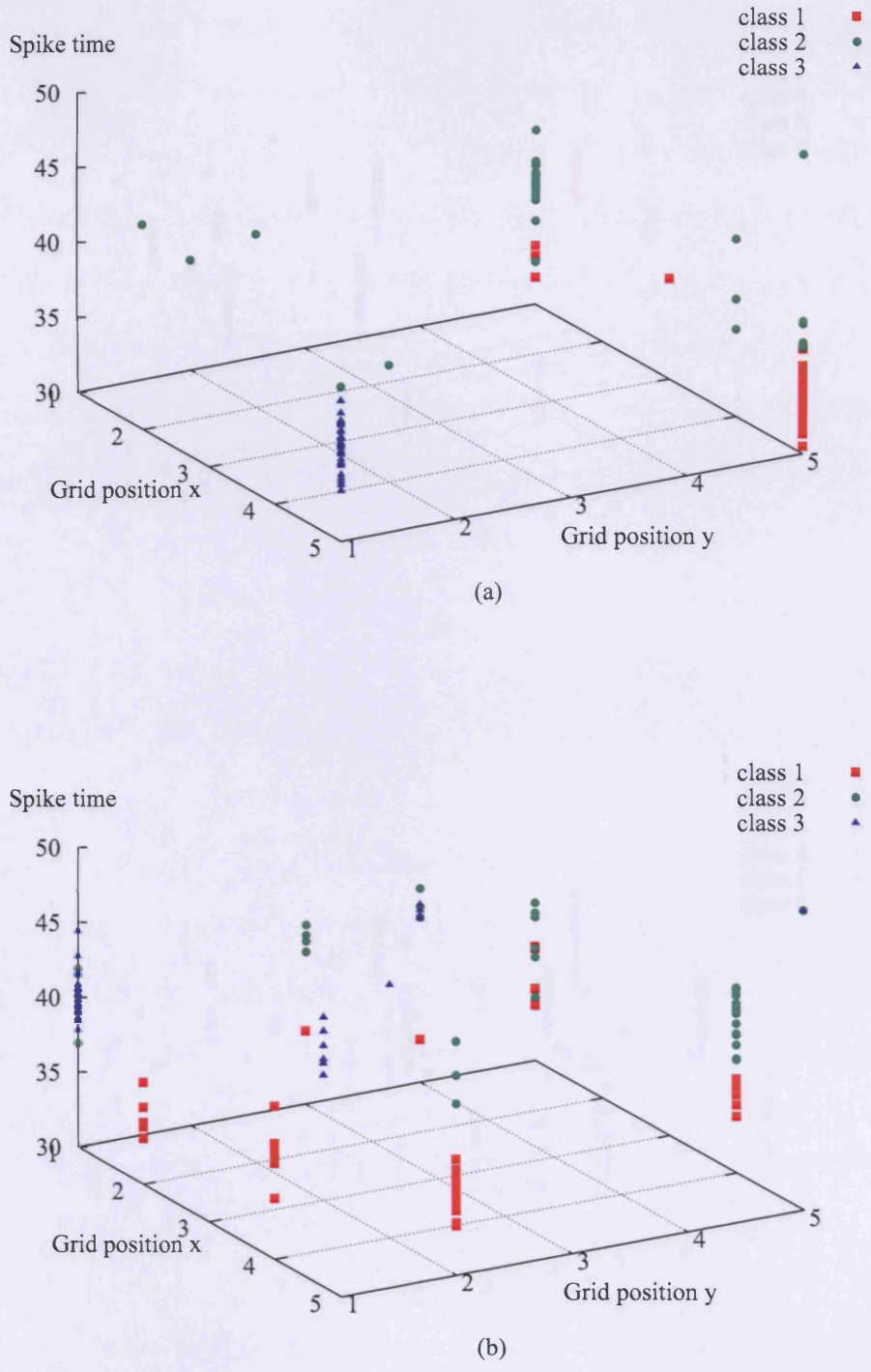
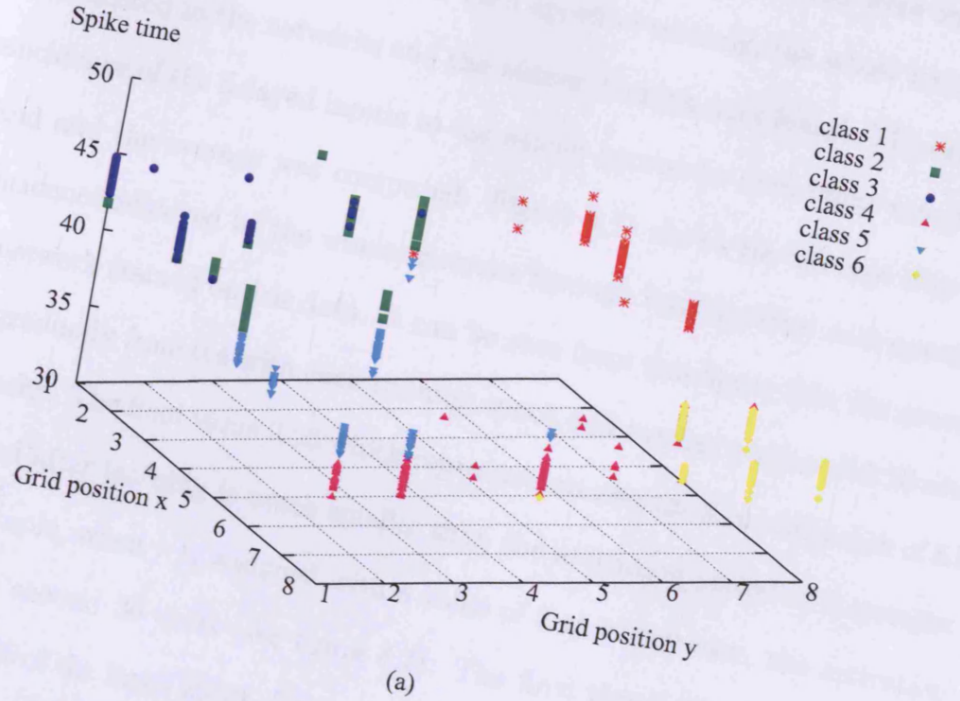
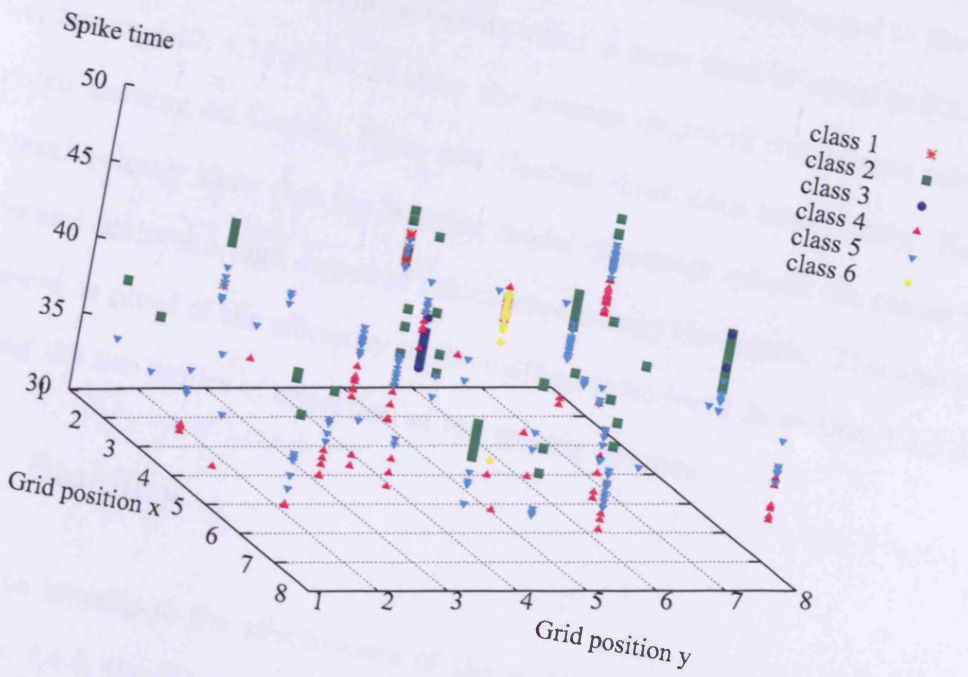


Figure 4.9. Clusters formed within the Wine data using SODA_SNN (a) with dynamic and (b) with fixed threshold.



(a)



(b)

Figure 4.10. Clusters formed within the Control chart data using SODA_SNN (a) with dynamic and (b) with fixed threshold.

networks similar to those described in the previous sub-section were trained on the corresponding data sets. After each epoch of training, the whole training set was presented to the networks and the winner neurons were found. The degree of coincidence of the delayed inputs to the winner neuron for each input sample was found and the average was computed. Figure 4.11 shows the average degree of coincidence achieved by the winner neurons through learning after each epoch on the network trained on Iris data. It can be seen from this figure that the average falls gradually from 0.4 with each training epoch and reaches a value of 0.23 after 20 epochs. The final value 0.23 which represents an average time difference of 6.9 obtained after learning is much smaller than the activation period of a synapse. For example, when τ is assigned with a value of 5 as in this case, the activation period is around 30 units (see figure 4.2). The final threshold value was set to detect 80% of the input effect. The value 6.9 is approximately equal to the period of the spike response function, where its effect is more than or equal to 0.8. Likewise, figures 4.12, 4.13 and 4.14 show the average degree of coincidence achieved through learning on Cancer, Wine and Control chart data respectively. Hence, the results clearly show that the learning model effectively adapts the connection delays and achieves a high degree of coincidence among the inputs. This also can be viewed as proof of the efficiency of the method introduced in section 4.4.2 for realising the two modes of operation of the spiking neurons.

4.6.4 Stability

In order to investigate the effectiveness of the stability measures introduced in sub-section 4.4.5, the delay distribution of the trained SNNs which obtained the optimum results in sub-section 4.6.1 were analysed. Figures 4.15, 4.16, 4.17 and 4.18 show the distribution of delays of the SODA_SNNs trained on Iris, Cancer, Wine and Control chart data respectively. It is notable here that initially the

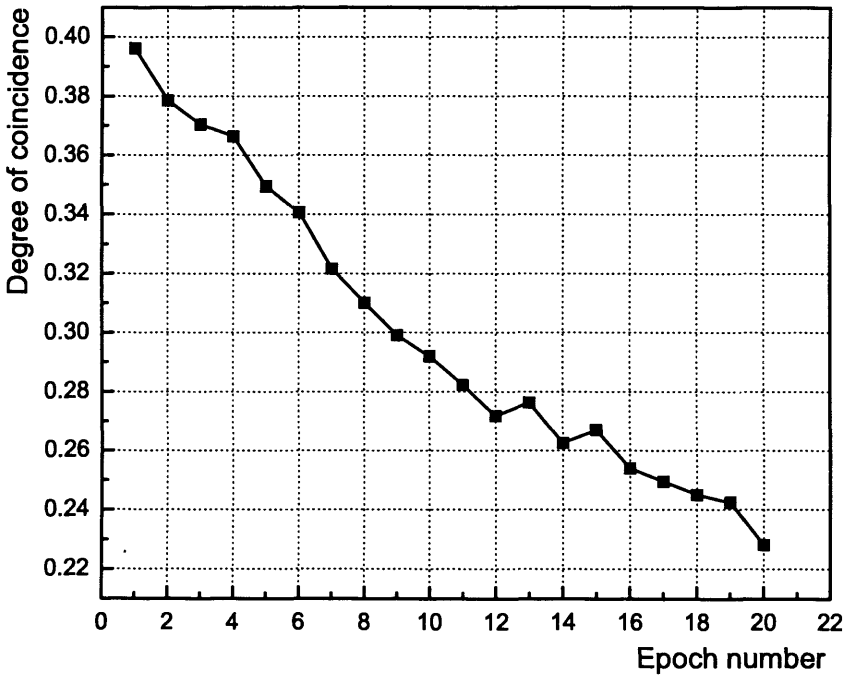


Figure 4.11. Degree of coincidence achieved on the SODA_SNN trained on Iris data.

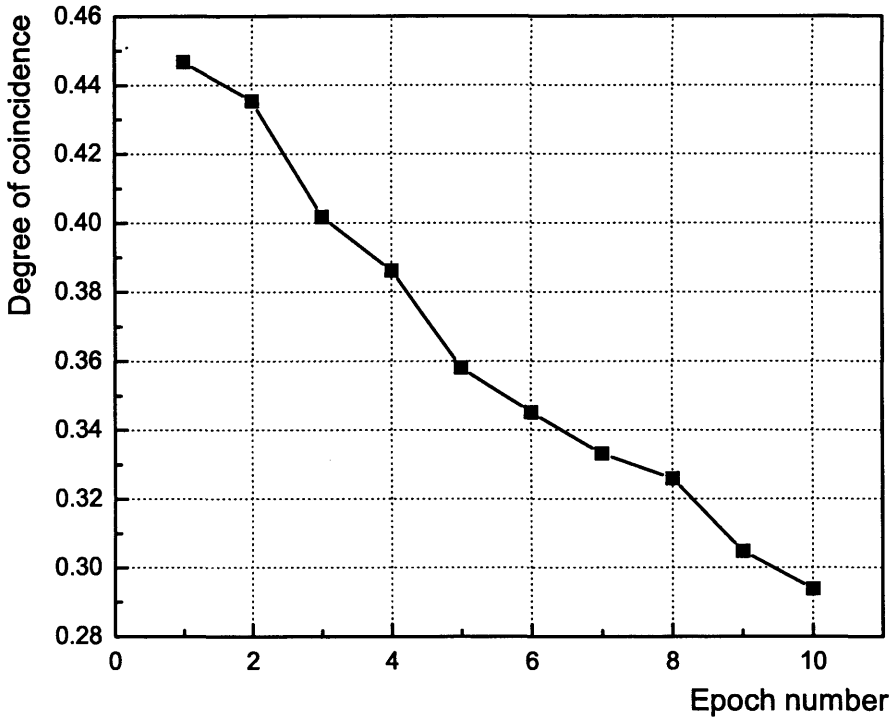


Figure 4.12. Degree of coincidence achieved on the SODA_SNN trained on Cancer data.

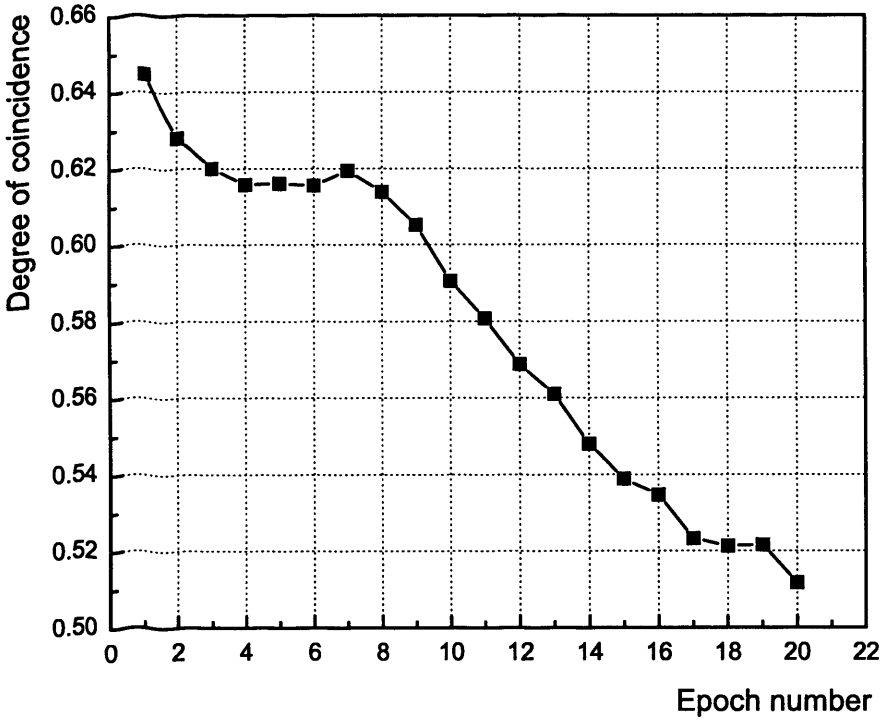


Figure 4.13. Degree of coincidence achieved on the SODA_SNN trained on Wine data.

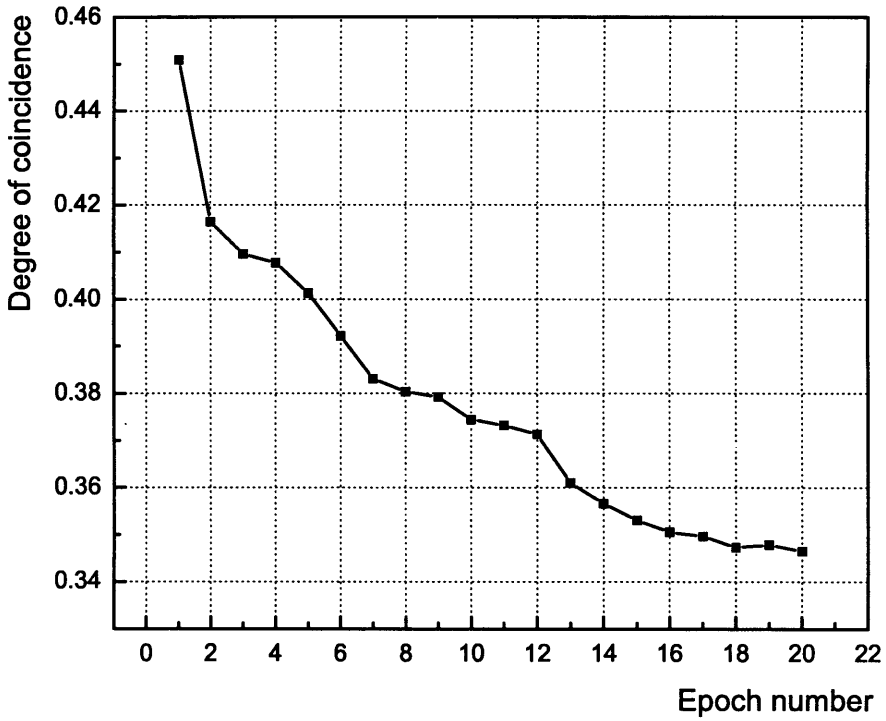


Figure 4.14. Degree of coincidence achieved on the SODA_SNN trained on Control chart data.

connection delays of the networks were randomly assigned in the range of 10 to 20. These figures show that with the stability measures, the connection delays reached a normal distribution with much of the delays concentrated in the central region of the allowed delay range. The proposed measures effectively kept the connection delays with relatively low values, resulting in a stable system.

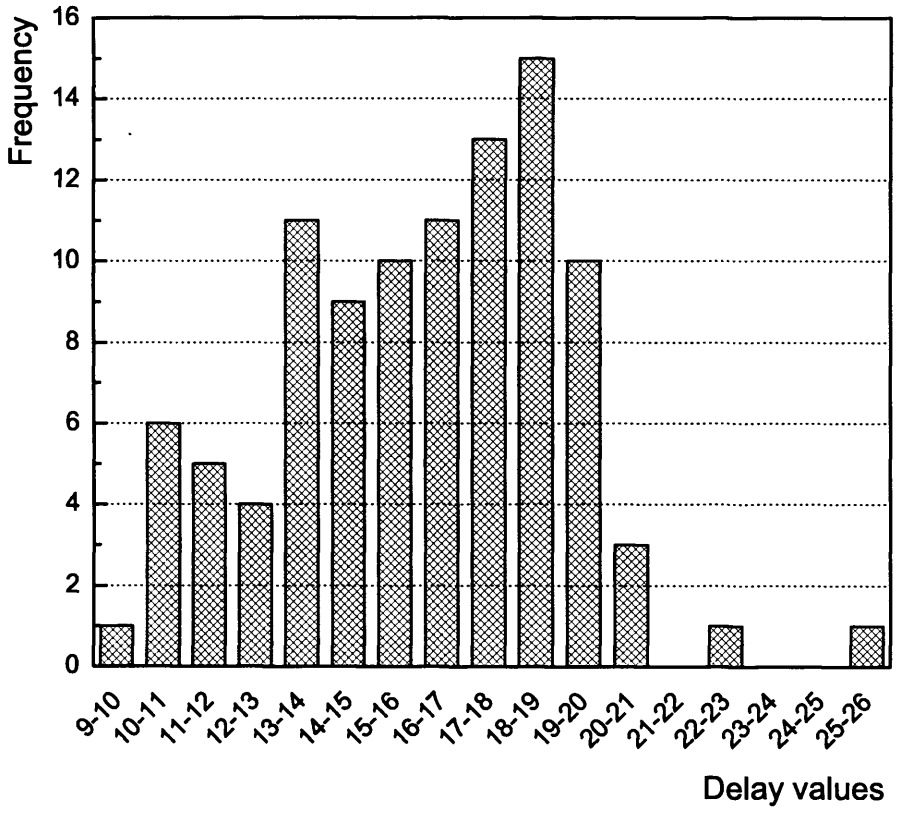


Figure 4.15. Distribution of the connection delays of the SODA_SNN trained on Iris data.

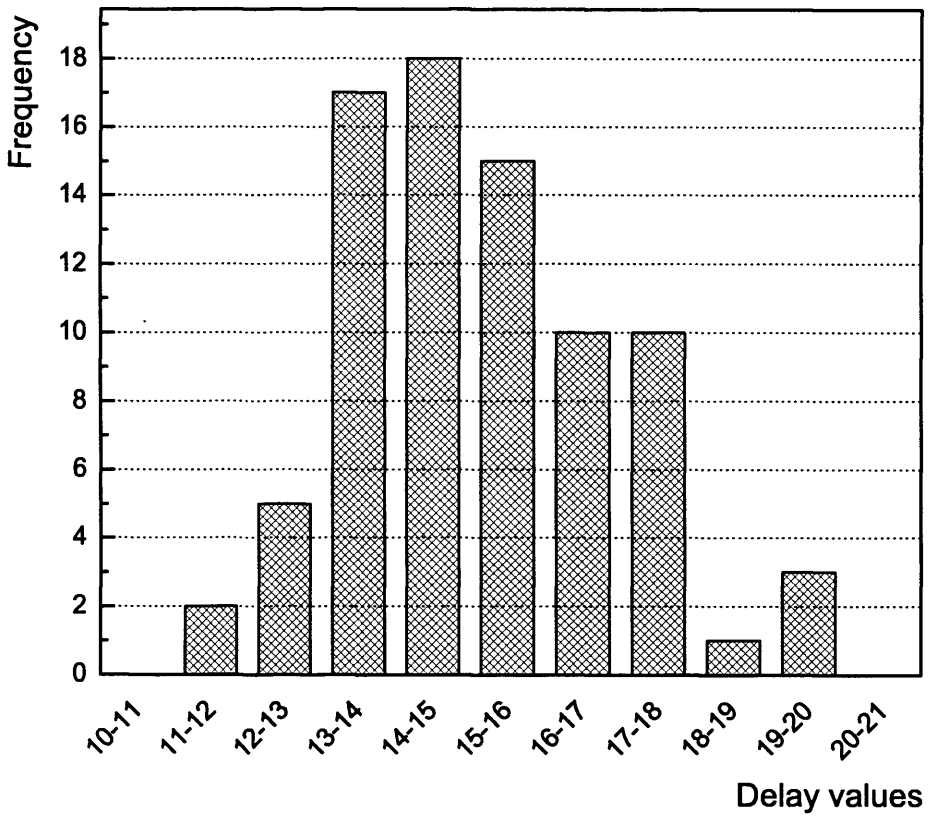


Figure 4.16. Distribution of the connection delays of the SODA_SNN trained on Cancer data.

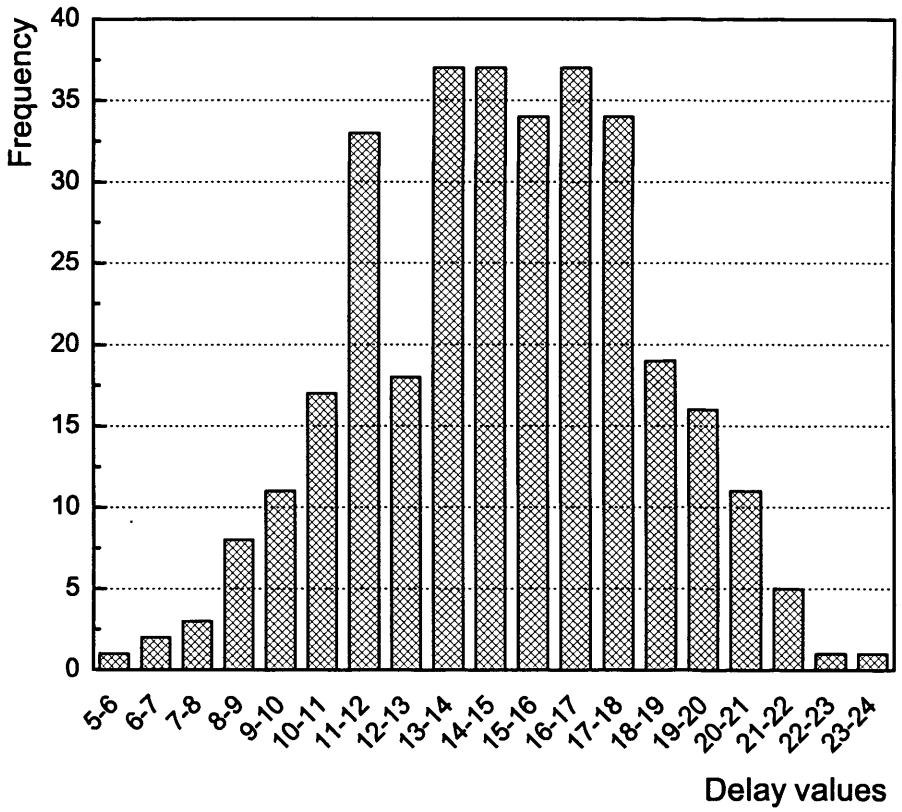


Figure 4.17. Distribution of the connection delays of the SODA_SNN trained on Wine data.

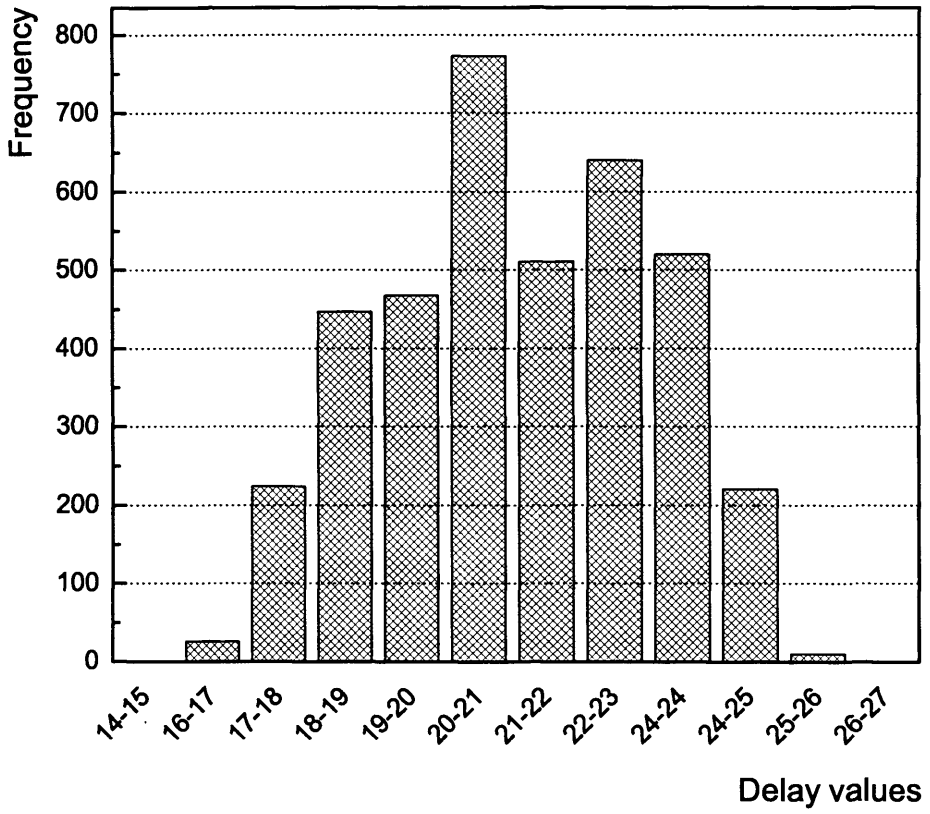


Figure 4.18. Distribution of the connection delays of the SODA_SNN trained on Control chart data.

4.7 Conclusion

Biological neurons have been found to be utilising a form of coincidence detection for information processing which is considered to be the key for some of their high speed cortical actions. The temporal coding spiking neural networks possess this coincidence detection ability, which is a relatively novel concept in neural networks for pattern recognition. This efficient feature of spiking neural networks was utilised for developing a learning model for temporal coding spiking neural networks. In this chapter, the Self-Organising Delay Adaptation Spiking Neural Network (SODA_SNN) for clustering tasks has been proposed. This model employs a Hebbian-based delay adaptation mechanism to train the network, which encodes the input information in the connection delays. This mechanism directly adapts the connection delays instead of the previous approaches of modifying connection weights to select suitable delayed connections. An efficient learning mechanism utilising a dynamic threshold approach has been employed, which enabled the network to learn efficiently. A stabilisation measure has been incorporated which utilises the degree of coincidence of inputs in order to control the delay adaptation effectively.

The proposed model has been implemented in software and various aspects have been analysed in order to investigate its characteristics. The SODA_SNN's clustering capability was successfully demonstrated by applying it to cluster benchmarking data sets and also a high dimensional data set. The performance of the model was found to be better than the previous unsupervised spiking neural network learning models. The clustering capability of the proposed model was compared with the Kohonen's SOM and found to be better. The proposed model's capability was shown to be better than the SOWA_SNN proposed in chapter 3, especially in terms of cluster formation. Through analysing the out-

put and the delay distribution during and after learning, the SODA_SNN was demonstrated to be performing efficiently. The method introduced to realise the two modes of operation of the spiking neurons was also showed to be effective. It can be concluded, therefore, that the self-organising delay adaptation spiking neural network is an efficient tool for clustering applications.

Chapter 5

Supervised Delay Adaptation Spiking Neural Network (SDA_SNN)

5.1 Introduction

This chapter investigates supervised learning in spiking neural networks and proposes a novel supervised learning model to train temporally coding spiking neural networks for classification tasks. The spiking neurons used in the proposed learning model operate as coincident detectors which become optimally active when they receive synchronised inputs. The proposed learning process adapts the network's connection delays rather than the connection strength. The learning model shifts the connection delays of one or a group of neurons in order to detect temporal patterns from a particular class. The network architecture of the proposed model is similar to a Learning Vector Quantisation (LVQ) network [Zurada, 1999] and the model utilises a Hebbian-based learning rule which acts as an error estimator. The efficiency of the model was shown by applying it to classify four data sets successfully. Analytical studies were also performed to establish the characteristics of the proposed model. The proposed model was able to achieve better classification accuracies with smaller networks and fewer training epochs

than the existing supervised SNN models and Multi Layer Perceptron (MLP) [Haykin, 1999] networks.

This chapter is structured as follows; section 5.2 introduces the supervised learning paradigm for artificial neural networks. Popular supervised learning models for conventional neural networks are also briefly explained here. In addition, this section summarises the previous research related to supervised learning in spiking neural networks. The proposed model for supervised delay adaptation learning is introduced in section 5.3. The implementation details of the proposed model are given in section 5.4. In section 5.5 the simulation results are presented with discussions. Finally, conclusions are given in section 5.6.

5.2 Supervised learning in artificial neural networks

Supervised learning, or learning with a teacher, is a well-known learning paradigm used to train artificial neural networks. This type of training is possible if the target information of the training data is available. A schematic diagram which describes this learning technique is shown in figure 5.1 [Haykin, 1999] and explained below.

The environment shown in figure 5.1 is the knowledge about, or collected information of, a particular domain. The teacher presents examples drawn from the environment, usually in an input-output pair, to the learning system. The actual output of the system is compared with the desired output and an error value is computed. The learning system modifies the connection weights in such a way as to reduce this error. This process is continued in a step-by-step manner until some terminating criteria is met. The terminating criteria could be based on several factors, such as the total error or the total change in the system.

The training can be stopped when the total error reaches a minimum value or the change in the system becomes stable. The stopping criterion is necessary for generalising the trained network in order for it to function properly with unseen data. After the training process is completed, the network can function independently in recall mode without the help of the teacher. In recall mode, when the network is presented with an input pattern, the generated output is expected to be closer to the desired output. Hence the network can be employed to find the desired output for data objects belonging to the same domain where the target information is unknown [Haykin, 1999; Zurada, 1999; Pham and Liu, 1999]. The following sub-sections describe some popular supervised learning models for conventional and spiking neural networks.

5.2.1 Supervised learning in conventional neural networks

There are a number of popular supervised learning models available for conventional networks. *Perceptrons* are the basis for most of these models, which can be considered as the simplest form of a neural network, with a single neuron consisting of adjustable synaptic connections. This non-linear neuron, which was based on the McCulloch-Pitts neuron model, consists of a linear combiner followed by a hard limiting activation function. Rosenblatt [Rosenblatt, 1958] introduced the perceptron and a novel method of supervised learning as a new approach to solving the pattern recognition problem. This single neuron model is capable of classifying patterns belonging to two linearly separable classes. By increasing the number of neurons, classification of more than two classes can be achieved. However, the classes should be linearly separable. In a different approach, Widrow and Hoff [Widrow and Hoff, 1960] introduced the *Least-mean square (LMS) algorithm*, which is also known as the *delta rule*. Rosenblatt's perceptron learning algorithm and the LMS algorithm are error correction algorithms, where the former applies

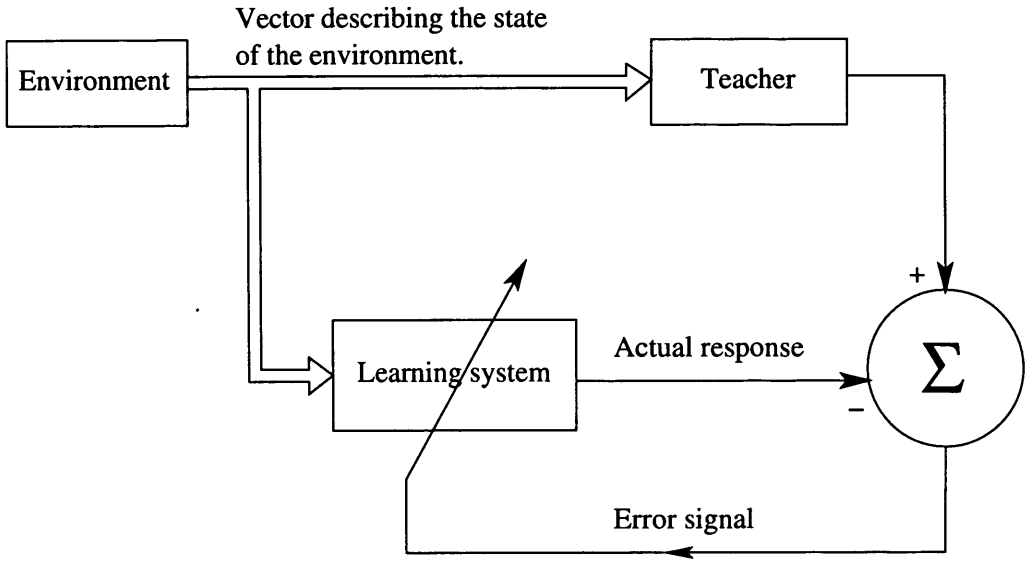


Figure 5.1. Schematic diagram of supervised learning. Redrawn from [Haykin, 1999].

to binary valued units and the latter to continuous valued units.

It was reported in [Minsky and Papert, 1969] that there are fundamental limits on what single layer perceptrons can compute. This was overcome with the introduction of *error back-propagation algorithm* for the training of multilayer perceptron networks, which was reported in [Rumelhart et al., 1986]. The multilayer perceptron network, a generalised version of the single layer network, includes one or more hidden layers in addition to the output layer of computing neurons. An important difference as concerns the neurons in the multilayer perceptron networks is that the non-linear activation function used is continuous instead of the hard limiting function of the perceptrons. In fact, this continuity is an essential condition for the error back-propagation algorithm to converge. During training, the actual output of the network and the desired output are utilised to compute an error value which is propagated back to each layer of neurons. Efficient learning rules were derived to update the weight of the connections between neurons in all layers. The multilayer perceptron network with the error back-propagation algorithm is the most popular network model for supervised learning. This model has been applied successfully in solving numerous difficult and diverse problems [Haykin, 1999].

Another supervised learning paradigm is the Radial Basis Function (RBF) network, which takes a completely different approach as compared to the error back-propagation algorithm [Haykin, 1999]. The network consists of an input layer, a hidden layer and an output layer. Hidden neurons are represented with radial basis functions, generally a Gaussian function, as specified in equation 2.13. The network is trained by finding proper values for the centre and radius for the basis functions of each hidden neuron and the connection weights for the output neurons. When an input pattern is presented to the network each hidden neuron computes the distance between its centre and this input vector, generally

through a Euclidean norm. Based on these values, the neurons in the output layer produce a weighted sum which is the output of the network [Pham and Liu, 1999].

The LVQ network is another important supervised learning model, which is structured with three layers of neurons: an input layer, a non-linear hidden layer and a linear output layer. The network is fully connected between the input and hidden layers and partially connected between the hidden and output layers, with each output neuron linked to a different group of hidden neurons. Each output neuron represents a particular class of input data. Connections between the hidden and output neurons are assigned with a fixed weight value 1. The connections between the input layer and hidden layer are updated during training. When an input pattern is presented to the network, a hidden layer neuron with the connection weights closest to the input pattern in terms of Euclidean distance is found. This neuron, which is called the winner, produces an output of 1 and the output neuron connected to this neuron also generates an output of 1. The connections are strengthened if this neuron is connected to the desired output neuron but otherwise weakened. The strengthening (or weakening) of the connections is performed in such a way that the connection weight vectors of a particular group of hidden neurons become closer to the corresponding input vectors [Pham and Liu, 1999].

5.2.2 Supervised learning in spiking neural networks

Supervised learning algorithms are reliable training models for artificial neural networks and they are, therefore, widely employed for training network models. A number of different supervised learning models for spiking neural networks can be found in the literature. These models can be grouped into two categories based on the underlying training strategy, namely error gradient decent based models

and Hebbian rule based models [Ponulak, 2005]. Most of the supervised learning models found in the literature adapt the connection weights. The following subsection summarises the existing learning models in the above two categories and analyses them.

5.2.2.1 Error gradient based learning models

It is well known that the error back-propagation is an efficient and guaranteed learning paradigm for multilayer feed-forward networks. The efficiency and reliability of this algorithm has inspired many supervised learning models, including the models for spiking neural networks. Bohte et al. [Bohte et al., 2002a] introduced a supervised learning method for multilayer feed-forward spiking neural networks based on error gradient decent. With the aid of some important assumptions, learning equations were derived in the same manner as Rumelhart's error back-propagation learning rules. The assumptions were taken to fulfil the continuity criteria of the activation function. A special type of network was utilised, where connections between two neurons were constructed of several subconnections, each characterised with different delay and weight values (see figure 3.3).

The precision and the classification capability of the learning model proposed in [Bohte et al., 2002a] was further improved with a population coding scheme. This coding scheme aimed to improve the resolution of the temporal input code, thus increasing the classification precision and capability of the network. According to the coding scheme, a single input value is represented with a vector of values. This vector was found with the aid of a one-dimensional receptive field comprised of overlapping Gaussian functions as shown in figure 5.2. This supervised learning model, along with the coding scheme, was applied to the classification of several data sets and found to be successful. In addition, its

performance was also found to be comparable with that of MLP networks.

Improvements for the model described above were proposed in [Schrauwen and VanCampenhout, 2004]. In the model proposed in [Bohte et al., 2002a], only the connection weights are modified, but here rules were proposed to adapt connection delays, time constant and the neuron's threshold in addition to the connection weights. The derivation of these rules was similar to the one proposed in [Bohte et al., 2002a]. However, the applicability of these rules was not verified. Improving the model proposed in [Bohte et al., 2002a] by adding a momentum term to the learning rule was specified in [Jianguo and Embrechts, 2001].

Error correction learning is the usual choice for supervised learning in conventional neural networks. A condition for convergence in error gradient descent models is that the activation function of the neurons should be continuous. The common use of a sigmoidal or hyperbolic tangent as the activation function is in line with this condition. In spiking neural networks, the output is the time when the neuron potential exceeds a threshold value. Hence, the activation function of a spiking neuron behaves more like a threshold function. Therefore, the error gradient method of supervised learning cannot be explicitly evaluated in spiking neural networks [Kasisnski and Ponulak, 2006]. The error gradient based learning models proposed for spiking neural networks tackle this problem with two assumptions. The first assumption is to consider the spike response function to be linearly increasing up to its maximum point and then decreasing linearly. The second assumption is that the activation function is considered to be continuous for a small enough region close to the moment of firing [Bohte et al., 2002a]. The model's efficiency relies on these assumptions. However, due to the discontinuity of the activation function, the learning rules proposed in [Bohte et al., 2002a] are very complex and require high computational effort.

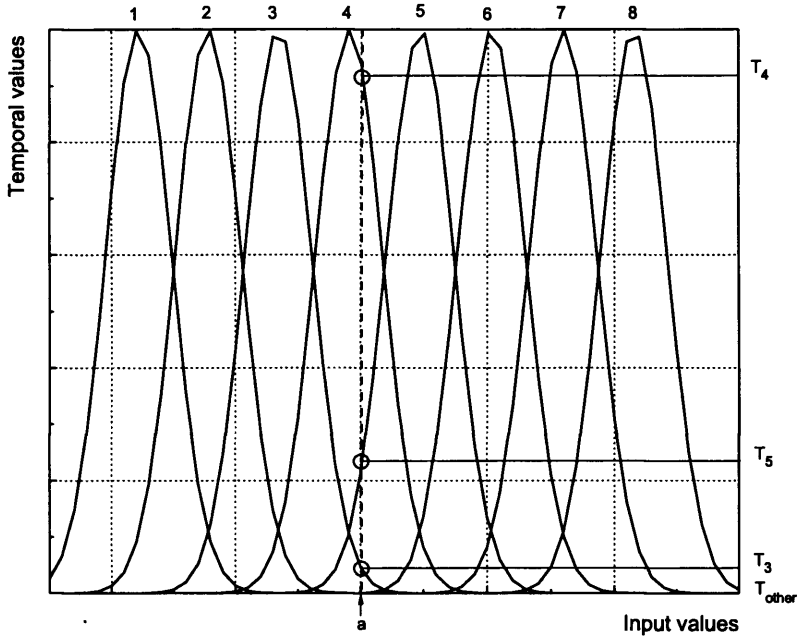


Figure 5.2. Population coding with overlapping Gaussian receptive fields. Here an input value a is translated into firing times as (T_o, T_o, T_3, T_4, T_5, T_o, T_o, T_o, T_o, T_o). T_3, T_4, T_5 are the intersecting points of the line at a with Gaussian curves 3, 4 and 5 respectively. T_o represents temporal value T_{other} where all the other curves intersect with the line at a. Redrawn from [Bohte et al., 2002a].

Even though the classification accuracy of the model proposed in [Bohte et al., 2002a] was comparable to the results obtained with MLP networks, the size of the network was much greater. The number of input nodes was increased by n-folds, where n is the number of nodes required to represent a single input value by the population coding scheme, thus increasing the number of connections. In addition, the multiple sub-connections which replaced a single connection increased the size of the network heavily.

5.2.2.2 Hebbian rule based learning models

A number of learning models for spiking neural networks are based on Hebbian type rules. Ruf and Schmitt [Ruf and Schmitt, 1997] proposed a Hebbian-based supervised learning model which encodes the input information in the connection weights. This learning model utilises the timings of the pre- and post-synaptic firing of the neurons. It was shown that the learning rule modifies the connection weights so as to be closer to a value representing the time between the pre- and post-synaptic firing events. The spiking neuron considered in this work was introduced by Maass [Maass, 1996], which was described in section 2.5. A mono-synaptic learning rule was proposed to train a single synapse with temporally encoded inputs. The network is activated for a time period for each learning cycle. During a learning cycle, an input to neuron i given at t_i enables the output neuron j to fire at some time t_j. Along with the input spike, a reference spike is also given, say at time t_0. This reference spike specifies the input strength when considered with the input t_i and determines the amount of weight modification when compared with the timing of the output spike. The learning rule is given by equation 5.1, which modifies the connection weight in such a way as to force the output neuron to fire at a time as close as possible to the firing time of the

specified reference spike.

$$\delta w_{ji} = \eta(t_j - t_0) \tag{5.1}$$

where δw_{ji} is the amount of change on strength of the synapse from neuron i to neuron j; η is the learning rate. The time difference $(t_j - t_0)$ is considered as an error value which the learning rule tries to reduce. It was mathematically proved that the repeated application of the rule 5.1 makes the connection strength w_{ji} to approach a value which is proportional to $\frac{1}{(t_i - t_0)}$. It was suggested in [Ruf and Schmitt, 1997] that the connections can be trained in parallel through a normalisation technique. This modified rule is given in equation 5.2.

$$\delta w_{ji} = \eta(t_j - t_0)/\|\mathbf{w}\| \tag{5.2}$$

where δw_{ji} and η are same as defined above. $\mathbf{w} = (w_{j1}, w_{j2}, ...w_{jn})$ is the weight vector representing the connections from all the input neurons to neuron j; $\|\mathbf{w}\|$ is the Euclidean norm. The mono-synaptic learning rule specified by equation 5.1 can train only a single synapse at a time. Both rules, 5.1 and 5.2 specify the error by the term $(t_j - t_0)$. 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. Hence, the applicability of this model for real problems is questionable. An important assumption of this model is that the potential rise in the neuron due to an incoming spike is linear.

Hebbian learning is much more biologically realistic. There are numerous examples in biological modelling studies where Hebbian-based learning rules have been implemented. In order to obtain the optimum power of the spiking neural networks, it is necessary to develop learning algorithms which capture the special aspects of these networks. Classical techniques have been shown to be excellent for the sigmoidal network models, but their efficiency is questionable in biologically realistic network models [Maass, 2001b]. In this context, the coincidence

detection capability of a spiking neuron posses great potential. A novel Hebbian-based learning model with a network of coincidence detecting spiking neurons is, therefore, introduced in the following sub-section as an alternative learning strategy.

5.3 Proposed supervised delay adaptation SNN for classification

This section presents the proposed supervised delay adaptation spiking neural network (SDA_SNN) for classification. The pattern detection capability of temporal coding spiking neural networks is utilised in this model. The proposed model encodes the input information in the connection delays in a similar fashion to that realised in the self-organising delay adaptation spiking neural network proposed in chapter 4. But instead of an unsupervised learning strategy, this model employs a supervised approach. The following sub-sections describe the model in detail.

5.3.1 Network architecture

A network architecture similar to that of an LVQ network is utilised for the SDA_SNN. Figure 5.3 illustrates a sample structure of the network. The structure comprises an input layer, a hidden layer and a linear output layer. The input layer is composed of a number of simple input neurons equal to the number of attributes present in the input patterns. The hidden layer is constructed with spiking neurons, introduced in chapter 2 of section 2.5. Here, the neurons are realised as coincidence detectors. Section 4.4.2 presents more detail regarding the realisation of coincidence detection with spiking neurons. The input neurons and the spiking neurons are fully connected with feed-forward connections. Each

connection is characterised by a weight value and a delay value. Each class of the data set is represented by a group of spiking neurons. These neurons are connected to a single linear neuron in the output layer. This neuron produces an output of 1 if any of the neurons connected to it fires first during a specified time window.

The proposed network structure can have only a single hidden layer of spiking neurons because of the utilisation of a Hebbian-based learning rule. This single layer of neurons is adequate for the detection of temporal patterns in the input data. The number of neurons representing each class in the input data can be varied depending on the complexity of the data. In addition, each class can be represented by different number of neurons depending on the resolution of the input of a particular class. This arrangement would keep the number of computing neurons to a minimum and hence control the computational cost.

5.3.2 The learning rule

Learning is achieved through adapting the network connections' delays in such a way that they can detect temporal input patterns. The learning rule proposed in this study is based on the delay adaptation rule introduced in [Tversky and Miikkulainen, 2002]. This Hebbian-based rule has been employed in the self-organising model proposed in chapter 4. The learning rule employed in the proposed model is specified in equation 5.3 and described graphically in figure 4.2.

$$\delta d_{ji} = (\delta t - s)\left(\frac{e^{-(\delta t - s)^2/\tau_{stdp}^2}}{\tau_{stdp}}\right) - b \tag{5.3}$$

where δd_{ji} is the change in delay value for the connection between the spiking neuron j and the input neuron i. τ_{stdp} is the time constant for the learning rule; b is a positive bias value to stabilise the learning. δt is basically the difference

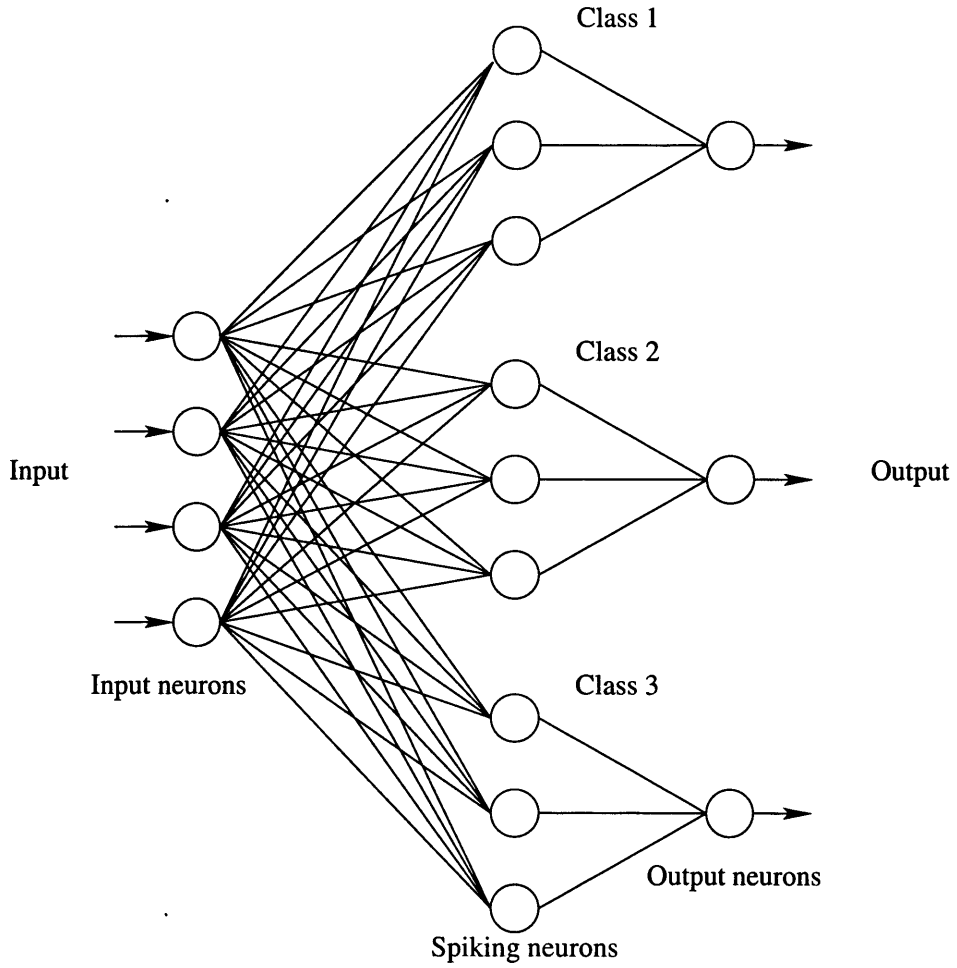


Figure 5.3. The structure of the supervised delay adaptation SNN.

between the timing of the delayed input spike and the timing of the output spike, which can be found with the equation 5.4.

$$\delta t = t'_j - (t_i + d_{ji}) \tag{5.4}$$

where t'_j is the firing time of a spiking neuron j and t_i is the firing time of input neuron i. s is a positive term which shifts the learning rule in the direction of positive X axis. Here b and s are realised in the same way for stabilising the learning as in chapter 4 (see sub-section 4.4.5 for more detail).

5.3.3 Delay change estimation

The objective of the learning process is to tune the connection delays of a single neuron in such a way that the delayed inputs coincide with each other. The training of the network is conducted in a supervised approach using the learning rule specified in equation 5.3. This rule estimates the amount of modification for each connection delay in order to achieve the objective of the training. There is a need for the selection of appropriate values for the learning rule parameters τ, τ_{stdp}, b and s. It was found experimentally that better learning could be achieved if the values for the time constants τ and τ_{stdp} are set to allow the learning function to reach its maximum when the spike response function begins to saturate, as shown in figure 4.2.

Connection delays are updated based on the difference between the time when an input spike arrives and the time when an output spike is generated. If an input spike arrives at a neuron through a delayed connection before there is an output spike, then the delay associated with that connection is increased so that the time difference can be reduced. If the input spike arrives after the generation of an output spike, then the connection delay is decreased. If the output spike follows an input spike within a very short time, then the delay change is set to zero

or slightly negative, in order to control the adaptation of the connection delays. This is achieved through the term s in the learning rule. A similar situation applies when an input spike reaches the neuron much after the output spike is generated. The change in connection delay is given by parameter b in this case. Thus b and s specify the range of spike time differences where the connection delays are increased or decreased.

5.3.4 Controlling the learning

Controlling the learning has been achieved in two distinct phases. One is to control the learning in the group of neurons which represent a particular class and the other is to control the learning in the network as a whole. Training the network must be controlled properly in order to achieve better classification and generalisation. Since a class is represented by a group of neurons, a measure based on the coincidence of inputs is incorporated to correctly train each neuron in that group. A simple linear measure given in equation 5.5 can be used to control the learning, which is based on the time between the first and the last delayed input spikes to a neuron. This difference specifies the time width in which all the input spikes reaches the neuron. A measure of coincidence can be specified by dividing this time difference by some appropriate value. Here the value t_{input_window} is a suitable selection.

$$cm_j = \frac{\max\{t'_i\} - \min\{t'_i\}}{t_{input\_window}}, \; i = 1..n \tag{5.5}$$

where t'_i, for $i = 1..n$ is the reaching time of input spikes t_i at neuron j, which is equivalent to $t_i + d_{ji}$; where d_{ji} is the delay of the connection between neuron i and j.

If the delays had been tuned to enable the input spikes to coincide, then further modification is not necessary. Hence, the amount of change for a particular

connection delay could be decreased while still achieving the required degree of coincidence. Hence, if the total change of connection delays falls below a minimum value or the change remains constant, then the training process is terminated. To control the learning in the whole network, a global error value is utilised. This error value is calculated based on the desired outputs and the actual outputs. Training continues until the error value falls below an acceptable value or when no change is observed.

5.3.5 Interpreting the results and classifying the input data

The output of the network is interpreted as in chapter 4. Based on the output, a trained network can effectively detect temporal patterns. The class of an input sample can be identified by the neuron first to fire. This spiking neuron is more likely to have the connection delays which can compensate for the differences in the firing timings of the input spikes. Hence, the group to which this neuron belongs would specify the class of the input pattern.

In a straightforward approach, an output neuron can be set to generate an output of 1 when it receives the first output spike from one of the spiking neurons connected to itself. All the other neurons can be forced to generate an output of 0. Hence, the output neuron connected to the spiking neuron which is first to fire will generate an output of 1 while all the others of 0. The activity of the network can be stopped after the firing event of a spiking neuron since the following spikes will not affect the output. This approach will reduce the computational time and effort significantly.

Even though the above procedure achieved a high degree of classification accuracy, it was observed that the accuracy can be increased further by utilising

the firing times of those following the winner neuron. More specifically, the firing times of the spiking neurons which are first to fire in each group can be used for this purpose. As observed in the previous two models, this model was also found to be firing within a well defined time window when correctly detecting patterns. It was observed that when the first firing neuron fails to identify the correct class of the input pattern, it was, in most cases, followed by a neuron from the correct group. This observation has been utilised in this model to increase the classification accuracy.

On completion of the training process, the output of the spiking neurons for the training set was obtained and analysed to determine the suitable time window of firing for neurons belonging to each group. These values were assigned to the corresponding output neurons. The output neurons were set to generate an output of 1 if the first spike they received fell within the specified time window. Otherwise the output was set to 0. The activity of the network can be stopped after the generation of output 1 from any of the output neurons.

5.3.6 The training process

During the training process, temporally coded input vectors were presented to the network and the outputs from all the neurons were found. The connection delays were adapted in two phases. In the first phase, the connections to all the neurons in the desired group were updated using equation 5.3. In the second phase, only the neuron which was the first to fire in the desired group of neurons was updated.

A major obstacle in delay adaptation learning was to keep the neurons active while detecting the coincidence of inputs with adequate accuracy. In the initial stage of training, a high threshold value will prevent neurons from firing. On the

other hand, a low threshold value will not detect the coincidence properly. In order to keep relevant neurons active, a low threshold value was assigned initially and increased after each training epoch in small equal steps to a preset value.

5.4 Implementation details

The proposed model was realised as a discrete model and implemented in software in a similar manner to the previous two SNNs. Continuous input values were coded using small temporal differences, as in the previous two models, using the equation 2.14. For ease of implementation, temporal differences were calculated in discrete unit intervals. The spiking neurons were realised as coincidence detectors by assigning a relatively low value for the time constant of the spike response function with respect to the input time window as described in sub-section 4.4.2 of chapter 4. The network connections were assigned with approximately equal weight values and random delay values in the mid range of the input time window. The connection weights were kept fixed and the delays were allowed to be in the range $(0, t_{input\_window})$. All the neurons were assigned with equal threshold value. Initially, a low threshold value was assigned and then increased linearly to a predefined value with each training epoch.

5.5 Simulation results and discussion

The SDA_SNN was analysed by applying it to classify the four data sets described in chapter 2 of section 2.6.2 in order to investigate its characteristics and the learning ability. Similar to chapter 4, classification capability, activity of the network, degree of coincidence and the stability of the proposed model were investigated. The following sub-sections discuss these in detail.

5.5.1 Classification capability

This section presents the classification results obtained by the SDA_SNN and compares them with those for other SNN models and the MLP network. Similar to the chapters 3 and 4, standard bench marking data sets, Cancer, Iris and Wine, were used for this purpose. In addition, the high-dimensional Control chart data was used to demonstrate the capability of the proposed model in more complex cases. The detailed description of each data set can be found in section 2.6.2, and a summary in table 2.1.

Networks were constructed to classify each data set and trained with approximately 66% of randomly selected samples of the data set. The remaining samples of the set were used to test the trained network. The classification accuracy of the training and testing phase were computed using the class detail of each sample available in the data sets. The training and testing of each network was conducted five times and the average classification accuracies were calculated. Table 5.1 lists the obtained optimum training and testing results on each data set. The data used for training and testing is given in Appendix B.

It was stated earlier that the network parameter values should be selected appropriately in order to achieve better learning. The parameter values of the networks obtained the optimum results for each classification task are listed in table 5.2.

The results provided for the supervised Hebbian-based learning model in [Ruf and Schmitt, 1997] were for artificial data sets. The results reported for the supervised error back-propagation learning model proposed in [Bohte et al., 2002a] are given in table 5.3. Although the classification accuracy for the training set of Iris data is approximately similar for both models, the classification accuracy

Data set	No. of neurons		No. of training epochs	Classification accuracy(%)	
	Input	Hidden		Training	Testing
Iris	4	3x3	15	97.4 ± 1.0	97.3 ± 1.0
Cancer	9	2x2	10	97.9 ± 0.6	97.3 ± 0.6
Wine	13	3x4	15	97.7 ± 0.7	96.8 ± 0.7
Control chart	60	6x4	30	97.0 ± 0.2	96.9 ± 0.2

Table 5.1. Average classification accuracy obtained for the SDA_SNN.

Data set	Training	τ	τ_{stdp}	b	s	η	Threshold	epochs
Iris	Phase1	5	25	0.05	2	0.1	0.7-0.8	10
	Phase2	5	25	0.05	2	0.1	0.8-0.85	5
Cancer	Phase1	5	25	0.05	2	0.1	0.5-0.7	7
	Phase2	5	25	0.05	2	0.1	0.7-0.8	3
Wine	Phase1	5	25	0.12	2	0.2	0.55-0.7	10
	Phase2	5	25	0.12	2	0.1	0.7-0.75	5
Control chart	Phase1	5	25	0.12	2	0.2	0.7-0.78	20
	Phase2	5	25	0.1	2	0.2	0.78-0.83	10

Table 5.2. Values assigned for the parameters of the network used to obtain the optimum results.

of SDA_SNN on the test set of Iris data has been found to be better. SDA_SNN obtained higher accuracy on both the training and testing set of Cancer data. For all the data sets, the SDA_SNN achieved its highest classification accuracy with a much smaller network. It can be seen from table 5.3 that the size of the networks used in [Bohte et al., 2002a] were very high due to the population coding scheme. In addition each connection was composed of multiple sub-connections. Hence in terms of accuracy and efficiency it can be claimed that the proposed model performs better. The SDA_SNN achieved higher classification accuracy for the Wine data and the high dimensional Control chart data, which exhibits the classification capability of the proposed model. Hence the SDA_SNN can be considered as a better supervised learning model for spiking neural networks.

In order to compare the classification capability of the SDA_SNN against sigmoidal networks, the same data sets were classified using a multilayer perceptron network with an error back-propagation algorithm. Programs were developed using *Matlab* and applied to classify the data sets. For each data set a network with two hidden layers was created. Computing neurons had hyperbolic tangent (*tansig*) activation and resilient back-propagation (*trainrp*) training was adopted. Initially, for ease of comparison, the number of training, testing samples and the number of training epochs were kept the same for MLP networks as for the corresponding SNNs obtaining the optimum results. The size of the MLP networks were selected according to the usual practice. The classification accuracies obtained with MLP in the above-mentioned configuration are given in table 5.4. Better results were obtained with the proposed model on all training and testing data sets. In all cases, the number of neurons and the number of connections required for the SNN models were far fewer compared to the MLP network.

The MLP networks mentioned above were trained further to get their highest classification accuracy. Table 5.5 lists the results obtained. Although the clas-

sification accuracies of the SDA_SNN on all the training sets were slightly lower than the MLP network, their accuracy on the test sets was found to be higher. It is notable here that the MLP networks required a very high number of training epochs to obtain their highest classification accuracies but the SDA_SNN required significantly fewer training epochs. In addition, the size of the SDA_SNNs utilised to classify the data sets were much smaller than the size of the MLP networks. Hence it can be concluded that the classification capability of the proposed model is better than that of the MLP networks with error back-propagation.

5.5.2 Network activity

The activity of the SDA_SNN during training was analysed by counting the active neurons after each training epoch. Table 5.6 summarises the average number of active neurons initially and after each phase of training on each data set. Here the SNNs analysed were selected to have the same configuration as that of the networks which obtained the optimum results in the previous sub-section. It was observed initially that almost all the neurons were active and the number of active neurons decreased gradually due to the increase of the threshold value with each epoch. However, the training process effectively kept an adequate number of neurons active, especially the neurons belonging to the relevant groups.

5.5.3 Degree of coincidence

In order to analyse the degree of coincidence of the inputs attained due to training, the output of the SDA_SNNs which obtained the optimal results listed in table 5.1 were utilised. The measure specified in sub-section 5.3.4, based on the difference between the arrival time of the first and the last input spikes to the winner neuron, was used for this purpose. Table 5.7 lists the values of the degree of coincidence

Data set	Network size	No. of training iterations	Classification accuracy(%)	
			Training	Testing
Iris	50x10x3	1000	97.4 ± 0.1	96.1 ± 0.1
Cancer	64x15x2	1500	97.6 ± 0.2	97.0 ± 0.6

Table 5.3. Classification accuracy for Bohte et al.s' [Bohte et al., 2002a] model.

Data set	Network size	No. of training epochs	Classification accuracy(%)	
			training	testing
Iris	4x5x5x3	15	96.8 ± 1.0	93.3 ± 1.8
Cancer	9x5x5x2	10	94.6 ± 0.9	94.1 ± 0.7
Wine	13x5x5x3	15	86.5 ± 1.3	77.8 ± 1.0
Control chart	60x30x12x6	30	53.9 ± 1.4	50.7 ± 1.9

Table 5.4. Average classification accuracy obtained for multilayer perceptron network with error back-propagation learning.

Data set	Network size	No. of training epochs	Classification accuracy(%)	
			training	testing
Iris	4x5x5x3	250	98.0 ± 0.9	96.9 ± 1.2
Cancer	9x5x5x2	200	98.4 ± 0.6	96.3 ± 0.2
Wine	13x5x5x3	90	99.6 ± 0.6	96.1 ± 1.0
Control chart	60x30x12x6	240	99.2 ± 0.4	96.7 ± 0.5

Table 5.5. Average highest classification accuracy obtained for multilayer perceptron network with error back-propagation learning.

Data set	Average No. of active neurons		
	initial	after 1^{st} phase	after 2^{nd} phase
Iris	5.9	4.3	3.8
Cancer	3.0	2.8	2.6
Wine	11.6	6.5	4.9
Control chart	15.8	9.8	5.7

Table 5.6. Average number of active neurons in SDA_SNN initially and after each phase of training.

initially and after each phase of training on each data set. The results show that the learning rules effectively adapted the connection delays and achieved high coincidence accuracy.

5.5.4 Stability

Several measures were incorporated in the SDA_SNN to stabilise and control the learning. Better stability was achieved by selecting suitable values for the parameters τ, τ_{stdp}, b and s of the learning rule given by equation 5.3. Measures to control the learning were discussed in sub-section 5.3.4. Delay distribution in the trained networks was analysed in order to investigate the effectiveness of the proposed stability and control measures. The values of the connection delays of the SNNs trained on the four data sets which obtained the optimum results were used for this purpose. The frequency of the connection delays were found and plotted to find their distribution. Figures 5.4, 5.5, 5.6 and 5.7 show the delay distribution in the SDA_SNNs trained on Iris, Cancer, Wine and Control chart data respectively. Initially, the networks were assigned with random delay values in the mid region (10 to 20) of the allowed delay range (0 to 30). It can be seen from the figures that the connection delays of the trained networks form an acceptably even distribution and many of the delays fall in the mid region of the allowed delay range. Thus it can be concluded that the stability and control measures effectively stabilise the adaptation of connection delays.

Data set	Degree of coincidence		
	initial	after 1st phase	after 2nd phase
Iris	0.37	0.17	0.16
Cancer	0.47	0.29	0.25
Wine	0.57	0.42	0.41
Control chart	0.63	0.35	0.34

Table 5.7. Average degree of coincidence of the inputs achieved by SDA_SNN initially and after each phase of training.

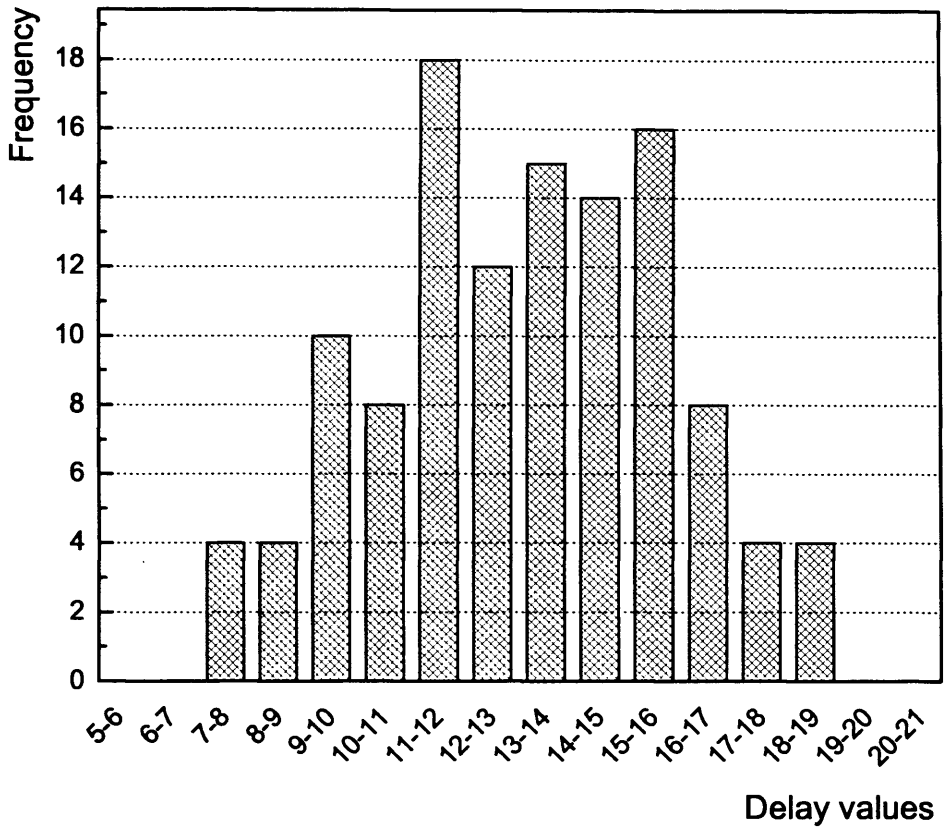


Figure 5.4. Distribution of the connection delays in the SDA_SNN trained on Iris data.

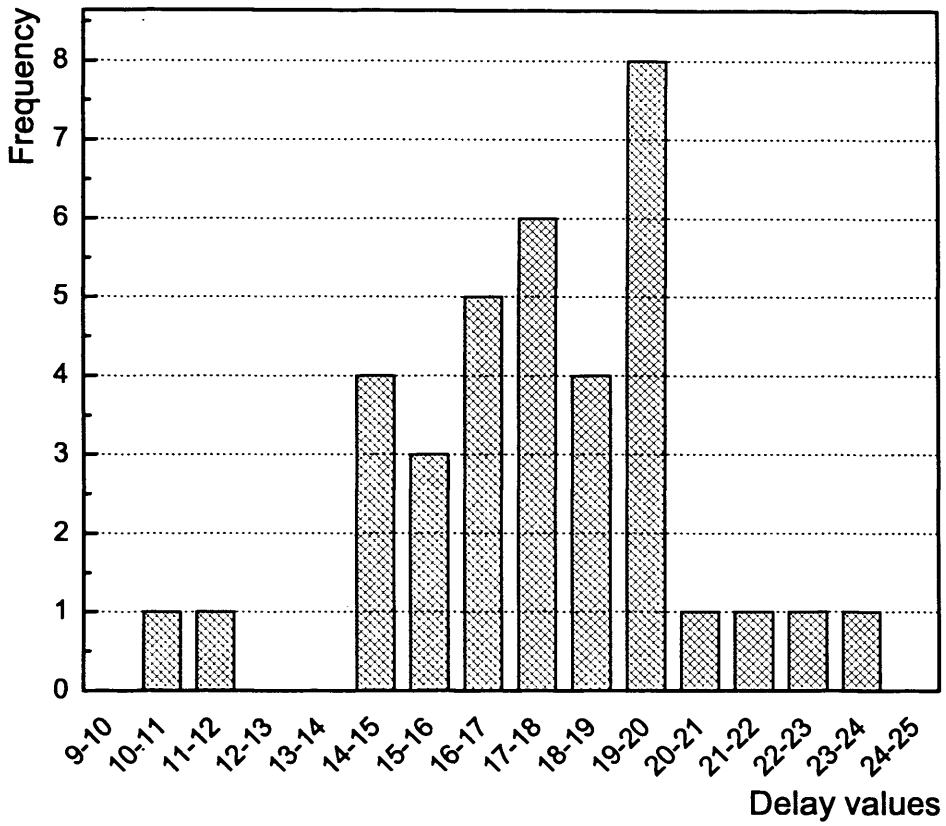


Figure 5.5. Distribution of the connection delays in the SDA_SNN trained on Cancer data.

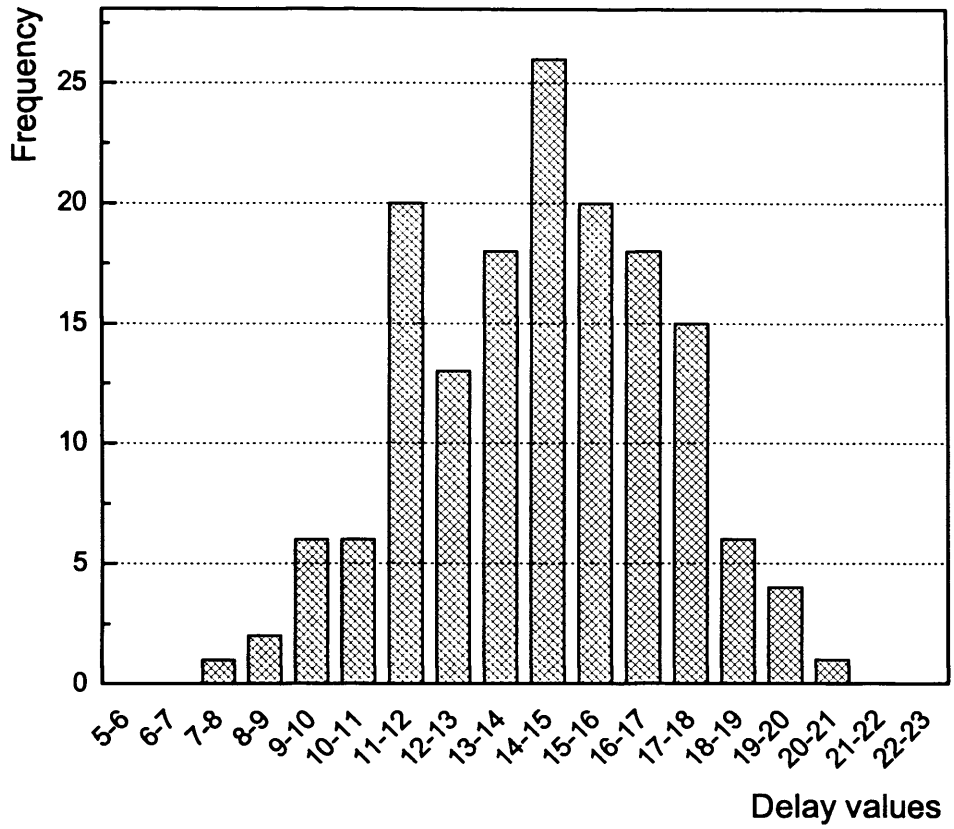


Figure 5.6. Distribution of the connection delays in the SDA_SNN trained on Wine data.

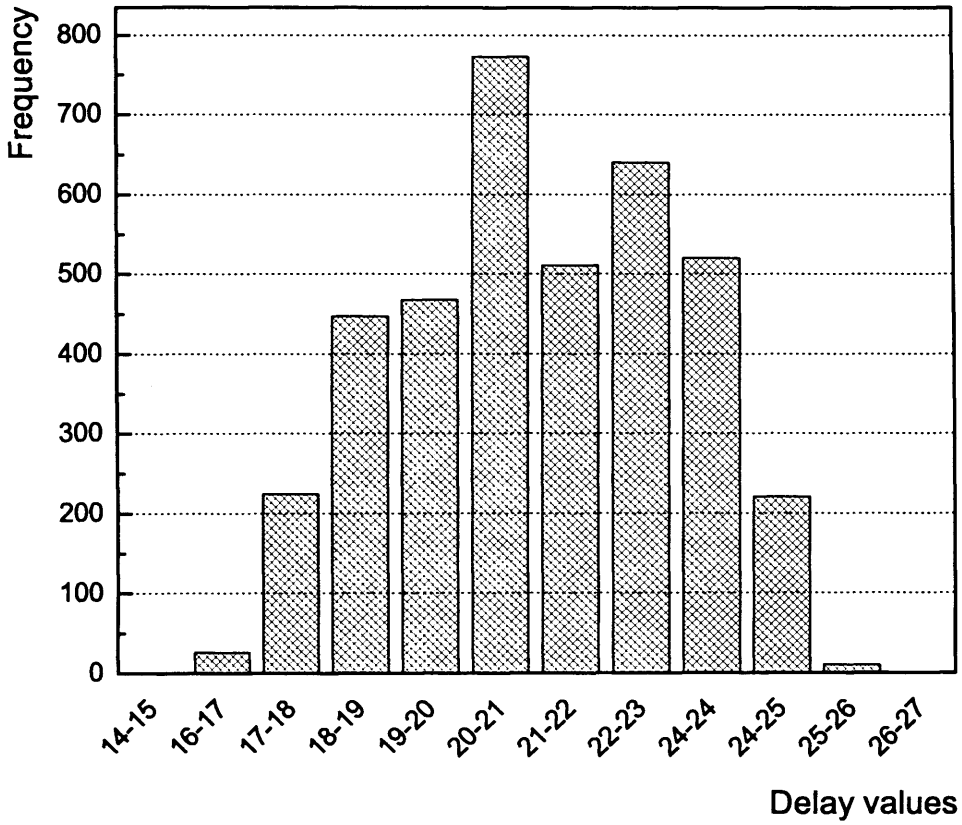


Figure 5.7. Distribution of the connection delays in the SDA_SNN trained on Control chart data.

5.6 Conclusion

The coincidence detection characteristic of a spiking neuron is utilised to develop a novel supervised learning model for temporal coding spiking neural networks. In this chapter the Supervised Delay Adaptation Spiking Neural Network (SDA_SNN) for classification tasks has been proposed. Aided by a Hebbian-based learning rule, the proposed model encodes the input information in the connection delays. Here, the connection delays are shifted instead of selecting the appropriate delays from a set of delayed connections through weight adaptation. Efficient stabilisation and training mechanisms have been incorporated in the proposed model.

The model was realised in software and applied successfully to classify four data sets. The performance of the proposed model was found to be better than the previous supervised spiking neural network models. SDA_SNN obtained better classification accuracy than the MLP networks for all the analysed test data. Further, the SDA_SNN required significantly fewer computing neurons and network connections and was able to learn with fewer training epochs. The compactness, accuracy and the ability to learn with fewer training epochs entitles this network to be considered as a better alternative for classifying data sets including complex and high-dimensional data.

Chapter 6

Conclusion and future work

This research has been focused on the development of alternative temporal coding spiking neural network learning models for clustering and classification tasks. The learning models have been developed appropriately to utilise the special features of the spiking neural networks and to incorporate knowledge from research on biological neural systems and conventional artificial neural networks. Throughout this research, a number of contributions have been made which are summarised in the first section of this chapter. In section two, the conclusions drawn from this study are given. Finally, possible directions for future research are suggested.

6.1 Contributions

The main contribution of this research is the development of the three learning models for spiking neural networks, namely, Self-Organising Weight Adaptation Spiking Neural Network, Self-Organising Delay Adaptation Spiking Neural Network and Supervised Delay Adaptation Spiking Neural Network. More specific detail regarding the contributions are summarised below:

- *Review on spiking neural networks*

  A comprehensive review on topics related to this research has been pre-

sented. This provides a better insight into artificial neural networks, biological neural networks, spiking neural networks and learning models for these networks.

- *Self-organising weight adaptation spiking neural network (SOWA_SNN)*

  A self-organising weight adaptation spiking neural network has been introduced for clustering tasks. The structure of the network is similar to Kohonen's SOM. Here the spiking neurons are realised as integrators and the input information is encoded in the connection weights in a way similar to Kohonen's SOM. Learning is achieved through a Hebbian-based learning rule with efficient stability measures. By incorporating several findings from biological neural networks and a spike time based cluster mapping approach, this network is able to cluster data sets successfully and with high accuracy. The model obtained better clustering accuracy than the previous unsupervised spiking neural network models and their performance was found to be better than the Kohonen's SOM.

- *Self-organising delay adaptation spiking neural network (SODA_SNN)*

  A self-organising delay adaptation spiking neural network for clustering has been proposed. The proposed network utilises a network architecture similar to Kohonen's SOM but with coincidence detecting spiking neurons as the output layer neurons. The network encodes the input information in the connection delays through a Hebbian-based rule which shifts the connection delays. The pattern detection capability of the spiking neural network was efficiently implemented in this model. The network was successfully applied to cluster data sets and was able to form clear clusters within the input data. Its clustering capability was found to be better than the previous unsupervised learning models for spiking neural networks. The performance of the model was compared with the Kohonen's SOM and found to be better.

- *Supervised delay adaptation spiking neural network (SDA_SNN)*

  A supervised delay adaptation spiking neural network has been proposed for classification tasks. The structure of the network is similar to that of an LVQ network except that the hidden layer neurons are coincidence detecting spiking neurons. A Hebbian-based learning rule has been incorporated for realising supervised learning. The idea of pattern detection through delay adaptation and coincidence detection has been successfully implemented. The model's classification capability has been demonstrated successfully and the performance has been found to be better than previously presented supervised spiking neural network models. The performance of the SDA_SNN was also found to be better than the MLP network with error-back propagation.

- *Realisation of the mode of operation*

  It is understood that biological neurons operate in two modes, as integrators and as coincidence detectors. An efficient way of realising these two modes of operations with temporal coding spiking neurons has been introduced and its effectiveness was verified with analytical studies.

- *Implementation of efficient stabilisation techniques*

  The proposed models employ a number of stabilisation techniques to control the Hebbian learning. A soft-bounding method to control the weight adaptation which utilises the current weight values has been incorporated into the SOWA_SNN. Stabilisation of the delay adaptation has been achieved through a measure utilising the degree of coincidence of the temporal inputs. These measures effectively stabilised the learning in the networks. The efficiency of these proposed measures was verified with analytical studies.

- *Introduction of effective learning techniques*

  A number of efficient learning techniques have been incorporated in the proposed models. An effective learning technique utilising a dynamic threshold to train coincidence detecting spiking neural networks has been introduced. This method effectively kept adequate number of neurons active during training, enabling the network to learn evenly throughout the network. A two-phase learning procedure has been introduced to train the SDA_SNN efficiently, which incorporated both supervised and unsupervised learning techniques. The efficiency of the proposed techniques was verified through analysing the networks during and after learning.

- *Simplified learning models for SNNs*

  Proposed learning models are relatively simple and thus easy to implement in software or hardware, requiring fewer computational resources. Utilisation of the timing of spikes and the coincidence detecting capability of the spiking neuron enabled a less complex development of these models.

- *Development of SNN software platform*

  Computer programs were written in C++ in object oriented approach to develop the SNN software platform for realising the spiking neural network and learning models. Simulation and analysis of the network and the learning models for data clustering and classification tasks can be performed easily with this software platform. This software can be easily modified for further study.

6.2 Conclusion

In recent years, temporal coding spiking neural networks have been receiving wider attention. Previous spiking neural network models applied for cluster-

ing or classification tasks utilised a weight adaptation mechanism, where connections with appropriate delays were selected for encoding the input information. The utilisation of multiple sub-connections and the need for a population coding scheme for higher accuracy resulted in increased network size and computational effort. This study investigated the learning in SNNs through weight adaptation and delay adaptation independently, utilising a temporal coding scheme which encodes input values through single spike times. As a result, three novel networks, namely, Self-Organising Weight Adaptation Spiking Neural Network (SOWA_SNN), Self-Organising Delay Adaptation Spiking Neural Network (SODA_SNN) and Supervised Delay Adaptation Spiking Neural Network (SDA_SNN), have been proposed.

The performances of the three temporal coding SNN models were found to be better than that of previous spiking neural network models. This finding shows that the spiking neural network can learn successfully through adapting weights as well as through delays. In addition, the successful application of the proposed networks for clustering and classification tasks further strengthens the case for the use of precise spike timings for computation and communication within the neural networks.

The performances of all the three proposed models were found to be better than that of previous spiking neural network models. Among the two proposed unsupervised models, the SODA_SNN's capability was observed to be better than the SOWA_SNN in terms of clustering accuracy and cluster formation. Hence, in terms of pattern detection, the capability of the coincidence detecting spiking neurons can be considered to be higher than that of the neurons as integrators.

The performances of the proposed models were demonstrated to be better than that of conventional neural network models. The SOWA_SNN was able to

cluster the analysed test data sets, including the high-dimensional Control chart data, with higher accuracy than that of the Kohonen's SOM. The SODA_SNN was able to obtain better clustering accuracy than the Kohonen's SOM for all of the analysed training and testing data sets. The SDA_SNN obtained better classification accuracies for all the analysed test data compared to that of the MLP networks, with significantly smaller network size and fewer training epochs. These results show that the temporal coding spiking neural networks are capable of out performing the conventional neural networks in the three proposed learning paradigms.

It can be concluded that the proposed SNN models can be considered as alternative tools for clustering and classification tasks instead of the existing spiking neural network models. The same applies to the conventional artificial neural network paradigms. The results of this study could be considered as an incremental improvement in the research on learning with spiking neural networks.

6.3 Future work

This section specifies a significant number of indicators for further research which can lead to better learning models with temporal coding spiking neural networks. A major area of interest could be the coding of continuous and nominal input values into spike codes and vice versa. The coding scheme utilised in this research is a linear method which converts the input values into discrete spike events. This linear method reduces the accuracy of the temporal inputs. A non-linear method could therefore be utilised as an alternative to improve the temporal coding scheme [Bohte et al., 2002a]. A complex population coding scheme can be found in the literature but with increased computational effort. Implementation of a better coding strategy can increase the clustering and classification capability

of the models even further.

The proposed models either adapt connection weights or connection delays. This research can be extended to incorporate both strategies into a single model. Since both the connection delays and weights play a significant role in the network functioning, a combined approach could produce a better learning model for spiking neural network.

The proposed models could be utilised for efficient feature selection. It was observed on the trained models that not all the input parameters were fully contributing to the output in some cases. Further study can reveal more on the applicability of the proposed models for the selection of dominating features among the input parameters.

The proposed models have been confined to those with simple design features in order to get a better understanding of the spiking neurons and the learning models. The efficiency of the models could be increased by adding special features such as momentum term for the learning rule, complex output grids and neighbourhood functions for unsupervised learning models and efficient terminating criteria.

The proposed models have been developed specifically for clustering and classification tasks. However, the strategies incorporated are not application specific and they can therefore be utilised to develop models in other application areas.

Bibliography

Abbott, L. F. and Nelson, S. B. (2000). Synaptic Plasticity: Taming the beast. *Nature Neuroscience*, 3(11s):1178–1183.

Abeles, M. (1982). Role of the cortical neuron: Integrator or coincidence detector? *Israel Journal of Medical Science*, 18(1):83–92.

Baldi, P. and Atiya, A. F. (1994). How delays affect neural dynamics and learning. *IEEE Transactions on Neural Networks*, 5(4):612–621.

Belatreche, A., Maguire, L. P., McGinnity, M., and Wu, Q. X. (2003). An evolutionary strategy for supervised training of biologically plausible neural networks. In *Proceedings of the sixth international conference on computational intelligence and natural computing*, pages 1524–1527, Cary, North Carolina, USA.

Bell, C. C., Han, V. Z., Sugawara, Y., and Grant, K. (1997). Synaptic plasticity in cerebellum-like structure depends on temporal order. *Nature*, 387(6630):278–281.

Bi, G. Q. and Poo, M. M. (1998). Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and post synaptic cell type. *Journal of Neuroscience*, 18(24):10464–10472.

Blum, K. I. and Abbott, L. F. (1996). A model of spatial map formation in the hippocampus of the rat. *Neural Computation*, 8(1):85–93.

Bohte, S. M. (2003). *Spiking Neural Networks.* PhD thesis, University of Leiden.

Bohte, S. M. and Kok, J. N. (2005). Applications of spiking neural networks. *Information Processing Letters*, 95:519–520.

Bohte, S. M., Kok, J. N., and La Poutré, H. (2002a). Error-back propagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48(1-4):17–37.

Bohte, S. M., La Poutré, H., and Kok, J. N. (2002b). Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer RBF networks. *IEEE Transactions on Neural Networks*, 13(2):426–435.

Bower, J. (1991). Exploring biological neural networks using realistic computer simulations. *Naval Research Reviews*, 43:17–22.

Carla, J. S. (1990). Impulsive activity and the patterning of connections during cns development. *Neuron*, 5(6):745–756.

Carnell, A. and Richardson, D. (2005). Linear algebra for time series of spikes. In *Proceedings of the 13th European Symposium on Artificial Neural Networks, 2005*, Bruges, Belgium.

Carr, C. E. (1993). Processing of temporal information in the brain. *Annual Review of Neuroscience*, 16:223–243.

Day, S. P. and Camporese, P. S. (1991). Continuous-time temporal back-propagation. In *Proceedings of the International Joint Conference on Neural Networks, 1991*, volume 2, pages 95–100, Seattle, USA.

Day, S. P. and Davenport, M. R. (1993). Continuous-time temporal back-propagation with adaptable time delays. *IEEE Transactions on Neural Networks*, 4(2):348–354.

Dayan, P. and Abbott, L. F. (2001). *Theoretical Neuroscience.* The MIT Press.

Debanne, D., Gähwiler, B. H., and Thompson, S. M. (1996). Cooperative interactions in the induction of long-term potentiation and depression of synaptic excitation between hippocampal CA3-CA1 cell pairs in vitro. *Proceedings of the National Academy of Sciences of the United States of America*, 93(20):11225–11230.

Eurich, C. W., Pawelzik, K., Ernst, U., Cowan, J. D., and Milton, J. (1999). Dynamics of self-organized delay adaptation. *Physical Review Letters*, 82(7):1594–1597.

Eurich, C. W., Pawelzik, K., Ernst, U., Thiel, A., Cowan, J. D., and Milton, J. G. (2000). Delay adaptation in nervous systems. *Neurocomputing*, 32-33:741–748.

Gaiarsa, J. L., Caillard, O., and Ben-Ari, Y. (2002). Long-term plasticity at GABAergic and glycinergic synapses: Mechanisms and functional significance. *Trends in Neurosciences*, 25(11):564–570.

Gerstner, W. (2001). Spiking neurons. In Maass, W. and Bishop, C. M., editors, *Pulsed Neural Networks*, pages 3–53. The MIT Press, Cambridge, first edition.

Gerstner, W., Kempter, R., van Hemmen, J. L., and Wagner, H. (1996). Neuronal learning rule for sub-millisecond temporal coding. *Nature*, 383(6595):76–78.

Gerstner, W. and Kistler, W. M. (2002). *Spiking Neuron Models.* Cambridge University Press, Cambridge, UK, first edition.

Habberly, L. B. (1985). Neuronal circuitry in olfactory cortex: Anatomy and functional implication. *Chemical Senses*, 10(2):219–238.

Haykin, S. (1999). *Neural Networks - A comprehensive foundation.* Prentice hall, New Jersey, second edition.

Hodgkin, A. L. and Huxley, A. F. (1952). A quantitative description of ion currents and its application to conduction and excitation in nerve. *Journal of Physiology*, 117(4):500–544.

Hopfield, J. J. (1995). Pattern recognition computation using action potential timing for stimulus representation. *Nature*, 376(6535):33–36.

Iannella, N. and Back, A. D. (2001). A spiking neural network architecture for nonlinear function approximation. *Neural Networks*, 14(6-7):933–939.

Ienne, P. (1997). Digital connectionist hardware: Current problems and future challenges. In Jos Mira, R. M.-D. and Cabestany, J., editors, *Biological and Artificial Computation: From Neuroscience to Technology (volume 1240 of Lecture Notes in Computer Science)*, pages 688–713. Springer, Berlin.

Innocenti, G. M., Lehmann, P., and Houzel, J. C. (1994). Computational structure of visual collosal axons. *The European journal of neuroscience*, 6(6):918–935.

Jahnke, A., Roth, U., and Schönauer, T. (2001). Digital simulation of spiking neural networks. In Maass, W. and Bishop, C. M., editors, *Pulsed Neural Networks*, pages 237–257. The MIT Press, Cambridge, Massachusetts, first edition.

Jain, A. K., Mao, J., and Mohiuddin, K. M. (1996). Artificial neural networks: A tutorial. *IEEE Computer*, 29(3):31–44.

Jeffress, L. A. (1948). A place theory of sound localisation. *Journal of comparative and physiological psychology*, 41:35–39.

Jianguo, X. and Embrechts, M. J. (2001). Supervised learning with spiking neural networks. In *Proceedings of International Joint Conference on Neural Networks IJCNN '01*, volume 3, pages 1772–1777, Washington, DC, USA. IEEE.

Kasisnski, A. and Ponulak, F. (2006). Comparison of supervised learning methods for spike time coding in spiking neural networks. *International Journal of Applied Mathematics and Computer Science*, 16(1):101–113.

Kistler, W. M. and van Hemmen, J. L. (2000). Modeling synaptic plasticity in conjunction with the timing of pre- and postsynaptic action potentials. *Neural Computation*, 12(2):385–405.

Koch, R. and Grover, M. (2002). Amygdala, a spiking neural network library.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69.

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480.

Konig, P., Engel, A. K., and Singer, W. (1996). Integrator or coincidence detector? The role of the cortical neuron revisited. *Trends in Neurosciences*, 19(4):130–137.

Lapedes, A. and Farber, R. (1987). Nonlinear signal processing using neural network: Prediction and system modelling. Technical report, Los Alamos National Laboratory, Los Alamos, NM, USA.

Maass, W. (1996). Lower bounds for the computational power of networks of spiking neurons. *Neural Computation*, 8(1):1–40.

Maass, W. (1997a). Fast sigmoidal networks via spiking neurons. *Neural Computation*, 9(2):279–304.

Maass, W. (1997b). Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671.

Maass, W. (1997c). Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons. In Mozer, M., Jordan, M. I., and Petsche, T., editors, *Advances in Neural Information Processing Systems, volume 9*, pages 211–217. The MIT Press, Cambridge, Massachusetts.

Maass, W. (2001a). Computing with spiking neurons. In Maass, W. and Bishop, C. M., editors, *Pulsed Neural Networks*, pages 55–85. The MIT Press, Cambridge, Massachusetts, first edition.

Maass, W. (2001b). Neural computation: a research topic for theoretical computer science? Some thoughts and pointers. In Rozenberg, G., Salomaa, A., and Paun, G., editors, *Current Trends in Theoretical Computer Science, Entering the 21st Century*, pages 680–690. World Scientific Publishing.

Maass, W. and Schmitt, M. (1999). On the complexity of learning for spiking neurons with temporal coding. *Information and Computation*, 153(1):26–46.

Marian, I. (2001). SpikeNNS - a simulator for spiking neural networks. Technical report, Department of Computer Science, National University of Ireland, Maynooth, Ireland.

Markram, H., Lübke, J., Frotscher, M., and Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of post synaptic APs and EPSPs. *Science*, 275(5297):213–215.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133.

Mendel, J. M. and McLaren, R. W. (1970). Reinforcement-learning control and pattern recognition systems. In Mendel, J. and Fu, K., editors, *Adaptive, Learning and Pattern Recognition Systems: Theory and Applications*, pages 287–318. Academic Press, New York.

Miller, K. D. (1996). Synaptic economics: Competition and cooperation in synaptic plasticity. *Neuron*, 17(3):371–374.

Minsky, M. L. and Papert, S. A. (1969). *Perceptrons*. The MIT Press, Cambridge, Massachusetts.

Natschläger, T. and Ruf, B. (1998). Spatial and temporal pattern analysis via spiking neurons. *Network: Computational Neural Systems*, 9(3):319–332.

Newman, D. J., Hettich, S., Blake, C. L., and Merz, C. J. (1998). UCI repository of machine learning databases.

O'Keefe, J. and Reece, M. L. (1993). Phase relationship between hippocampal place units and EEG theta rhythm. *Hippocampus*, 3(3):317–330.

Orr, M. J. L. (1996). Introduction to RBF networks. Technical report, Centre for cognitive science, University of Edinburg, Edinburg, UK.

Panchev, C. and Wermter, S. (2001). Hebbian spike-timing dependent self-organization in pulsed neural networks. In *Proceedings of the world congress on Neuroinformatics*, Vienna, Austria.

Pfister, J.-P., Barber, D., and Gerstner, W. (2003). Optimal hebbian learning: A probabilistic point of view. In Kaynak, O., Alpaydin, E., Oja, E., and Xu, L., editors, *ICANN*, volume 2714 of *Lecture Notes in Computer Science*, pages 92–98. Springer.

Pfister, J.-P., Toyoizumi, T., Barber, D., and Gerstner, W. (2006). Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning. *Neural Computation*, 18(6):1318–1348.

Pham, D. T. and Liu, X. (1999). *Neural Networks for Identification Prediction and Control*. Springer-Verlag, London, Great Britain, second edition.

Pham, D. T. and Sagiroglu, S. (2001). Training multilayered perceptrons for pattern recognition: A comparative study of four training algorithms. *International Journal of Machine Tools and Manufacture*, 41(3):419–430.

Ponulak, F. (2005). ReSuMe - new supervised learning method for spiking neural networks. Technical report, Institute of Control and Information Engineering, Poznan University of Technology, Polland.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

Ruf, B. and Schmitt, M. (1997). Learning temporally encoded patterns in networks of spiking neurons. *Neural Processing Letters*, 5(1):9–18.

Ruf, B. and Schmitt, M. (1998). Self-organization of spiking neurons using action potential timing. *IEEE Transactions on Neural Networks*, 9(3):575–578.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by back-propagating errors. *Nature*, 323:533–536.

Schrauwen, B. and VanCampenhout, J. (2004). Improving SpikeProp: Enhancements to an error-backpropagation rule for spiking neural networks. In *Proceedings of the 15th ProRISC Workshop.*

Sejnowski, T. J. (1977). Storing covariance with nonlinearly interacting neurons. *Journal of Mathematical biology*, 4(4):303–321.

Shepherd, G. M. and Koch, C. (1990). Introduction to synaptic circuits. In Shepherd, G. M., editor, *The synaptic organization of the brain*, pages 3–31. Oxford university press, New York.

Shi, S. H., Hayashi, Y., Petralia, R. S., Zaman, S. H., Wenthold, R., Svoboda, K., and Malinow, R. (1999). Rapid spine delivery and redistribution of AMPA

receptors after synaptic NMDA receptor activation. *Science*, 284(5421):1811–1816.

Smith, L. S. (2004). Spiking neural network simulator: User's guide. Technical report, Department of Computing Science and Mathematics, University of Stirling, Stirling, Scotland, UK.

Song, S., Miller, K. D., and Abbott, L. F. (2000). Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nature Neuroscience*, 3(9):919–926.

Stanford, L. R. (1987). Conduction velocity variations minimize conduction time differences among retinal ganglion cell axons. *Science*, 238(4825):358–360.

Thorpe, S. J. (1990). Spike arrival times: A highly efficient coding scheme for neural networks. In Eckmiller, R., Hartmaan, G., and Hauske, G., editors, *Parallel processing in neural systems*, pages 91–94. Elsevier, North-Holland.

Thorpe, S. J., Delorme, A., and van Rullen, R. (2001). Spike-based strategies for rapid processing. *Neural Networks*, 14(6-7):715–725.

Thorpe, S. J., Fize, D., and Marlot, C. (1996). Speed of processing in the human visual system. *Nature*, 381(6582):520–522.

Thorpe, S. J. and Imbert, M. (1989). Biological constraints on connectionist models. In Pfeifer, R., Schreter, Z., Fogelman-Soulié, F., and Steels, L., editors, *Connectionism in Perspective*, pages 63–92. Elsevier, Amsterdam, second edition.

Tversky, T. and Miikkulainen, R. (2002). Modeling directional selectivity using self-organizing delay-adaptation maps. *Neurocomputing*, 44-46:679–684.

Ultsch, A. and Möerchen, F. (2005). ESOM-Maps: Tools for clustering, visualization, and classification with emergent SOM. Technical Report No. 46,

Dept. of Mathematics and Computer Science, University of Marburg, Marburg, Germany.

van Rossum, M. C. W., Bi, G. Q., and Turrigiano, G. G. (2000). Stable Hebbian learning from spike timing-dependent plasticity. *Journal of Neuroscience*, 20(23):8812–8821.

Vesanto, J. and Alhoniemi, E. (2000). Clustering of the self-organising map. *IEEE Transactions on Neural Networks*, 11(3):586–600.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. J. (1989). Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(3):328–339.

Wan, E. A. (1990). Temporal backpropagation for FIR neural networks. *Proceedings of IEEE International Joint Conference on Neural Networks*, 1:575–580.

Weigend, A. S., Huberman, B. A., and Rumelhart, D. E. (1990). Predicting the Future: a Connectionist Approach. *International Journal of Neural Systems*, 1(3):193–209.

Widrow, B. and Hoff, M. E. J. (1960). Adaptive switching circuits. *IRE Western Electric Show and Convention Record, Part 4*, pages 96–104.

Widrow, B. and Winter, R. (1988). Neural nets for adaptive filtering and adaptive pattern recognition. *Computer*, 21(3):25–39.

Willshaw, D. J. and von der Malsburg, C. (1976). How patterned neural connections can be set up by self-organisation. In *Proceedings of the Royal Society of London, Series B*, volume 194, pages 431–445.

Xie, X. and Seung, H. S. (2004). Learning in neural networks by reinforcement of irregular spiking. *Physical Review E*, 69(4):Id. 041909.

Zador, A. M. (2000). The basic unit of computation. *Nature neuroscience*, Supplement 3:1167.

Zell, A., Mache, N., Huebner, R., Schmalzl, M., Sommer, T., and Korb, T. (1992). SNNS: Stuttgart Neural Network Simulator. Technical report, Institute for Parallel and Distributed High Performance Systems (IPVR), University of Stuttgart, Stuttgart.

Zurada, J. M. (1999). *Introduction to Artificial Neural Systems.* West Publishing Company, St. Paul, USA.

Appendix A

Program source code

This appendix provides comprehended source code of the software developed to implement the proposed learning models. In the proposed models the spiking neurons were implemented in two modes of operation namely integration and coincidence detection. Realisation of the two modes of operation depends only in the assignment of correct parameter values. Hence much of the source code for implementation of the weight and delay adaptation models is similar. Since self-organising weight and delay adaptation models were based on Kohonen's SOM, the programs for both modes are similar except the learning procedures. The program for the SDA_SNN was designed similar to the previous two models in a way to reuse the source code.

This appendix is separated into several sections. Section A.1 gives the definitions of classes *Vector* and *matrix*. spikeNN is the main class which handles the implementation of the spiking neural network and learning procedures. Section A.2 gives the code for implementing SOWA_SNN and SODA_SNN. This includes the major operations of the network and the code for applying the models to cluster data sets. Section A.3 provides the code for implementing the SDA_SNN. Most of the names chosen for the methods and variables are self explanatory and do exactly same as the name suggests.

A.1 Class definitions

A.1.1 Definition of class *Vector*

```
#ifndef VECTOR_H
#define VECTOR_H

typedef double eltype;

class Vector{
 private:
  int nelt;
  double *velt;
 public:
  Vector(int);
```

```
  ~Vector();

  //Set all elements to val
  void initialise(double val);

  //length of the vector
  int getcnt(){return nelt;};

  //read element i
  double getelt(int i){return velt[i-1];};

  //store e for element at i
  void setelt(int i, double e){velt[i-1]=e;};

  double min();
  double max();
  double sum();
  double average();
  double stdev();
  void print_Vector();

  //normalise the vector
  void normalize();

  //To write a vector into a file
  int write_Vector_to_file(const char*,int);
  //File name, open type(write 1 /append 2)

  //To read a vector from a file
  int read_Vector_from_file(const char*);

  //fill the vector with values
  //increasing from min to max
  void fill_inc(int seed, int min, int max);

  //fill the vector randomly
  void fill_random(int seed);

  //multiply the elements with m
  void multiply(double m);

  //swap elements i and j
  void swap_elts(int i, int j);

  //shuffle elemenst for n time
  void shuffle(int);

  //copy the vector onto
  //another vector cv
  void copy_vector(Vector cv);
};

#endif
```

A.1.2 Definition of class *matrix*

```
#ifndef MATRIX_H
```

```
#define MATRIX_H
#include "vector.h"

class matrix{
private:
  int rows;
  int cols;
  double *melt;
public:
  matrix(int,int);
  ~matrix();
  int index(int i,int j){return (i-1)*cols+j-1;};
  void initmatrix(double type);
  int nrows(){return rows;};
  int ncols(){return cols;};
  double getelt(int,int);
  void setelt(int, int , double);
  void fill_random(int seed, double mn, double mx);
  void print_matrix();

  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 normalize_column();
  void normalize_row();

  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);
  double average(int k);
  int elements_below(float e, int k);
};

#endif
```

A.1.3 Definition of class *spikeNN*

```
#include "matrix.h"
#include "matrix3d.h"

class spikeNN{
protected:
```

```
int N_record, N_attrib, N_target_attrib;
int N_train, N_test, N_class, N_sample, N_bias;
int max_epocs, cont_learn;

int N_input, N_output, N_rows, N_cols;

matrix *source_data;
Vector *source_target;
matrix *processed_data;

matrix *train_matrix, *train_target;
matrix *test_matrix, *test_target;
Vector *test_data_count;

double maxw, minw, maxd, mind, dbias, shift;
Vector *threshold;
Vector *pspmax;

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

double learning_rate0, learning_rate;
double sigma_w0, sigma_w;

double max_threshold, min_threshold;

double total_error, min_error, tolerance;
double total_weight_change, prev_weight_change;
double total_dlay_change, prev_dlay_change;
double nei_poten, nei_dep;
int precision;
Vector *inpv, *tv;

int order_by_time, order_by_position;
int vary_neighborhood, latteral_effect;
double winner_bias_chg, looser_bias_chg;

//Synaptic weights - N_input X N_output
matrix *wght;     //Connection weights
matrix *dwght;    //weight change
matrix *delay;    //Connection delays
matrix *ddelay;   //delay change

double tc;   //time constant for the spike response fn.
double tcpd;. //time constant for the learning rule

//to store synapse potential
// N_input X N_output
matrix *sp;
Vector *np;        //Neuron(soma) potential - N_output
Vector *op;        //Network output - N_output
Vector *exact_op; //Exact Output - N_output

//to store test output N_test X N_output
matrix *test_output;
matrix *test_winner;
```

```
  matrix *train_winner;
  Vector *coincidence_measure;

public:
  spikeNN();
  ~spikeNN();
  void createNN();
  void init_spikeNN();
  void init_parameters();

  void update_synapse_potential(int);
  void update_output(int);
  double spike_response(int t);
  void simulate_SNN();
  void find_winner();
  void find_exact_winner();

  void select_data();

  float dist(int, int);
  void set_threshold_fixed(double);
  double get_threshold_min();
  double get_threshold_max();
  void set_threshold(int);
  void set_learning_rate(int epocNo);
  double tpl_nehbr(int xi);
  void set_sigma(int epocNo);
  void compute_coincidence_measure();

  void train_spikeNN();
  void update_dlay(int index);
  double dlay_change(int index);

  void test_train_set(const char *opt_test);
  void test_spikeNN(const char *opt_train);

};
```

A.2 Source code for implementing the unsupervised models

A.2.1 Source code for implementing the SNN

```
#include "spikenn.h"
# include <iostream>
# include <math.h>
# include <stdlib.h>
# include <time.h>
# include <fstream>
#include "utils.h"

using namespace std;

spikeNN::spikeNN()
{
```

```
}

spikeNN::~spikeNN()
{
}

void spikeNN::createNN()
{

  source_data=new matrix(N_record,N_attrib);
  source_target=new Vector(N_record);
  processed_data=new matrix(N_record, N_input);

  //Matrix to contain the training records;
  train_matrix=new matrix(N_train,N_input);
  train_target=new matrix(N_train,N_target_attrib);

  //Matrix to contain the test records;
  test_matrix=new matrix(N_test,N_input);
  test_target=new matrix(N_test,N_target_attrib);

  //count data from each class
  test_data_count=new Vector(N_class);

  //input vector
  inpv =new Vector(N_input);
  //target vector
  tv=new Vector(N_class);

  //matrix to hold connection weights
  wght =new matrix(N_input,N_output);
  //matrix to hold changes to connection weights
  dwght =new matrix(N_input,N_output);

  //Matrix to store connection delays
  delay=new matrix(N_input,N_output);
  //matrix to store connectio delay change
  ddelay=new matrix(N_input,N_output);

  //vector to store threshold of each output neuron
  threshold=new Vector(N_output);

  //Matrix to store the synapse (connection) potentials
  sp=new matrix(N_input,N_output);

  //Vector to store neuron potentials
  np=new Vector(N_output);

  //Vector to store ouput
  op=new Vector(N_output);

  //Exact output for tie breaking
  exact_op=new Vector(N_output);

  coincidence_measure=new Vector(N_output);
}
```

```
void spikeNN::init_parameters()
{
//Initialise parameters such as size, epochs, etc
//to be defined in the child_class
}

void spikeNN::init_spikeNN()
//Initialise the network before each cycle of learning
{
  sp->initmatrix(0.0);

  np->initialise(0.0);

  op->initialise(twindow);

  exact_op->initialise(twindow);

  dwght->initmatrix(0.0);
  ddelay->initmatrix(0.0);
}

void spikeNN::set_threshold_fixed(double max_threshold)
{
  for(int i=1;i<=N_output;i++)
    threshold->setelt(i,max_threshold);
}

double spikeNN::spike_response(int st)
//spike response function
{
  double val=0.0;
         if (st>0) val=(st/tce)*exp(1-(st/tce));
  return val;
}

void spikeNN::update_synapse_potential(int st)
//update the potential of each connection
{
  double inpT,dlay,spikeT,spoten;

  for(int i=1;i<=N_input;i++){
     inpT=inpv->getelt(i);
     if ((inpT<0)||(inpT>twindow_input)) continue;
       for(int j=1;j<=N_output;j++){
          if (np->getelt(j)<0.0) continue;
          dlay=delay->getelt(i,j);
          spikeT=st-inpT-dlay;
          if (spikeT<0) continue;
          spoten=spike_response((int)spikeT)*wght->getelt(i,j);
          sp->setelt(i,j,spoten);
       }//end for j
  }//end for i
}

void spikeNN::update_output(int st)
//sum up the potential from each connection
{
   double sv,pnp,spike_time;
```

```
   for(int i=1;i<=N_output;i++){
    if(np->getelt(i)<0) continue;
      sv=0.0;
      for(int j=1;j<=N_input;j++)
          sv=sv+sp->getelt(j,i);
      pnp=np->getelt(i);
      np->setelt(i,sv);
      if(sv>=threshold->getelt(i)) {
        spike_time=(threshold->getelt(i)-pnp)/(sv-pnp);
        op->setelt(i,st);
        exact_op->setelt(i,st-1+spike_time);

        np->setelt(i,-1000);
        for(int j=1;j<=N_input;j++)
            sp->setelt(j,i,0);
      }
  }
}

void spikeNN::simulate_SNN()
{
    int st=(int)inpv->min();
    do{
       st=st+dt;
        update_synapse_potential(st);
        update_output(st);
    }while(st<twindow);

}
```

A.2.2 Source code for implementing SOWA_SNN

```
void  spikeNN::train_spikeNN()
//code for the learning procedure
{
  int epoc;
  total_weight_change=0.0;

  int winner_row, winner_col;
  int xindexs, yindexs;
  int xindexe, yindexe, index ;

  double nei_dist;
  epoc=0;

//repeat until the terminating
//criteria met
  do{
    epoc++;
    cont_learn=1;
    nei_poten=0.0; nei_dep=0.0; dep=0.0;
    total_weight_change=0.0;
    prev_weight_change=0.0;
    set_sigma(epoc);
    //set_learning_rate(epoc);
```

```
//repeat for all the training samples
    for(int r=1;r<=N_train;r++){
      //initialise the network
      init_spikeNN();
      //get a training vector
      train_matrix->get_row_Vector(r,1,N_input,*inpv);
      winner_found=0;
      //simulate the network with the input
      simulate_SNN();
      //find the winner
      find_winner();

      if (winner_found) {
//Find the neighbourhood of cooperation
                winner_row=winner/N_cols+1;
                winner_col=winner%N_cols;
                if (winner_col==0){
                   winner_col=N_cols;
                   winner_row--;
                }

                nei_dist=(int)sigma_w;
                xindexs=winner_row-nei_dist;
                yindexs=winner_col-nei_dist;

                if (xindexs<=0) xindexs=1;
                if (yindexs<=0) yindexs=1;

                xindexe=winner_row+nei_dist;
                yindexe=winner_col+nei_dist;

                if (xindexe>N_rows) xindexe=N_rows;
                if (yindexe>N_cols) yindexe=N_cols;

//update all neurons in the neighbourhood
                for(int i=xindexs;i<=xindexe;i++)
                        for(int j=yindexs;j<=yindexe;j++)
                        {
                                index=(i-1)*N_cols+j;
                                update_weight(index);
                        }
          }
    total_weight_change+=wght_change();

    }
    cout<<"epoc :"<<epoc<<endl;
    cout<<" weight_change:"<<endl;
    cout<<"Neigh_poten :"<<nei_poten<<endl;
    cout<<"Neigh_depression :"<<nei_dep<<endl;

   cont_learn=(total_weight_change>tolerance);
   prev_weight_change=total_weight_change;
  }while((epoc<max_epocs) && (cont_learn));
}

double spikeNN::wght_change()
{
double twc=0.0;
```

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

 return (twc);
}

void spikeNN::update_weight(int n)
//update the connection weights of neuron n
{
  double input,weight,newweight,dw,deltaT,dlay;
  double wf,output,v,nf,tf;

    output=op->getelt(n);
    for(int i=1;i<=N_attrib;i++){
      input=inpv->getelt(i);
 if ((input<0)||(input>(twindow_input))||(output>=twindow))
                continue;
      dlay=delay->getelt(i,n);
      deltaT=output-dlay-input;
      weight=wght->getelt(i,n);
      wf=0.0;
      dw=0.0;
      newweight=0.0;
      v=0.0;

      if (deltaT>=0){
        wf=(maxw-weight)/maxw;
                        nf=tpl_nehbr(n);
                        tf=exp(-deltaT/tcpd)-ffact;
        dw=learning_rate*wf*nf*tf;
        nei_poten+=mod(dw);
      }

              if (deltaT<0){
        wf=0.5;
                        nf=tpl_nehbr(n);
                        tf=exp(deltaT/tcpd);
        dw=-learning_rate*wf*nf*tf;
        nei_dep+=mod(dw);
      }

      newweight=weight+dw;
      dwght->setelt(i,n,dw);
      wght->setelt(i,n,newweight);
   }
}
```

A.2.3 Source code for implementing SODA_SNN

```
void spikeNN::compute_coincidence_measure()
{
 Vector temp(N_input);

 int inpt, dlay, mx, mn, diff;
 double cf=1;
```

```
 for(int j=1;j<=N_output;j++){
        for(int i=1;i<=N_input;i++){
                inpt=(int)inpv->getelt(i);
                dlay=(int)delay->getelt(i,j);
                temp.setelt(i, inpt+dlay);
        }

        mx=temp.max();
        mn=temp.min();
        diff=mx-mn;
        cf=diff/(double)twindow_input;
        coincidence_measure->setelt(j,cf);
 }
}

void  spikeNN::train_spikeNN()
//train the network
{

  int epoc;
  int winner_row, winner_col;
  int nei_dist, xindexs, xindexe;
  int yindexs, yindexe, index;
  total_dlay_change=0.0;

  epoc=0;

  do{
    epoc++;
    cont_learn=1;
    nei_poten=0.0; nei_dep=0.0; dep=0.0;
    total_dlay_change=0.0;
    prev_dlay_change=0.0;
    set_sigma(epoc);
    set_learning_rate(epoc);
    set_threshold(epoc);
    //set_threshold_fixed(get_threshold_min());

    for(int r=1;r<=N_train;r++){
      init_spikeNN();
      train_matrix->get_row_Vector(r,1,N_input,*inpv);
      inpv->write_Vector_to_file(outputfile,2);

      winner_found=0;

      simulate_SNN();
      find_winner();
      compute_coincidence_measure();
      if (winner_found) {

                winner_row=winner/N_cols+1;
                winner_col=winner%N_cols;
                if (winner_col==0){
                   winner_col=N_cols;
                   winner_row--;
                }
```

```
                nei_dist=(int)sigma_w;
                xindexs=winner_row-nei_dist;
                yindexs=winner_col-nei_dist;

                if (xindexs<=0) xindexs=1;
                if (yindexs<=0) yindexs=1;

                xindexe=winner_row+nei_dist;
                yindexe=winner_col+nei_dist;

                if (xindexe>N_rows) xindexe=N_rows;
                if (yindexe>N_cols) yindexe=N_cols;

                for(int i=xindexs;i<=xindexe;i++)
                        for(int j=yindexs;j<=yindexe;j++)
                        {
                                index=(i-1)*N_cols+j;
        update_dlay(index);
        total_dlay_change+=dlay_change(index);
      }
          }

    }
    cout<<"epoc :"<<epoc<<endl;
    cout<<" Delay change:"<<endl;
    cout<<"Neigh poten :"<<nei_poten<<endl;
    cout<<"Neigh depression :"<<nei_dep<<endl;
    cout<<"Total :"<<total_dlay_change<<endl;

    //cont_learn=((total_wght_change-prev_wght_change)>tolerance);
   cont_learn=(total_dlay_change>tolerance);
   prev_dlay_change=total_dlay_change;
  }while((epoc<max_epocs) && (cont_learn));
}

void spikeNN::update_dlay(int index)
//update the connection delays
//of neuron index
{
  double input,deltaT,dd,dlay,new_dlay;
  double cf,output,v;//,nf,ntc;
  double max_delay=maxd;
  double min_delay=mind;

  double dn, dtt;

    ddelay->initmatrix(0.0);

    output=op->getelt(index);
    for(int i=1;i<=N_input;i++){
      input=inpv->getelt(i);
      if ((input<0)||(output>=twindow))
                        continue;
      dlay=delay->getelt(i,index);
      deltaT=output-dlay-input-shift;

      cf=coincidence_measure->getelt(index);
      dd=0.0;
```

```
        new_dlay=0.0;
        v=0.0;

        if (deltaT>=0){
            dn=tpl_nehbr(index);
            dtt=deltaT*(exp(-deltaT*deltaT/(tcpd*tcpd))/tcpd)-dbias;
            dd=learning_rate*dn*dtt*cf;
            nei_poten+=mod(dd);
         }

         if (deltaT<0){
            dn=tpl_nehbr(index);
            dtt=deltaT*(exp(-deltaT*deltaT/(tcpd*tcpd))/tcpd)-dbias;
            dd=learning_rate*dtt*dn*cf;
                nei_dep+=mod(dd);
         }
         new_dlay=dlay+dd;
         if (new_dlay>max_delay) new_dlay=max_delay;
         if (new_dlay<min_delay) new_dlay=min_delay;
         ddelay->setelt(i,index,dd);
         delay->setelt(i,index,new_dlay);
          }
}
```

A.2.4 Section of class spikeNN for testing the data sets

```
void spikeNN::test_spikeNN(const char *opt_test)
//store the output of the network for test data
{
  int st, wx, wy;
  matrix data_store(N_test,5);
  matrix op_whole(N_input, N_output);

  test_matrix->write_matrix_to_file(opt_test,1);
  wght->write_matrix_to_file(opt_test,2);
  delay->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_input,*inpv);
    test_target->get_row_Vector(r,1,N_class,*tv);

    st=(int)inpv->min();
    do{
      st=st+dt;
      update_synapse_potential(st);
      update_output(st);
    }while(st<twindow);

    find_exact_winner();
    wx=winner/N_cols+1;
    wy=winner%N_cols;
    if (wy==0) {
       wy=N_cols;
       wx=wx-1;
    }
    data_store.setelt(r,5,test_target->getelt(r,1));
    data_store.setelt(r,1,wx);
```

```
    data_store.setelt(r,2,wy);
    data_store.setelt(r,3,winning_time);
    data_store.setelt(r,4,winner);

  }
   op_whole.write_matrix_to_file(opt_test,2);
   data_store.write_matrix_to_file(opt_test,2);
}

void spikeNN::test_train_set(const char *opt_train)
//store the output of the network for train data
{
  int wx, wy, inpt, dlay, st;
  matrix eff_inpt(N_train, N_input);

  matrix data_store(N_train,5);
  wght->write_matrix_to_file(opt_train,1);
  delay->write_matrix_to_file(opt_train,2);
  train_matrix->write_matrix_to_file(opt_train,2);
  for(int r=1;r<=N_train;r++){
    init_spikeNN();
    train_matrix->get_row_Vector(r,1,N_input,*inpv);

    st=(int)inpv->min();
    do{
      st=st+dt;
      update_synapse_potential(st);
      update_output(st);
    }while(st<twindow);

    op->write_Vector_to_file(opt_train,2);
    find_exact_winner();

    wx=winner/N_cols+1;
    wy=winner%N_cols;
    if (wy==0) {
       wy=N_cols;
       wx=wx-1;
    }
    data_store.setelt(r,5,train_target->getelt(r,1));
    data_store.setelt(r,1,wx);
    data_store.setelt(r,2,wy);
    data_store.setelt(r,3,winning_time);
    data_store.setelt(r,4,winner);

   for(int i=1;i<=N_attrib;i++){
      inpt=(int)inpv->getelt(i);
      dlay=(int)delay->getelt(i,winner);
      eff_inpt.setelt(r, i, inpt+dlay);
   }

  }
  data_store.write_matrix_to_file("train_output.dat",1);
  data_store.write_matrix_to_file(opt_train,2);
  eff_inpt.write_matrix_to_file(opt_train,2);
}

void spikeNN::select_data()
```

```
//To select test and train data
//from the whole data set
{

  Vector svect(N_input);
  int seedgen,seed,i, st;

  seedgen=time(NULL)%1000;
  for(i=0;i<seedgen;i++)
    rand();

  seed=rand();
  srand(seed);

  int *choosed=new int[N_record+1];
  int row_no,cnt=1;

  for(i=1;i<=N_record;i++)
    choosed[i]=0;

  if (N_test) {
  do{
  row_no=(rand()%N_record)+1;
  if (!choosed[row_no]){
        st=source_target->getelt(row_no);
        if (test_data_count->getelt(st)) {
 processed_data->get_row_Vector(row_no,1,N_attrib,svect);
                if (N_bias) svect.setelt(N_input,1.0);
                test_matrix->set_row_Vector(cnt,1,N_input,svect);
          test_target->setelt(cnt,1,st);
 test_data_count->setelt(st, test_data_count->getelt(st)-1);
        choosed[row_no]=1;
          cnt++;
                }
  }
  }while(cnt<=N_test);
 }

  cnt=1;
  for(i=1;i<=N_record;i++)
    if(!choosed[i]){
      processed_data->get_row_Vector(i,1,N_attrib,svect);
      if (N_bias) svect.setelt(N_input,1.0);

      train_matrix->set_row_Vector(cnt,1,N_input,svect);
      train_target->setelt(cnt,1,source_target->getelt(i));
      cnt++;
    }
}
```

A.2.5 Code for clustering Wine data set with SOWA_SNN

```
#ifndef WINE_H
#define WINE_H

#include "spikenn.h"
```

```
class wine : public spikeNN{
private:

int N_sourceRow, N_sourceCol;

public:
     wine();
     ~wine();
void init();
void acquire_data();
void preprocess_data();
void shuffle_trainset();
void init_parameters();
void start();

};

#endif

#include "wine.h"
#include <math.h>
# include <iostream>
# include <stdlib.h>
# include <time.h>
# include <fstream>

wine::wine()
{
}

wine::~wine()
{
}

void wine::init()
{

  N_sourceRow=178;
  N_sourceCol=14;

  N_record=178;
  N_attrib=13;
  N_bias=0;
  N_target_attrib=1;
  N_train=120;
  N_test=58;

  N_input=N_attrib+N_bias;
  N_rows=4;
  N_cols=4;
  N_output=N_rows*N_cols;

  N_class=3;

  learning_rate=0.001;

  dt=1;
  timestep=1;
```

```
  twindow=65;
  early_fire=50;
  late_fire=60;
  twindow_input=30;

  tolerance=0.001;
  maxw=1.0;
  minw=0.0;
  maxd=30.0;
  mind=0.0;

  outputfile="iris_test_output.txt";

}

void wine::preprocess_data()
{
  int i,j;
  double target;

 source_data->copymatrix
        (1,1,N_record,N_attrib,*processed_data);

 processed_data->normalize_column();

 for(i=1;i<=N_record;i++)
  for(j=1;j<=N_attrib;j++){
    target=processed_data->getelt(i,j);
    target=twindow_input-(target*twindow_input);
    processed_data->setelt(i,j,target);
  }
}

void wine::shuffle_trainset()
{

  int seedgen=time(NULL)%1000;
    for(int i=0;i<seedgen;i++)
      rand();

  int nshuffle=N_train/3;
  int r1,r2,t1,t2;
  Vector tmptrv1(N_input);
  Vector tmptrv2(N_input);

  for(int cnt=0;cnt<=nshuffle;cnt++){
    r1=1+rand()%N_train;
    r2=1+rand()%N_train;
    train_matrix->get_row_Vector(r1,1,N_input,tmptrv1);
    train_matrix->get_row_Vector(r2,1,N_input,tmptrv2);

    t1=(int)train_target->getelt(r1,1);
    t2=(int)train_target->getelt(r2,1);

    train_matrix->set_row_Vector(r1,1,N_input,tmptrv2);
    train_matrix->set_row_Vector(r2,1,N_input,tmptrv1);

    train_target->setelt(r2,1,t1);
```

```
    train_target->setelt(r1,1,t2);
  }
}

void wine::init_parameters()
{
int seedgen;
int seed_array[100];
int seed_cnt=0;
int i,j;

  seedgen=time(NULL)%1000;
  for(i=0;i<seedgen;i++)
    rand();

  for(i=0;i<100;i++)
    seed_array[i]=rand();

   delay->initmatrix(0.0);
   wght->fill_random(seed_array[seed_cnt++],0.3,0.5);
}


void wine::start()
{
  init();
  createNN();
  acquire_data();
  preprocess_data();

  select_data();

  shuffle_trainset();
  init_parameters();

  tce=35;
  tcpd=15;
  ffact=0.15;

  tolerance=0.001;

  set_threshold_fixed(N_input*0.5*0.5);

  test_train_set("train_set_output0.txt");
  test_spikeNN("test_set_output0.txt");

  max_epocs=25;
  sigma_w0=5;
  learning_rate=0.02;

  order_by_position=1;
  latteral_effect=1;
  vary_neighborhood=1;

  shuffle_trainset();
  train_spikeNN();
  test_train_set("train_set_output1.txt");
  test_spikeNN("test_set_output1.txt");
```

```
}

void wine::acquire_data()
{

  matrix source_matrix(N_sourceRow,N_sourceCol);
  Vector svect(N_attrib);

  source_matrix.read_matrix_from_file("D:/DataSets/wine/wine.data");

  for(int i=1;i<=N_sourceRow;i++){
      source_matrix.get_row_Vector(i,2,N_sourceCol,svect);
      source_data->set_row_Vector(i,1,N_attrib,svect);
      source_target->setelt(i,source_matrix.getelt(i,1));
  }
}
```

A.2.6 Code for clustering Iris data set with SODA_SNN

```
#ifndef IRIS_H
#define IRIS_H


#include "spikenn.h"

class iris: public spikeNN
{
private:

int N_sourceRow, N_sourceCol;

public:
iris();
~iris();
void init();
void acquire_data();
void preprocess_data();
void shuffle_trainset();
void init_parameters();
void start();
};

#endif

#include "iris.h"
#include <math.h>
# include <iostream>
# include <stdlib.h>
# include <time.h>
# include <fstream>

using namespace std;

iris::iris()
{
}
```

```
iris::~iris()
{
}

void iris::init()
{

  N_sourceRow=150;
  N_sourceCol=5;

  N_record=150;
  N_attrib=4;
  N_bias=0;
  N_target_attrib=1;
  N_train=120;
  N_test=30;

  N_input=N_attrib+N_bias;
  N_rows=5;
  N_cols=5;
  N_output=N_rows*N_cols;

  N_class=3;

  dt=1;
  timestep=1;
  twindow=65;
  early_fire=35;
  late_fire=50;
  twindow_input=30;

  tolerance=0.001;
  maxw=1.0;
  minw=0.0;
  maxd=twindow_input;
  mind=0.0;
}

void iris::init_parameters()
{
int seedgen;
int seed_array[100];
int seed_cnt=0;
int i,j;


  seedgen=time(NULL)%1000;
  for(i=0;i<seedgen;i++)
    rand();

  for(i=0;i<100;i++)
    seed_array[i]=rand();

  wght->fill_random(seed_array[seed_cnt++],0.4,0.6);

  delay->fill_random(seed_array[seed_cnt++],0.0,1.0);
```

```
    for(i=1;i<=N_input;i++)
            for(j=1;j<=N_output;j++)
           delay->setelt
               (i,j,10+int(delay->getelt(i,j)*10));
}

void iris::start()
{
  init();
  createNN();
  acquire_data();
  preprocess_data();

  select_data();

  shuffle_trainset();
  init_parameters();

  tce=5;
  tcpd=25;
  dbias=0.05;
  shift=2;

  // ++++++++++++++++Initial output+++++++++

  set_threshold_fixed(N_input*0.5*0.5);

  test_train_set("train_set_output0.txt");
  test_spikeNN("test_set_output0.txt");

  // ++++++++++++++Training Phase 1 +++++++

  max_epocs=20;
  sigma_w0=5;

  min_threshold=(N_input)*0.5*0.7;
  max_threshold=(N_input)*0.5*0.8;

  order_by_position=1;
  vary_neighborhood=1;
  latteral_effect=1;

  train_spikeNN();

  test_train_set("train_set_output1.txt");
  test_spikeNN("test_set_output1.txt");

  // ++++++++++++End Training Phase 1 ++++++++

}

void iris::acquire_data()
{

  matrix source_matrix(N_sourceRow,N_sourceCol);
  Vector svect(N_attrib);

  source_matrix.read_matrix_from_file
```

```
        ("D:/DataSets/iris/iris_data.txt");

  for(int i=1;i<=N_sourceRow;i++){
      source_matrix.get_row_Vector(i,1,N_attrib,svect);
      source_data->set_row_Vector(i,1,N_attrib,svect);
      source_target->
        setelt(i,source_matrix.getelt(i,N_sourceCol));
  }
}
```

A.3 Source code for implementing SDA_SNN

```
#include "matrix.h"
#include "vector.h"

#ifndef SPIKENN_H
#define SPIKENN_H

class spikeNN{
protected:

  int N_record,N_attrib,N_target_attrib;
  int N_train,N_test,N_class;
  int max_epocs, cont_learn;

  int N_input,N_output,N_popuNeuron,N_rows,N_cols;

  matrix *source_data;
  Vector *source_target;
  matrix *processed_data;

  matrix *train_matrix,*train_target;
  matrix *test_matrix,*test_target;

  double maxw, minw, maxd, mind, dbias;
  Vector *threshold;
  Vector *psp_max;

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

  double learning_rate;
  double max_threshold, min_threshold;
  Vector *mx_threshold, *mn_threshold;

  double total_error, min_error,tolerance, ctol, shift;
  double total_dlay_change, prev_dlay_change;
  double nei_poten, nei_dep;
  int precision;
  Vector *inpv, *tv, *test_data_count;

  int update_all, order_network, setthreshold;

  matrix *wght;
  matrix *dwght;
  double tc;
```

```
  double tcpd;
  matrix *delay;
  matrix *ddelay;

  matrix *sp;
  Vector *np;
  Vector *op;
  Vector *exact_op;

//Vector to store the output of the network
        Vector *opy;
//Vector to store winners of each group
        Vector *row_winner;
        Vector *errorv;
        Vector *coincidence_measure;

  matrix *test_output;
  matrix *test_winner;
  matrix *train_winner;

public:
  spikeNN();
  ~spikeNN();
  void createNN();
  void init_spikeNN();
  void init_parameters();

  void select_data();

  void set_threshold_fixed(double);
  double spike_response(int t);
  void update_synapse_potential(int);
  void update_output(int);
  void simulate_SNN();

//Method to find the output of the network
  void process_output();

  void find_winner();
  void find_exact_winner();
//Method to find the winner in the group i
  int local_winner(int i);

  void compute_error(int);
  double error();
  void compute_max_psp();
  void compute_coincidence_measure(int target);

  void train_spikeNN_order();
  void train_spikeNN_tune();
  void train_spikeNN_finetune();
  void test_train_set(const char *opt_test);
  void test_spikeNN(const char *opt_train);

  void update_dlay(int);
  void update_weight();

  double error_train_set();
```

```
  double wght_change();
  double dlay_change();

  void store_input_effect(int);

  void store_output(const char* outputfile, int type);

  void store_test_output();
  double test_train_set();

};


#endif


#include "spikenn.h"
# include <iostream>
# include <math.h>
# include <stdlib.h>
# include <time.h>
# include <fstream>
#include "utils.h"
using namespace std;


spikeNN::spikeNN()
{

}

spikeNN::~spikeNN()
{
}

void spikeNN::createNN()
{

  source_data=new matrix(N_record,N_attrib);
  source_target=new Vector(N_record);
  processed_data=new matrix(N_record, N_input);
  train_matrix=new matrix(N_train,N_input);
  train_target=new matrix(N_train,N_target_attrib);
  test_matrix=new matrix(N_test,N_input);
  test_target=new matrix(N_test,N_target_attrib);
  test_data_count=new Vector(N_class);
  inpv =new Vector(N_input);
  tv=new Vector(N_class);

  wght =new matrix(N_input,N_output);
  dwght =new matrix(N_input,N_output);
  delay=new matrix(N_input,N_output);
  ddelay=new matrix(N_input,N_output);
  sp=new matrix(N_input,N_output);
  threshold=new Vector(N_output);
  psp_max=new Vector(N_output);
  np=new Vector(N_output);
  op=new Vector(N_output);
```

```
  exact_op=new Vector(N_output);

//vector to store the output of the network
  opy=new Vector(N_rows);
  row_winner=new Vector(N_rows);
  errorv=new Vector(N_rows);
  coincidence_measure=new Vector(N_output);
  mx_threshold=new Vector(N_rows);
  mn_threshold=new Vector(N_rows);

}

void spikeNN::process_output()
{
int index, wi;
double wt;

        for(int i=1;i<=N_rows;i++){
                index=(i-1)*N_cols+1;
                wt=twindow;
                wi=0;
                for(int j=index;j<index+N_cols;j++){
                        if (wt>op->getelt(j)){
                                wt=op->getelt(j);
                                wi=j;
                        }
                        opy->setelt(i,wt);
                        row_winner->setelt(i,wi);
                }
        }
}

int spikeNN::local_winner(int target)
{
 double lwin_t=twindow;
 int index=(target-1)*N_cols+1;
 int lw=0;

 for(int j=index;j<index+N_cols;j++){
         if  (lwin_t>=exact_op->getelt(j)) {
                 lw=j;
                 lwin_t=exact_op->getelt(j);
         }
 }
 return lw;
}

void spikeNN::compute_error(int target)
{
int wi;
double wt, diff, e, t_target;

                wt=twindow;
                wi=0;
                for(int i=1;i<=N_rows;i++)
                        if (wt>opy->getelt(i)){
                                wt=opy->getelt(i);
                                wi=i;
```

```
                    }
                diff=twindow-wt;
                t_target=opy->getelt(target);

                if (diff){
                        for(i=1;i<=N_rows;i++){
                                e=(t_target-opy->getelt(i))/diff;
                                errorv->setelt(i,1-e);
                        }
                }
}

double spikeNN::error()
{
double e, tte;

tte=0.0;
        for(int i=1;i<=N_rows;i++){
                e=errorv->getelt(i);
                tte+=mod(e);
        }
return tte;
}

void spikeNN::simulate_SNN()
{
    int nst=(int)inpv->min();
    do{
       nst=nst+dt;
        update_synapse_potential(nst);
        update_output(nst);
      }while(nst<twindow);
        process_output();

}

void  spikeNN::train_spikeNN_order()
{

  int epoc, target, winner_row, winner_col, index, correct=0;

  double total_error, wop;

  epoc=0;
  total_dlay_change=0.0;
  prev_dlay_change=1000.0;
  errorv->initialise(1.0);
  coincidence_measure->initialise(1.0);
  do{
    epoc++;
    cont_learn=1;
    nei_poten=0.0; nei_dep=0.0;
                total_error=0.0;
                correct=0;

    set_threshold(epoc);

    for(int r=1;r<=N_train;r++){
```

```
        init_spikeNN();
        train_matrix->get_row_Vector(r,1,N_input,*inpv);

            target=train_target->getelt(r,1);

        winner_found=0;
        errorv->initialise(1.0);

        simulate_SNN();
            find_exact_winner();

        if (winner_found){
                    winner_row=winner/N_cols+1;
                    winner_col=winner%N_cols;
                    if (winner_col==0){
                            winner_col=N_cols;
                            winner_row--;
                    }
                    if ((winner_row==target)&&(winning_time<twindow))
                           correct++;
                    compute_coincidence_measure(target);

                    index=(target-1)*N_cols+1;
                            for(int j=index;j<index+N_cols;j++)
                                    update_dlay(j);

            }
      total_dlay_change+=dlay_change();
            total_error+=error();

      }
      cout<<"epoc :"<<epoc<<endl;
      cout<<"Neigh_poten :"<<nei_poten<<endl;
      cout<<"Neigh_depression :"<<nei_dep<<endl;
      cout<<"Total :"<<total_dlay_change<<endl;
            cout<<"Total_error :"<<(float)N_train-correct<<endl;

     cont_learn=((((N_train-correct)/N_train)>tolerance)||
                ((prev_dlay_change-total_dlay_change)>ctol));

     prev_dlay_change=total_dlay_change;
     //cont_learn=1;
   }while((epoc<max_epocs) && (cont_learn));
}

void  spikeNN::train_spikeNN_tune()
{

   int epoc, target, winner_row, winner_col, rowwn, index, correct=0;

   double total_error, wop;
   int lw;
   total_dlay_change=0.0;
   prev_dlay_change=1000.0;
   epoc=0;
   total_dlay_change=0.0;
   errorv->initialise(1.0);
   coincidence_measure->initialise(1.0);
```

```
  do{
    epoc++;
    cont_learn=1;
    nei_poten=0.0; nei_dep=0.0; dep=0.0;

        total_error=0.0;
        correct=0;

    set_threshold(epoc);

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

            target=train_target->getelt(r,1);

      winner_found=0;
      errorv->initialise(1.0);

      simulate_SNN();
            find_exact_winner();

      if (winner_found){
                  winner_row=winner/N_cols+1;
                  winner_col=winner%N_cols;
                  if (winner_col==0){
                        winner_col=N_cols;
                        winner_row--;
                   }

                   if ((winner_row==target)&&(winning_time<twindow))
                         correct++;
                   lw=local_winner(target);
                        compute_coincidence_measure(target);
                        update_dlay(lw);

       total_dlay_change+=dlay_change();
             total_error+=error();

    }
    cout<<"epoc :"<<epoc<<endl;
    cout<<"Neigh_poten :"<<nei_poten<<endl;
    cout<<"Neigh_depression :"<<nei_dep<<endl;
    cout<<"Total :"<<total_dlay_change<<endl;
          cout<<"Total_error:"<<(float)N_train-correct<<endl;

   cont_learn=((((N_train-correct)/N_train)>tolerance)||
                ((prev_dlay_change-total_dlay_change)>ctol));
   prev_dlay_change=total_dlay_change;
   cont_learn=1;
  }while((epoc<max_epocs) && (cont_learn));
}

#include "spikenn.h"

class control_chart: public spikeNN
{
private:
```

```
int N_sourceRow, N_sourceCol;

public:
control_chart();
~control_chart();
void init();
void acquire_data();
void preprocess_data();
void shuffle_trainset();
void init_parameters();
void start();
};

void control_chart::start()
{
  init();
  createNN();
  acquire_data();
  preprocess_data();

  test_data_count->setelt(1,83);// Out of 250
  test_data_count->setelt(2,83);// Out of 250
  test_data_count->setelt(3,83);// Out of 250
  test_data_count->setelt(4,83);// Out of 250
  test_data_count->setelt(5,83);// Out of 250
  test_data_count->setelt(6,83);// Out of 250

  select_data();

  shuffle_trainset();
  init_parameters();

  // ++++++++++++++++Initial output++++++++++++++++++++++++++++++++
  tc=5;
  tcpd=25;
  dbias=0.12;
  shift=2;

  set_threshold_fixed((N_input)*0.5*0.55);

  test_train_set("train_set_output0.txt");
  test_spikeNN("test_set_output0.txt");

  // ++++++++++++++++Training Phase 1 ++++++++++++++++++++++++++++++

  max_epocs=20;
  learning_rate=0.2;
  min_threshold=(N_input)*0.5*0.7;
  max_threshold=(N_input)*0.5*0.78;

  //shuffle_trainset();
  train_spikeNN_order();
  test_train_set("train_set_output1.txt");
  test_spikeNN("test_set_output1.txt");

  // ++++++++++++End Traing Phase 1 ++++++++++++++++++++++++++++++++
```

```
  // ++++++++++++Training Phase 2 ++++++++++++++++++++++++++++++

  tc=5;
  tcpd=25;
  dbias=0.1;
  shift=2;

  max_epocs=10;
  learning_rate=0.2;
  min_threshold=(N_input)*0.5*0.78;
  max_threshold=(N_input)*0.5*0.83;

  shuffle_trainset();
  train_spikeNN_tune();
  test_train_set("train_set_output2.txt");
  test_spikeNN("test_set_output2.txt");

  // ++++++++++++End Training Phase 2 ++++++

}
```

Appendix B

Data sets

The training and testing data used for analysing the proposed models are given in the CD Rom attached to the thesis. The text files containing the data are separated into three folders corresponding to the appropriate model. The files are named in the form of *SNN model name_Data base name_training/Testing_trial number*. For example data set used to train SOWA_SNN in trial 1 would be in the folder *SOWA_SNN* and named as *SOWA_Iris_Train_Trial1.txt*.